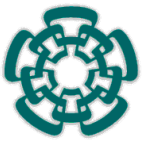


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Zacatenco

Departamento de Ingeniería Eléctrica
Sección de Computación

Implementación Eficiente en FPGA del Modo CCM usando AES

Tesis que presenta:
Emmanuel López Trejo

para obtener el Grado de:
Maestro en Ciencias

en la Especialidad de:
Ingeniería Eléctrica

Opción:
Computación

Director de la Tesis:

Dr. Francisco Rodríguez Henríquez

Ciudad de México, México. Septiembre 2005

A **Dios** sea la gloria.
Gracias te doy Señor
por permitirme ver
realizado este sueño.

A mi madre adorada,
por su gran e inmenso amor,
que sin duda jamás podré pagar,
por ser simplemente mi mejor amiga
y la persona que más quiero en este mundo.

A mi hermanita preciosa,
por su gran cariño y amor,
y por todos los grandes
momentos que hemos vivido juntos.

A Paty, mi gran amor,
por todo su amor y cariño
que han hecho de mi vida algo
maravilloso.

Agradecimientos

Gracias le doy primeramente a Dios por permitirme culminar con éxito este sueño, así como por las grandes bendiciones que de su mano he recibido a lo largo de mi vida. Sé que sin su ayuda nada hubiese podido conseguir.

Agradezco a mi mamá por su infinito amor, por su paciencia y apoyo incondicional, sin los cuales gran parte de lo que soy no sería posible. Gracias mamá por estar siempre a mi lado y por el aliento constante en los momentos más difíciles, por tu disposición a escuchar mis problemas, frustraciones, sueños, anhelos, etc. Te agradezco por escuchar mis pláticas sobre los problemas que tenía que resolver, aunque no entendieras de lo que te estaba hablando. Te doy las gracias por las palabras de motivación, de orientación y de confianza sin las cuales mucho de esto no sería posible. Pero por sobre todo, te doy las gracias por traerme al mundo, aun cuando casi te costó la vida y porque cada día me entregas una parte importante de tu ser. Esta tesis es para ti, porque sé que es la culminación del excelente trabajo que llevas realizando por más de 24 años. Gracias, mil gracias.

Gracias a mi hermanita adorada por su apoyo y paciencia en los momentos difíciles, y por tener siempre la frase adecuada para hacerme reír. Gracias por ser como eres y por todo el amor que me brindas. Te quiero mucho no lo olvides.

Agradezco a Paty por su apoyo incondicional durante este tiempo y porque ha traído una nueva luz a mi vida. Gracias por todos los bellos momentos que hemos vivido juntos y por las constantes palabras de ánimo que me motivaron a seguir adelante.

Una gratitud especial al resto de mi familia que en mayor o menor medida contribuyeron para que todo esto fuera posible. Gracias por su amor y cariño, forman una parte muy especial en mi vida.

Agradezco al Consejo Nacional de Ciencia y Tecnología, por el apoyo económico recibido durante la realización de mis estudios de Maestría mediante el otorgamiento de la beca-crédito, así como a través del proyecto número 45306.

Una gratitud especial a mis profesores en el CINVESTAV, cuyo conocimiento contribuyó a ampliar y consolidar mi visión sobre las ciencias de la computación. Sin duda no podré pagar sus valiosas enseñanzas.

Agradezco al Dr. Francisco Rodríguez Henríquez por su tiempo y sus innumerables comentarios para el desarrollo de esta tesis, así como por gran paciencia y amplio conocimiento para llevarla a buen término.

Gracias también a los Dres. Arturo Díaz Pérez y Carlos Artemio Coello Coello, por sus valiosos comentarios para el enriquecimiento de este trabajo. Gracias por el tiempo que invirtieron en la revisión y corrección de este documento.

No puedo olvidar a mis compañeros y amigos, los cuales también contribuyeron para hacer más agradable y sencilla mi estancia en el CINVESTAV. Gracias a cada uno de ellos por su ayuda, orientación y amistad.

Finalmente, pero no menos importante, gracias a Sofía Reza por su apoyo y ayuda para resolver los trámites académicos, gracias por la gran paciencia al desarrollar su trabajo.

Resumen

La tecnología de redes inalámbricas ha tenido un crecimiento exponencial en los últimos años, lo que ha incrementado la necesidad de contar con esquemas de seguridad que brinden una adecuada protección de los datos. Diversos algoritmos han sido diseñados para tal fin, sin embargo, el utilizado en el estándar IEEE 802.11b, el WEP, ha sido vulnerado en múltiples ocasiones, lo que hace necesario el desarrollo de nuevas alternativas para proteger la información que se transmite a través de este tipo de redes.

En años recientes, nuevos esquemas se han propuesto para sustituir al WEP, siendo el modo de operación CCM el candidato más fuerte para tomar el lugar como el esquema de seguridad en las redes inalámbricas. En esta tesis, presentamos la implementación de dicho modo de operación en un FPGA y brindamos las ideas básicas para conseguir una implementación eficiente. El diseño presentado en este trabajo, consigue una eficiencia altamente competitiva con las aplicaciones comerciales, así como con implementaciones académicas reportadas en la literatura abierta.

Presentamos el análisis de los resultados obtenidos y damos una proyección de su utilidad comparada con aplicaciones comerciales disponibles actualmente.

Abstract

Over the last years Wireless technology has grown exponentially, which has increased the necessity of suitable security schemes that provides an adequate protection of the information. Many algorithms had been designed for this purpose, however, the most widely used in the IEEE 802.11b standard, the WEP, has been broken in multiple occasions, making necessary the development of new alternatives to protect the information that is broadcast in this type of networks.

In recent years, new schemes had been proposed to substitute the WEP, being the CCM mode of operation the most promising scheme for becoming the new security protocol in wireless network environments. In this Thesis, we present the FPGA implementation of the CCM mode of operation and we give the basic ideas in order to get an efficient implementation. The design presented in this work, achieves a great performance compared with commercial applications, and also with the academic ones reported in the open literature.

We present a rigorous analysis of the results obtained giving a projection of its usefulness compared with commercial applications currently available.

Índice general

Índice de figuras

Índice de cuadros

Introducción

Conforme la tecnlogía de cómputo móvil se hace más accesible a una gran cantidad de personas, el desarrollo de aplicaciones para este tipo de plataformas se ha convertido en un gran campo de oportunidad para establecerse en el liderato de este creciente mercado.

Debido a la naturaleza inherente a la computación móvil, diversos factores influyen en la calidad de los servicios que se desarrollan para estos dispositivos portátiles, entre los cuales encontramos el poco poder de cómputo que tienen comparados con las computadoras de escritorio actuales. Así mismo, se encuentra la necesidad de acceder a la información desde prácticamente cualquier lugar.

Cada vez es más frecuente el hacer uso de redes inalámbricas, ya sea para conectarse a la Internet, hacer transferencias de información o bien para realizar operaciones bancarias. Las facilidades otorgadas por las redes inalámbricas han cambiado nuestra forma de vida, pues nos proporcionan una mejor manera de acceder a la información importante de acuerdo a nuestro estilo de vida. Así mismo, nos brindan la ventaja de movilidad, lo que nos permite estar comunicados casi en cualquier parte de la ciudad.

Sin embargo, las redes inalámbricas son más suceptibles a ataques que ponen en riesgo la integridad de la información transmitida. Debido a que el canal de comunicación no está restringido a un cable, cualquier persona es capaz (al menos en teoría) de robar o alterar los datos.

Con el gran impacto que están teniendo las redes inalámbricas, la protección de la información que es transmitida a través de este tipo de redes, se ha convertido en uno de los objetivos primordiales de los desarrolladores de aplicaciones para dispositivos móviles, los cuales, en su afán de proveer seguridad a sus productos, han diseñado múltiples esquemas que garanticen que los datos son protegidos de manera adecuada.

Diversas soluciones se han desarrollado para asegurar la protección de la información. Sin embargo, muchas de ellas, no consiguen el objetivo de brindar la seguridad de los datos transmitidos, ya sea porque tienen deficiencias en su diseño, o porque resultan imprácticas para las plataformas móviles.

Gracias al abaratamiento de la tecnología, hoy en día, el número de dispositivos móvi-

les se ha incrementado, lo que genera una necesidad creciente de desarrollar esquemas que brinden un adecuado nivel de protección de la información. Diversos esquemas se han desarrollado con ese fin, siendo, en tiempo recientes, el más destacado el modo de operación CCM, el cual se ha posicionado como una alternativa competitiva para proveer la seguridad de la información.

Actualmente es posible utilizar el protocolo CCMP, basado en el modo de operación CCM, en algunas tarjetas inalámbricas comerciales. Dichos productos realizan las operaciones necesarias a través de implementaciones en software. El objetivo de esta tesis es determinar la conveniencia de implementar el modo de operación CCM en una arquitectura de hardware reconfigurable, con la finalidad de observar el incremento en la velocidad de procesamiento.

En esta tesis, presentamos la implementación de dicho modo de operación en un FPGA, para lo cual utilizamos al AES como cifrador por bloques. El objetivo primordial de la tesis es obtener una implementación que sea eficiente y competitiva con las aplicaciones comerciales existentes.

La tesis está organizada de la siguiente manera: En el capítulo 1, se presenta una descripción de la seguridad en los dispositivos móviles. Se describen los ataques a los que están propensas las redes inalámbricas y se enlistan los servicios que deben cumplir para evitar dichos ataques. Se presentan además las características del estándar IEEE 802.11, y se dan las bases teóricas del funcionamiento de su esquema de seguridad, así como las deficiencias del mismo. Incluimos también las nuevas propuestas de seguridad para este tipo de redes y se describen los dispositivos de hardware reconfigurable, así como las propuestas de seguridad disponibles para los mismos.

El capítulo 2 describe al cifrador por bloques AES, donde se da una breve semblanza histórica y se explica la teoría de su funcionamiento. Cerramos el capítulo con una descripción de algunas ideas para optimizar su rendimiento. En el capítulo 3 se da la teoría correspondiente al modo de operación CCM, se describen las entradas y los parámetros de dicho modo y se explican los procesos que lo componen. Concluimos dicho capítulo con la descripción de algunas pruebas realizadas de dicho modo con tarjetas comerciales.

En el capítulo 4 nos enfocamos a la descripción de la manera en que se realizó la implementación, tanto del AES, como del modo de operación CCM, en un FPGA. Mostramos la arquitectura diseñada y la manera en que fue implementada en el lenguaje de programación VHDL. El capítulo 5 contiene los resultados obtenidos por la implementación, mostrando tablas comparativas con algunas implementaciones reportadas en la literatura abierta. Finalmente las conclusiones son presentadas en el capítulo 6.

Capítulo 1

Seguridad en dispositivos de cómputo restringido

La tecnología de cómputo móvil se encuentra en una fase de desarrollo exponencial, donde cada día la necesidad de este tipo de tecnología más confiable está creciendo de manera impresionante. Las grandes corporaciones desarrollan aplicaciones que proveen beneficios sustanciales para los negocios, lo que ha provocado que los usuarios adopten rápidamente dichas aplicaciones.

Con el rápido desarrollo de dispositivos móviles cada vez más poderosos, sus aplicaciones y las redes inalámbricas están proporcionando el acceso a datos desde casi cualquier sitio. En poco tiempo la tecnología inalámbrica ha madurado hasta el punto de encontrarse lista para una adopción a gran escala [16].

En este capítulo se presentan los aspectos referentes a la seguridad en los dispositivos de cómputo restringido y se describen las diversas soluciones enfocadas a brindar un nivel de seguridad aceptable para la tecnología móvil e inalámbrica.

1.1. Seguridad en dispositivos portátiles

Una de las mayores preocupaciones al desarrollar soluciones de tecnología móvil e inalámbrica es la seguridad de la información. Garantizar la protección adecuada de los datos en un medio alámbrico es difícil; si se añade la transmisión inalámbrica y el almacenamiento en dispositivos de procesamiento limitado, la tarea se hace aún más compleja.

Los dispositivos móviles, tales como los teléfonos celulares, los asistentes digitales personales (*PDA* por sus siglas en inglés) y los localizadores, son inherentemente más inseguros que sus contrapartes fijas. Esto en parte se debe a su limitado ancho de banda, memoria

y capacidad de procesamiento. Otra razón es que envían sus datos a través del aire donde cualquiera con la tecnología adecuada puede interceptarlos [21].

La seguridad es la combinación de procesos, procedimientos y sistemas usados para asegurar la confidencialidad, integridad y disponibilidad de la información [21].

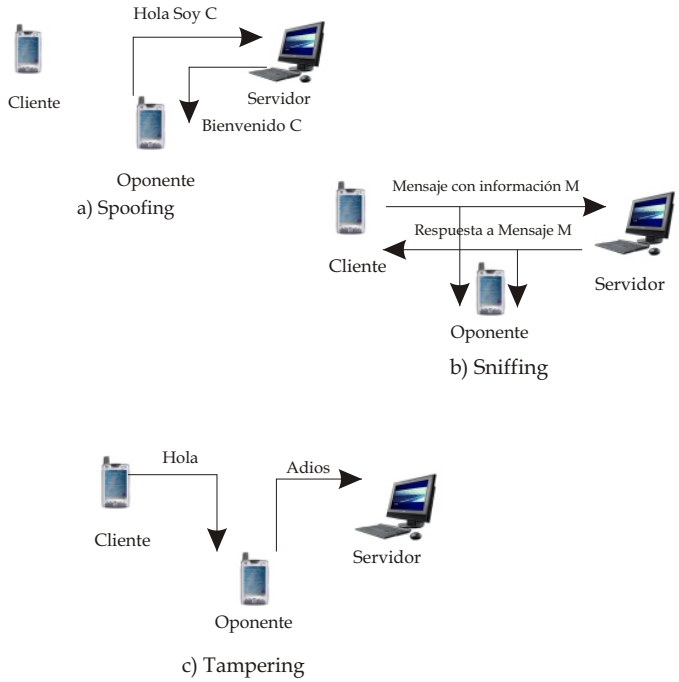


Figura 1.1: Diversos ataques a las redes inalámbricas

Existen algunos ataques a los que toda transmisión inalámbrica se encuentra expuesta, algunos de ellos son ilustrados en la figura 1.1, los cuales serán explicados en los párrafos siguientes.

Suplantación

La suplantación es el intento de una entidad externa de obtener acceso no autorizado a una aplicación o a un sistema. Si el oponente obtiene el acceso, puede crear mensajes falsos en su intento de conseguir más información y así poder infiltrarse a otras partes del sistema. La suplantación es un problema de seguridad importante en las conexiones de Internet, debido a que un cliente puede creer que se comunica con la página de su banco y proporcionar datos importantes que le permitirán a algún intruso llevar a cabo un ataque.

Espionaje

El espionaje es una técnica usada para monitorear el flujo de información en una red. Aunque puede ser utilizado para buenos propósitos, suele asociarse más con la copia no autorizada de datos en la red [16]. Entidades no autorizadas son capaces de obtener información que les permitirá causar daño a las aplicaciones de los usuarios. Este tipo de ataque es muy peligroso pues es fácil de hacer y difícil de detectar.

Modificación de datos

También conocida como la alteración de la integridad, involucra la modificación maliciosa de los datos. Frecuentemente esto implica la interceptación de la información en una transmisión, aunque también puede realizarse en datos almacenados en un servidor o cliente. El objetivo es hacer pasar la versión modificada por una versión original.

Debido a la gran importancia de la seguridad en las aplicaciones que se desarrollan en dispositivos móviles, es indispensable conocer los servicios que dichas aplicaciones necesitan satisfacer con el objetivo de evitar los ataques descritos anteriormente y así ser consideradas seguras. Dichos servicios son ilustrados en la figura 1.2.

Los cuatro servicios principales que necesitan ser cubiertos son: [39]

- *Autenticación:* El receptor desea asegurarse que el emisor autorizado es quien efectivamente envió el mensaje que recibió. En este servicio se encuentran incluidos los esquemas de identificación y los protocolos de claves de acceso. Existen dos tipos de autenticación: la *autenticación de la entidad* y la *autenticación del origen de datos*. Frecuentemente el término *identificación* es utilizado para referirse a la autenticación de la entidad, la cual se ocupa de verificar la identidad de los participantes en una comunicación. La autenticación del origen de datos se centra en relacionar la información sobre el creador y hora de creación de los datos. La figura 1.2(a) ilustra la autenticación por medio de la respuesta de un reto, el cual, normalmente se basa en cifrar un mensaje aleatorio utilizando una llave que sólo el emisor verdadero conoce. Dicho emisor, descifra el mensaje y lo envía al receptor para demostrar que es quien dice ser.

- *Integridad de la información:* El receptor quiere estar seguro que el mensaje del emisor no ha sido alterado. Por ejemplo, pueden ocurrir errores en la transmisión, o bien, puede ser que una tercera persona haya interceptado el mensaje y que maliciosamente lo haya alterado antes que el destinatario lo reciba. Existe una amplia variedad de métodos criptográficos para detectar la modificación maliciosa o accidental de mensajes. De entre ellos, las funciones Hash suelen ser el bloque fundamental para proveer este servicio (véase Apéndice A.1). En la figura 1.2(b) se muestra la integridad mediante el uso de una función Hash que es verificada por el receptor para comprobar que el mensaje no sufrió modificaciones.

- *Confidencialidad:* Un atacante no debe ser capaz de leer un mensaje intercambiado por el emisor y el receptor. Las principales herramientas son los algoritmos de cifrado y descifrado. La figura 1.2(c) ilustra la confidencialidad mediante el uso de una función de cifrado ($E()$) del mensaje (M) a través de una llave (K).
- *No repudio:* El emisor no debe ser capaz de negar que él no envió un determinado mensaje. El no repudio es particularmente importante en el comercio electrónico, donde es fundamental que un cliente no pueda negar que haya autorizado el pago por un producto. La figura 1.2(d) muestra el no repudio a través del uso de una firma digital, las cuales son explicadas más adelante en este capítulo.

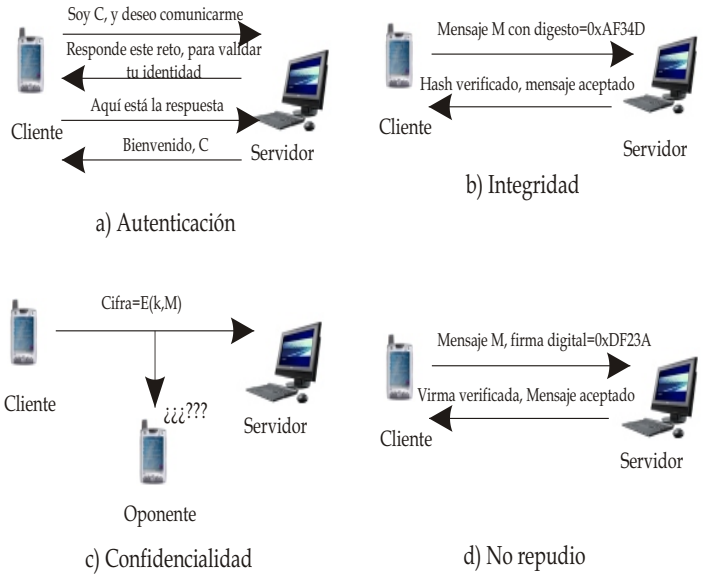


Figura 1.2: Los diferentes servicios a satisfacer

Cada diseñador de aplicaciones de tecnología móvil debe determinar qué servicios de seguridad cubrir, puesto que no todas las aplicaciones necesitan de todos los servicios. Además, es necesario que los diseñadores consideren la capacidad de cómputo de la plataforma de desarrollo, debido a que algunas de las soluciones propuestas para cumplir con dichos servicios involucran una gran cantidad de cálculos y requieren de un amplio poder de cómputo.

Como se ha mencionado en esta sección, la seguridad en las aplicaciones móviles es importante, sin embargo, debido a las limitaciones en los dispositivos actuales, se hace necesario el desarrollo de nuevas técnicas que permitan garantizar un nivel de seguridad adecuado. El desarrollo de dichas técnicas sigue avanzando cada día con la construcción de dispositivos móviles cada vez más poderosos.

Tecnología para la seguridad

Una vez que se han explicado los diferentes ataques a los que está expuesta la información, se dará un panorama de la tecnología y herramientas matemáticas disponibles para las aplicaciones móviles con la finalidad de disminuir los riesgos al desarrollar aplicaciones seguras.

- **Criptografía:** El objetivo básico de la criptografía es permitir a dos entidades comunicarse a través de un canal seguro, sin que una tercera entidad sea capaz de entender lo que está siendo transmitido. Esta capacidad es necesaria para garantizar un medio seguro, por lo que se ocupa de todos los aspectos de la transferencia de datos, incluyendo autenticación, firmas digitales y cifrado. Superficialmente la criptografía es un concepto simple, pero en realidad es bastante compleja, especialmente para implementaciones a gran escala en dispositivo móviles. De manera general existen dos tipos de criptografía:

  - *S*imétrica: Basada en una llave común que es compartida por el emisor y el receptor, denominada llave secreta, la cual es útil tanto para cifrar como para descifrar. Este tipo de criptografía es simple y eficiente de implementar, sin embargo su principal problema radica en la dificultad para distribuir las llaves entre los participantes.
  - *A*lgoritmos de llave Pública: A diferencia de la simétrica, este tipo de algoritmos requieren de dos llaves, una denominada llave pública y otra llamada llave privada. Ambas llaves están relacionadas matemáticamente, y es computacionalmente imposible conocer la llave privada a partir de la pública. Podría parecer que los algoritmos de llave pública hacen obsoleta a la criptografía simétrica, sin embargo su flexibilidad tiene un costo elevado, y radica en la gran cantidad de cálculos que son necesarios para su implementación, los cuales son normalmente más costosos que los simétricos. Debido a lo anterior este tipo de algoritmos no son usados para cifrar grandes cantidades de información, en su lugar, típicamente se utilizan en aplicaciones donde únicamente se requiere cifrar cantidades pequeñas de datos, por ejemplo firmas digitales y/o para enviar las llaves que serán utilizadas en un algoritmo simétrico (conocidas como llaves de sesión). Para una mejor explicación de ambos algoritmos véase [17, 29].

- **Firmas digitales:** Las firmas digitales son usadas para verificar que un mensaje proviene verdaderamente del emisor que dice enviarlo. Están basadas en la noción de que sólo el creador de la firma tiene conocimiento de la llave privada y puede ser verificada usando la llave pública que corresponda. La firma digital es creada mediante el uso de una función hash (véase el Apéndice **??**) para procesar el mensaje, firmando el resultado de la misma, lo cual se concatena a la información que se envía, agregándose una marca de tiempo y/o cualquier otra información requerida. El conjunto de información formada es cifrada con la llave privada del emisor usando un algoritmo de llave pública. El resultado es la firma digital. La función hash es útil debido a que si el mensaje es alterado, producirá un cambio en la salida de la función hash utilizada para verificar

la integridad del mensaje, lo que permitirá que el receptor pueda darse cuenta si ocurrió corrupción en los datos. Una propiedad importante de las firmas digitales es que están relacionados con el documento firmado, por lo que no pueden ser falsificadas para firmar otro documento sin el permiso del poseedor de la firma.

- **Certificados digitales:** Los certificados digitales proveen una forma de garantizar que una llave pública pertenece a la entidad que representa y con esto validar una firma digital. Para que esto sea exitoso, el certificado debe ser verificado para confirmar que representa a la entidad que dice (persona u organización). Esto es realizado usando una tercera entidad de confianza denominada Autoridad Certificadora. Entre las autoridades certificadoras más difundidas podemos mencionar a VeriSign, Entrust y Certicom [16]. El usuario puede comprar un certificado digital de dichas autoridades y distribuir su llave pública para realizar transferencia de información. Véase [17, 29] para una explicación más amplia sobre certificados digitales.

- **Infraestructura de llave pública:** La infraestructura de llave pública (PKI por sus siglas en inglés) es el término utilizado para describir a la organización de reglas y sistemas que definen la seguridad de un sistema. El Grupo de Ingeniería de Internet (IETF por sus siglas en inglés) X.509 define PKI como "el conjunto de hardware, software, personas, y procedimientos necesarios para crear, manejar, almacenar, distribuir, y revocar certificados digitales basados en la criptografía de llave pública". Los componentes de la infraestructura de llave pública incluyen [10]:

  - Autoridades certificadoras, las cuales son responsables de otorgar y revocar certificados.
  - Poseedores de certificados digitales otorgados por autoridades certificadoras. Dichos certificados pueden ser utilizados para firmar documentos digitales.
  - Repositorios donde se almacenan certificados así como listas de revocación.
  - Políticas de seguridad que definen las acciones de seguridad tomadas por la organización.

Como puede apreciarse, existen diversos métodos para proteger la información en las aplicaciones móviles, sin embargo el uso de cada uno de ellos depende de las necesidades de las organizaciones, así como de los recursos disponibles para su implementación.

En las siguientes secciones abordaremos de manera específica la tecnología de las redes inalámbricas, así como la seguridad inherente a ella y algunas nuevas propuestas para la protección de la información.

1.1.1. Tecnología de redes inalámbricas

La comunicación inalámbrica es el proceso de transmitir información en un medio electromagnético a distancia a través del aire, en lugar de hacerlo por medio de cables o cualquier otro conducto físico [21].

Los mensajes de una red inalámbrica se transmiten a través del aire utilizando algunos espectros de frecuencia, los cuales son escasos, fuertemente regulados, y frecuentemente son recursos difíciles de conseguir.

Una red inalámbrica es un sistema flexible de comunicación implementado como una extensión, o como una alternativa, para las redes alámbricas. Las redes inalámbricas transmiten y reciben los datos a través del aire utilizando la tecnología de radio frecuencia, por lo que minimizan la necesidad de conexiones alambradas y de este modo combinan la conectividad con la movilidad.

Las redes inalámbricas eliminan el enlace físico a la red, permitiendo a los usuarios una conexión directa con el sistema de distribución sin que sea necesario conectar cables. El decir que una red inalámbrica está libre de cables no es estrictamente correcto, debido a que en una red de este tipo existen uno o más puntos de acceso, los cuales están conectados a la red a través de los cables de datos tradicionales.

Para efectos prácticos podemos dividir a las redes inalámbricas en dos grandes categorías: largo y corto alcance. Las de corto alcance son aquellas en las cuales la transmisión está confinada a un área limitada. En este tipo de redes podemos situar a las redes de área local (LANs por sus siglas en inglés), tales como las encontradas en empresas, escuelas, plantas de manufactura u hogares: así mismo se incluyen las redes de área personal (PANs por sus siglas en inglés), donde computadoras personales se intercomunican a distancias muy cortas. Este tipo de redes típicamente operan en espectros de frecuencia que no se encuentran bajo licencia y que están reservados para uso industrial, científico y médico. Las bandas de frecuencias disponibles varían de país en país, sin embargo la banda más utilizada es la de 2.4 GHz, la cual está disponible en casi todo el planeta. Otras frecuencias utilizadas son 5 GHz y 40 GHz. La disponibilidad de estas frecuencias permite a los usuarios operar una red inalámbrica sin que sea necesario obtener una licencia y por ende sin pagar los derechos de uso de una frecuencia. En la figura 1.3(a) se ilustra una red de corto alcance.

A diferencia de las redes de corto alcance, las de largo alcance están diseñadas para proveer el acceso a la información desde lugares más remotos, y típicamente abarcan una ciudad, un estado o bien un país entero. Las redes de largo alcance más comunes son las denominadas Redes Inalámbricas de Área Amplia (*WWAN* por sus siglas en inglés). Una opción para este tipo de redes son las satelitales, las cuales proveen un servicio considerado global. El esquema genérico de una red de largo alcance se ilustra en la figura 1.3(b) .

Conforme a lo explicado en los párrafos anteriores, podemos encontrar 2 subdivisiones para cada uno de los dos tipos de redes, dichas subdivisiones son explicadas en el cuadro 1.1 [16].

A pesar de que los desarrolladores de aplicaciones no requieren conocer completamente el funcionamiento interno de una determinada tecnología de red inalámbrica, sí es importante que se tomen en cuenta determinadas características como la velocidad, puesto que eso deriva en aplicaciones que utilizan de manera más eficiente los recursos disponibles.

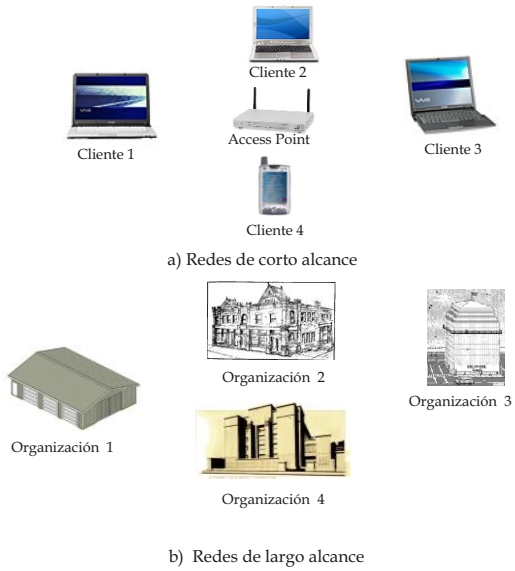


Figura 1.3: Redes de corto y largo alcance

Debido a que existe una gran diversidad de tecnología para redes inalámbricas, el tratar de cubrir todas sería un trabajo muy extenso y fuera de los propósitos de esta tesis, por lo que únicamente nos enfocaremos a las redes inalámbricas de área local (WLANs). El lector interesado puede consultar [16, 21].

Redes Inalámbricas de área Local (WLANs)

Este tipo de redes pertenecen a uno de los segmentos de mayor crecimiento dentro de la industria de las telecomunicaciones. La adopción de estándares por parte de la industria, y el correspondiente desarrollo de múltiples productos para WLANs por parte de los fabricantes más importantes, ha derivado en la implementación de soluciones de WLANs en muchos de los segmentos del mercado, incluyendo pequeñas oficinas, hogares, grandes empresas, plantas de manufactura, y lugares públicos tales como aeropuertos, centros de convenciones, hoteles e incluso cafeterías [16].

Diversos estándares han sido adoptados para usarse con este tipo de redes, por lo que es importante conocer sus especificaciones y la seguridad que brindan. En este trabajo nos enfocaremos al estándar IEEE 802.11, pues es el más utilizado en la actualidad.

El estándar IEEE 802.11

Fue aprobado en julio de 1997, siendo el primer estándar definido para una WLAN. Usa los mismos protocolos de intercambio que Ethernet, pero permite la comunicación sin cables, para lo cual usa una radiofrecuencia para la transmisión de la información. Dentro de este estándar se han desarrollado múltiples versiones, siendo la versión “b” la más popular y más

Tipo de red	**área de cobertura**	**Función**	**Costo asociado**	**Velocidad**	**Tecnología utilizada**
Red inalámbrica de área personal (WPAN)	Típicamente 10m	Reemplazo de cableado, redes personales	Muy bajo	0.1 - 4 Mbps	IrDA, Bluetooth, 802.15
Red inalámbrica de área local (WLAN)	En edificios, escuelas; típicamente 100m	Extensión o alternativa para las redes alámbricas (LAN)	Medio	1-54 Mbps	802.11a, b, g, HIPERLAN/2
Red inalámbrica de área Amplia (WWAN)	Cobertura amplia basada en múltiples repetidores	Extensión de una LAN	Alto	8 Kbps - 2 Mbps	GSM, TDMA, CDMA, GPRS, EDGE, WCDMA
Red Satelital	Cobertura global	Extensión de una LAN	Muy alto	2 Kbps - 19.2 Kbps	TDMA, CDMA, FDMA

Cuadro 1.1: Diferencias entre los tipos de redes inalámbricas disponibles [16]

utilizada. A continuación se explican las diferencias que existen entre las versiones existentes del IEEE 802.11.

- **802.11b/Wi-Fi:** Es el estándar más popular en la familia del 802.11x. La especificación fue aprobada al mismo tiempo que la 802.11a en 1999, pero desde entonces ha obtenido una amplia aceptación en el mercado de las redes inalámbricas. Utiliza el espectro de frecuencia 2.4GHz, el cual tiene la ventaja de estar disponible de forma global para las configuraciones de WLANs. La velocidad máxima de transmisión es de 11 Mbps lo cual sobrepasa a los 10Mbps que es parte del estándar original de Ethernet, lo que convierte al 802.11b en una alternativa real para las LANs. Con la finalidad de facilitar la compatibilidad entre los productos, la alianza Wi-Fi ha creado la certificación denominada Wireless Fidelity (Wi-Fi), la cual garantiza que dicho producto será capaz de interoperar con otros productos que cuenten también con la certificación Wi-Fi. Los mecanismos de seguridad definidos en este estándar son explicados en la sección 1.1.2.

- **802.11a:** Es la alternativa de alta velocidad para el 802.11b, transmite a 5GHz con velocidades de hasta 54Mbps. Debido a la frecuencia y la tecnología de modulación, es

incompatible con las redes basadas en 802.11b. Al ser más complejo de implementar, los productos para este estándar no fueron liberados para el uso comercial hasta el año 2002, lo que provocó que su uso no se extendiera. La alianza Wi-Fi ha incluido una certificación para los productos del 802.11a, con esto trata de generar la confianza que existe para los 802.11b.

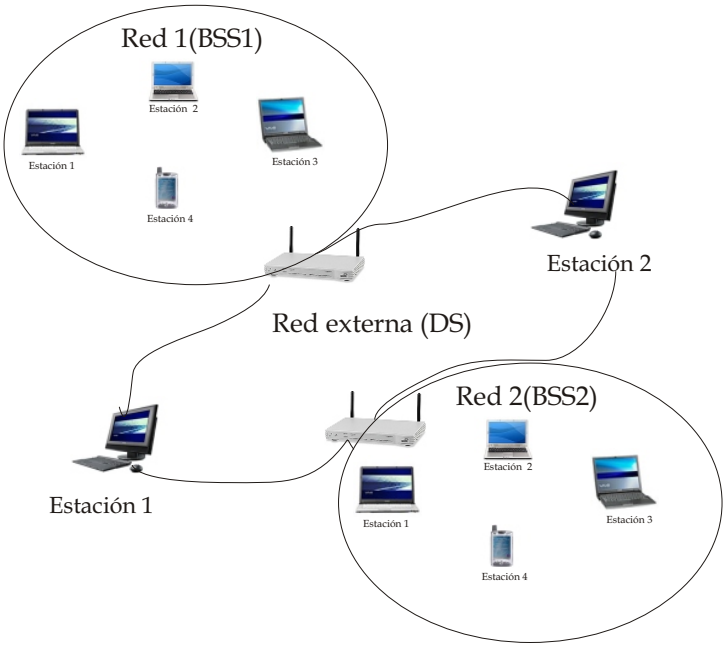
- **802.11g:** Este estándar ofrece una mejora en la velocidad de transmisión en redes inalámbricas. A diferencia del 802.11a, utiliza la misma frecuencia que el 802.11b, lo que permite compatibilidad entre los productos. Sin embargo gracias a una mejora en la forma de modulación, logra una velocidad de hasta 54Mbps, sin embargo para que esa velocidad sea obtenida, es necesario que los productos estén bajo el estándar 802.11g. Debido a su compatibilidad con el 802.11b, se convierte en una opción atractiva para las organizaciones que cuenten con infraestructura 802.11b.

Una WLAN basada en el IEEE 802.11 es un grupo de estaciones (nodo de redes alambradas LANs), ubicadas dentro de un área físicamente delimitada, donde cada estación es capaz de comunicarse mediante radio frecuencia con una estación base. Existen dos tipos de diseño para las WLANs: [9]

Figura 1.4: Red tipo Ad-Hoc

- **Ad-Hoc:** Este tipo de configuración tiene la habilidad de comunicarse con redes externas sin el uso de protocolos adicionales para el ruteo. Normalmente son creadas para permitir la comunicación entre dispositivos de forma directa, por lo que requieren muy poco Hardware y recursos. La figura 1.4 ilustra este tipo de redes.

- **Basadas en Infraestructura:** Están compuestas por uno o más Conjuntos de Servicios Básicos (*BBS* por sus siglas en inglés). Cada estación tiene un enlace a la infraestructura, el Sistema de Distribución (*DS* por sus siglas en inglés), el cual permite el acceso a redes externas. La conexión de la estación al DS se denomina Punto de Acceso (*AP* por sus siglas en inglés), el cual administra los paquetes que se intercambian entre el *BSS* y el *DS* como se muestra en la figura 1.5.



Red basada en Infraestructura

Figura 1.5: Red tipo Infraestructura

Sin duda, la tecnología de redes inalámbricas representa uno de los desarrollos más importantes de la actualidad, sin embargo existen ciertos problemas inherentes que ponen en riesgo la seguridad de la información que viaja a través de ellas.

Para un diseño correcto en una red inalámbrica es necesario considerar diversos aspectos que difieren de una red alámbrica, por ejemplo el medio, pues debido a que el principal medio de difusión es el aire, en una red inalámbrica el envío de información no puede ser restringido a un espacio. Este aspecto es importante ya que cualquier oponente que se encuentre dentro del rango de la red inalámbrica tendrá acceso completo a la información.

Otro de los aspectos que deben tomarse en cuenta en el diseño de los esquemas de seguridad es el ancho de banda, pues el espectro para las comunicaciones es muy limitado. Dichas limitaciones afectan directamente a las implementaciones para la seguridad, pues éstas deben utilizar muy poco intercambio de información, así como ser implementadas en

dispositivos que tienen poco poder de cómputo y limitaciones de memoria, lo que reduce significativamente la complejidad de los algoritmos que son utilizados.

Los ataques a los que las redes inalámbricas son susceptibles se pueden clasificar en dos grupos principales: pasivos y activos.

Los ataques pasivos se producen cuando usuarios no autorizados obtienen acceso al canal de comunicación, logrando así ejecutar un ataque pasivo mediante el robo de información en su totalidad o en partes. Debido a las características del medio de transmisión, los ataques pasivos de esta naturaleza se pueden realizar de forma muy sencilla, pues al menos en teoría cualquiera con un receptor adecuado, dentro del rango de transmisión, puede espiar el tráfico existente en la red inalámbrica.

Los ataques activos por otro lado, se refieren a aquellos en los que el intruso modifica maliciosamente los datos que son transmitidos. Las redes inalámbricas son muy susceptibles a este tipo de ataques, pues la potencia de la señal utilizada para transmitir la información no está regulada por ninguna entidad. Esto hace que sea posible para algún nodo actuar como el punto de acceso y no permitir que nadie en la red se comunique, logrando así un ataque conocido como *"Denegación de servicio"*, que aunque es un ataque simple, no deja de ser poderoso.

Debido a lo anterior, se hace necesario contar con esquemas de seguridad que garanticen la integridad de la información que fluye por la red. En las siguientes secciones abordaremos algunas propuestas que son utilizadas en la actualidad para garantizar la protección del tráfico de la red.

1.1.2. El protocolo WEP

La seguridad en las redes inalámbricas, es un aspecto muy importante, debido a que la naturaleza de estas redes las hace más vulnerables a diversos ataques, por lo que la IEEE diseñó un esquema de seguridad para proteger las redes basadas en el IEEE 802.11b [7].

Cuando la IEEE creó la especificación 802.11, se implementó el WEP (*Wired Equivalent Privacy*) con el intento de proveer los niveles básicos de autenticación y cifrado de datos. Esto era necesario debido a que las redes inalámbricas no cuentan con una protección física como las redes alámbricas. Tanto el 802.11a y el 802.11b utilizan a WEP como esquema de cifrado e integridad [16].

El protocolo WEP, fue concebido como un método de cifrado para las redes inalámbricas, el cual busca proveer seguridad equivalente a un punto de acceso cableado. WEP fue originalmente diseñado para llaves de 40 bits, y después se desarrolló el WEP2 para incrementar la longitud de las llaves para extenderlas a 104 bits [7].

El cifrado se hace con base en paquetes, es decir, cada paquete es en esencia un mensaje

(*plaintext*) a enviar. A continuación se explica la manera en que funciona el WEP.

Denominamos M al mensaje a enviar, y se calcula la suma de verificación (*checksum*) para que la integridad pueda ser verificada posteriormente. El *checksum* se calcula usando un código de redundancia cíclica (CRC por sus siglas en inglés) de 32 bits denominado $CRC32$ (véanse los algoritmos 1 y 2). Esta suma de verificación será llamada CS, por lo que $CS = CRC32(M)$. Este valor es agregado al final del mensaje, lo que forma entonces el texto en claro P. La figura 1.6 ilustra la composición de P. El polinomio generador del $CRC32$ se muestra en la ecuación 1.1.

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^4 + x^0 \quad (1.1)$$

La idea básica de un algoritmo CRC es simplemente tratar el mensaje como un número binario muy grande, para dividirlo entre otro número binario constante (el polinomio generador), y hacer al residuo de esta división la suma de verificación (*checksum*). El receptor puede realizar la misma división y comparar el residuo con la suma de verificación recibida [22].

A continuación brindamos un ejemplo de la forma en que se calcula el CRC de un mensaje [23].

- Para realizar el cálculo del CRC de un mensaje, elegimos primero un divisor, al cual se le denomina polinomio generador, o simplemente "polinomio".
- Se puede elegir un polinomio para calcular el CRC. Sin embargo existen algunos que resultan ser más seguros que otros, por lo que es recomendable elegir un polinomio que haya sido ampliamente probado.
- La longitud del polinomio es un aspecto importante, puesto que afecta todo el cálculo. Típicamente se eligen polinomios de 16 ó 32 bits de longitud con la finalidad de simplificar la implementación en las computadoras modernas. La longitud de un polinomio se determina por la posición real del bit más significativo. Para propósitos de este ejemplo, utilzaremos el polinomio 10011 de longitud 4.
- Una vez que se eligió el polinomio, se realiza el cálculo del CRC, lo que es en realidad una simple división (en aritmética del CRC). El único truco es que se le tienen que agregar tantos ceros como el valor de la longitud del polinomio (en este ejemplo 4). Por lo que el CRC es calculado como:

```
Mensaje original                    : 1101011011
Polinomio                           :      10011
Mensaje después de agregar 4 ceros  : 11010110110000

           1100001010 = Cociente (no importa en realidad)
      ---------------
```

```
10011 | 11010110110000 = Mensaje con ceros (1101011011 + 0000)
=Pol.  10011,,.,,....
        -----,,.,,....
         10011,.,,....
         10011,.,,....
         -----,.,,....
          00001.,,....
          00000.,,....
          -----.,,....
           00010,,....
           00000,,....
           -----,,....
            00101,....
            00000,....
            -----,....
             01011....
             00000....
             -----....
              10110...
              10011...
              -----...
               01010..
               00000..
               -----..
                10100.
                10011.
                -----.
                 01110
                 00000
                 -----
                  1110 = Residuo = CRC
```

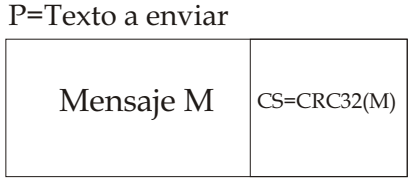


Figura 1.6: Composición del texto en claro

El siguiente paso es cifrar P, lo cual se hace usando el $RC4$, que es un cifrador por flujo de datos. Este cifrador se inicializa con un valor denominado semilla, que genera un flujo de

Algoritmo 1 Algoritmo del CRC32 basado en tablas de consulta

Requiere: $Mensaje, TablaCRC32, MesLong$

1: $RegistroSalida \leftarrow 1.$
2: $Buffer \leftarrow Mensaje$
3: Agregar 4 bytes al $Buffer$
4: **for** $i = 0$ to $MesLong$ **do**
5: $MSB = RegistroSalida[3]$
6: $TableValue = TablaCRC32[MSB]$
7: $RegistroSalida = (RegistroSalida << 1) + Buffer[i]$
8: $ResgistroSalida[3] = TableValue \oplus RegistroSalida[3]$
9: **end for**
10: $RegistroSalida$ almacena el CRC32 de $Mensaje$

Algoritmo 2 Algoritmo para calcular la Tabla de Consulta del CRC

Requiere: $polimonio$ {El polinomio $0x04C11DB7H$ es de los más utilizados}

1: $TablaCRC32 \leftarrow 0$
2: **for** $i = 0$ to 255 **do**
3: $crc = i$
4: **for** $j = 0$ to 7 **do**
5: **if** $(crc\&1) = 1$ **then**
6: $crc = ((crc >> 1) \oplus polinomio)$
7: **else**
8: $crc = crc >> 1$
9: **end if**
10: **end for**$TablaCRC32[i] = crc$ {Asignamos el valor calculado al elemento correspondiente en la tabla}
11: **end for**

llaves, el cual es una cadena de longitud arbitraria de bytes pseudoaleatorios. WEP utiliza un vector de inicialización, IV, para el valor de la semilla. El IV consiste de una cadena de 24 bits que es generada para cada paquete. Algunas de las primeras implementaciones de WEP utilizaban valores secuenciales para IV, mientras que las más recientes utilizan algún tipo de generador aleatorio [7].

Sin importar cómo es escogido el valor de IV, éste es agregado al inicio de la llave del WEP. Los 24 bits de IV son incluidos en el tamaño de la llave del WEP. Es decir, cuando nos referimos a una llave de 64 o 128 bits, en realidad los tamaños reales son de 40 y 104 bits respectivamente con los 24 bits de IV. El IV y la llave del WEP forman la semilla, la cual se denomina S. En la figura 1.7 se muestra cómo se compone la semilla.

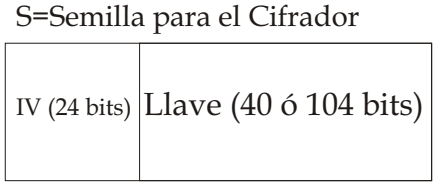


Figura 1.7: Composición de la Semilla S

Entonces la semilla es introducida en el $RC4$, el cual genera el flujo de llaves (KS). Este flujo es entonces "sumado" (mediante la operación XOR) con el texto en claro P, para producir el mensaje cifrado C. Se agrega C al IV, y el resultado es enviado a través de la red al receptor. El algoritmo 3 muestra los pasos a seguir para realizar el cifrado de un mensaje. La figura 1.8 ilustra el proceso de cifrado.

Algoritmo 3 Cifrado con WEP

Requiere: M, K, IV

1: Calcular $CS = CRC32(M)$

2: Agregar CS a M y formar P

3: Formar S con IV y K

4: $KS = RC4(S)$

5: $C = P \oplus KS$

6: Enviar IV y C

Cuando un mensaje cifrado con WEP es recibido, el proceso simplemente se realiza en forma invertida para obtener el mensaje. Esto se hace separando el IV del mensaje y lo concatena a la llave K para producir la semilla S. Si tanto el emisor como el receptor tienen la misma llave, entonces la semilla será igual. Se introduce la semilla S al $RC4$ para producir el mismo flujo de llaves, el cual es "sumado" (XOR) con el resto del mensaje cifrado. Esto producirá el texto claro original, el cual consiste en el mensaje M concatenado con CS. El receptor calcula $CS_r = CRC32(M)$ para comprobar la integridad del mensaje recibido. Esto

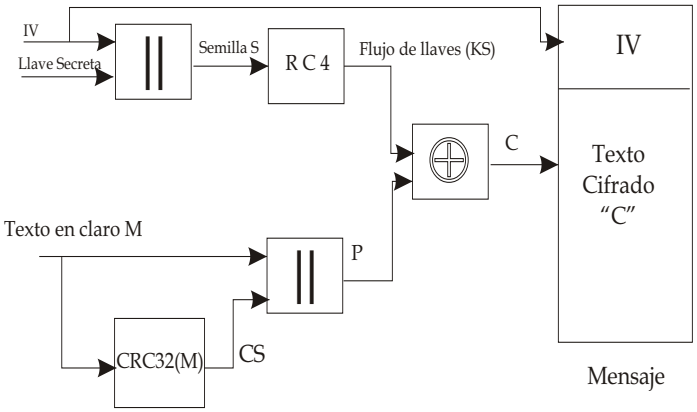


Figura 1.8: El proceso de cifrado con WEP

se hace comparando CS_r con CS. Si son iguales entonces el mensaje es aceptado, en caso contrario se asume que el paquete sufrió modificaciones en la transmisión.

WEP es utilizado para realizar la autenticación en el estándar IEEE 802.11b por medio de retos (basado en una llave secreta compartida), para lo cual es necesario llevar a cabo los siguientes pasos [40]:

1. Una estación cliente envía una trama de solicitud de autenticación al punto de acceso (AP).

2. Cuando el AP recibe una trama de solicitud de autenticación, contesta con una trama de Autenticación que contiene un mensaje aleatorio de reto de 128 bytes, el cual es generado por WEP.

3. El cliente al recibir el reto, lo agrega a una trama de Autenticación cifrando el resultado con la llave secreta compartida y lo envía de regreso al AP.

4. El AP recibe la trama y la descifra con la llave secreta compartida, para comparar el texto original que le envió al cliente, con el recibido. Si son iguales, entonces envía una trama donde indica que la autenticación se realizó de manera satisfactoria. En caso contrario, envía una Autenticación negativa.

El proceso de autenticación se ilustra en la figura 1.9.

Para comprender un poco más el proceso de cifrado de WEP, se analizará al cifrador por flujo de datos $RC4$.

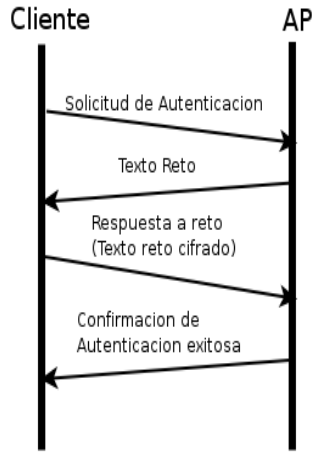


Figura 1.9: Autenticación por retos en el IEEE 802.11

El cifrador *RC4*

El *RC4* es el cifrador más utilizado en aplicaciones de software. Fue diseñado por Ron Rivest en 1987, pero fue mantenido bajo secreto hasta que fue dado a conocer en 1994. El *RC4* tiene un estado interno secreto el cual es la permutación de todas las 2^n palabras posibles de n bits, con dos índices en él. En aplicaciones prácticas $n = 8$, por lo que el *RC4* tiene un estado interno de $\log_2(2^8! \times (2^8)^2) \approx 1700$ bits [30].

El *RC4* es un algoritmo sorprendentemente sencillo [7]. Está constituido de dos algoritmos: El algoritmo de expansión de llave (*Key Scheduling Algorithm (KSA)*) y el algoritmo generador de números pseudoaleatorios (*Pseudo Random Generation Algorithm (PRGA)*). Los dos algoritmos usan una caja de sustitución (*S-BOX*) de 8×8 bytes, la cual es un arreglo de 256 números únicos y en un rango de 0 a 255. Lo anterior quiere decir que la *S-BOX* contiene los números del 0 al 255 pero permutados de diferentes formas. El *KSA* realiza la permutación inicial de la *S-BOX* basado en el valor de la semilla S la cual puede tener una longitud de hasta 256 bits.

Primero el arreglo de *S-BOX* es llenado con los valores 0 a 255 de manera secuencial. Este arreglo lo denominaremos $SBOX$. Un segundo arreglo de 256 bytes es llenado con el valor de la semilla, repitiéndola como sea necesario hasta que el arreglo esté completamente lleno. Este arreglo se denomina K. La $SBOX$ es entonces "revuelta" usando el algoritmo 4.

El *KSA* reacomoda el arreglo de bytes de *S-BOX* conforme a la semilla proporcionada a su entrada.

Para generar el flujo de llaves se utiliza el *PRGA*. Este algoritmo tiene dos contadores, i y j, los cuales son inicializados en 0. El algoritmo 5 muestra el *PRGA*.

El *RC4* es lo suficientemente simple para memorizarlo e implementarlo sin gran esfuerzo, y es bastante seguro si es utilizado de manera adecuada. Sin embargo, existen algunos problemas con la manera en que *RC4* es utilizado en el WEP [7].

Algoritmo 4 Algoritmo del KSA

1: $j = 0$
2: **for** $i = 0$ to 255 **do**
3: $j = (j + SBOX[i] + K[i])$ mod 256;
4: swap $SBOX[i]$ y $SBOX[j]$
5: **end for**

Algoritmo 5 Algoritmo del PRGA

1: $i = 0$, $j = 0$
2: **for all** Elementos en flujo de llaves **do**
3: $i = (i + 1)$ mod 256;
4: $j = (j + SBOX[i])$ mod 256;
5: swap $SBOX[i]$ y $SBOX[j]$
6: $t = (SBOX[i] + SBOX[j])$ mod 256;
7: Sacar el valor de $SBOX[t]$;
8: **end for**

Problemas de seguridad con el WEP

Existen diversos problemas con la seguridad proporcionada por el WEP. Sin embargo es necesario puntualizar que nunca fue diseñado para ser un protocolo criptográfico fuerte, sino que fue pensado únicamente como una forma para proporcionar seguridad equivalente al alámbrico, como su mismo nombre lo indica. Existen algunos problemas con el protocolo, algunos de los cuales son ocasionados por el uso de la función $CRC32$ como método para verificar la integridad del mensaje, y algunos otros por la manera en que los IV son utilizados [7].

En [4], se demuestra que el $CRC32$ no detecta ataques intencionados, lo que lo hace vulnerable a modificaciones en el mensaje cifrado, que el receptor no podrá detectar y provocará que el mensaje se tome como válido. El ataque se basa en el hecho de que la autenticación en WEP, usando el $CRC32$, es una función lineal, por lo que la suma de verificación se distribuye sobre la operación XOR, es decir, $c(x \oplus y) = c(x) \oplus c(y)$ para todas las elecciones de x y y. Esta es una propiedad general de los algoritmos CRC. Lo anterior significa que es posible realizar modificaciones controladas al texto cifrado, sin que la suma de verificación se altere, y por lo tanto, no detecte las alteraciones mal intencionadas.

Desafortunadamente, tanto la versión de 40 y 104 bits, tienen defectos en cuanto a seguridad se refiere. Para la autenticación el WEP no soporta más de 4 llaves (almacenadas en el Punto de Acceso para ser utilizadas por las aplicaciones) y no cuenta con un mecanismo para cambiar esas llaves de forma dinámica. El resultado es que las mismas llaves son utilizadas

por múltiples clientes y puntos de acceso y no son cambiadas nunca. Esto significa que los usuarios maliciosos pueden "escuchar" la comunicación y al usar algunos de los paquetes de software disponibles gratuitamente, pueden acceder rápidamente al punto de acceso [16].

Debido a que la seguridad del RC4 se basa en la longitud del vector de inicialización IV, que en el caso del WEP es de 24 bits, se repetirá cada 2^{24} paquetes. Para los puntos de acceso con un tráfico moderado, esto ocurre en cuestión de horas, por lo que, los atacantes monitoreando el tráfico de la red pueden detectar dos mensajes cifrados con el mismo vector de inicialización y ser capaces de determinar las llaves y obtener la información en claro utilizando alguno de los ataques al WEP como el encontrado en [30].

Existen diversos ataques al protocolo WEP, a continuación veremos algunos de los más representativos.

- **Fuerza Bruta:** La fuerza bruta siempre es un ataque posible para cualquier criptosistema. La única pregunta que es necesario responder es la factibilidad. Con el WEP, la fuerza bruta se aplica mediante la captura de paquetes y se prueba el descifrado con cada una de las posibles llaves comprobando si se trata del mensaje original. Aunque es un ataque costoso, computacionalmente hablando, en términos generales puede durar desde meses hasta años, por lo que depende del atacante y sus recursos, así como del contexto, para que sea o no factible.

- **Modificación del Mensaje:** En [4] se da la demostración de que el WEP es inseguro contra la modificación del mensaje mal intencionado, sin que la modificación sea detectada. Esto debido al uso del $CRC32$ en el algoritmo de autenticación, con lo cual gracias a que el $CRC32$ es lineal, permite realizar modificaciones controladas al texto cifrado, sin que sean detectadas adecuadamente.

- **Inserción de Mensajes:** Debido a que el $CRC32$ es un método que no usa llave, un oponente puede introducir mensajes maliciosos con la finalidad que sean descifrados por el punto de acceso y así tener información sobre la llave.

- **El ataque FMS:** Es probablemente el ataque más famoso y más efectivo contra el WEP. Fue desarrollado por Fluhrer et al. [30], y ha sido implementado de manera eficiente en [36, 37]. Actualmente existen herramientas como AirSnort que implementan este ataque de manera satisfactoria. Está basado en la debilidad del algoritmo KSA (véase el algoritmo 4) del $RC4$ y el uso de los IVs. Existen IV denominados débiles, los cuales brindan información de la llave secreta. Dado que la misma llave es utilizada con diferentes IVs, si se capturan suficientes paquetes con IVs débiles, y se conoce el primer byte del flujo de llaves, la llave puede ser calculada. Por suerte para el atacante, el primer byte de una trama de 802.11b es casi siempre el byte 0xAA. Esto significa que el primer byte del flujo de llaves puede ser fácilmente obtenido mediante la operación $\oplus$ entre el primer byte del texto cifrado con 0xAA.

Como se ha visto, existen muchos problemas de seguridad inherentes al WEP. Por ejemplo, no proporciona las metas fundamentales de protección equivalente a las redes alámbricas

e incluso no proporciona autenticación e integridad de la información, por lo que se ha vuelto necesario que nuevos esquemas de seguridad sean propuestos [19].

1.1.3. Nuevas propuestas para la seguridad

Debido a que el WEP no es considerado un protocolo seguro, y no puede cubrir los requisitos de seguridad que las WLANs demandan, ha sido necesario que se desarrollen nuevos métodos de seguridad. Nuevos esquemas han sido propuestos y están agrupados dentro del estándar IEEE 802.11i.

Acceso protegido por el Wi-Fi (WPA)

El *Wi-Fi Protected Access* (WPA) ha sido sugerido por el grupo de trabajo del 802.11 con la finalidad de corregir las fallas en la seguridad encontradas en el WEP. Ha sido pensado como un "parche" en software sobre el hardware existente [18].

Para eliminar las debilidades del WEP, dentro del WPA se ha definido el Protocolo de Integridad de Llave Temporal (*TKIP* por sus siglas en inglés). La instalación de este protocolo puede llegar a incluir una actualización del firmware y del driver de la tarjeta. Los requerimientos para que pueda ser ejecutado en el hardware existente imponen grandes restricciones [19]:

- Todos los sistemas desarrollados deben ser actualizables en software o en firmware.
- Deben permitir que la implementación actual del WEP no sea modificada.
- El rendimiento no debe ser disminuido con las modificaciones.

WPA incluye una función hash para defenderse del ataque FMS [30], un código de integridad de mensaje (*MIC* por sus siglas en inglés) y un manejador de llave basado en el 802.11X para evitar la reutilización de llaves y facilitar la distribución de llaves [18]. La figura 1.10 ilustra el proceso de encapsulamiento.

La llave temporal (TK) de 16 bytes es obtenida del esquema de manejo de llave durante la autenticación, y es introducida a la función hash junto con la dirección (6 bytes) del emisor (TA) y los 48 bits del IV, frecuentemente llamado contador de secuencia del TKIP. La función hash proporciona una llave para el $RC4$ de 16 bytes donde los tres primeros bytes son derivados del IV. La llave es utilizada sólo para una trama de WEP, debido a que el IV es implementado como un contador que se incrementa con cada paquete, por lo que la llave es también utilizada como una llave por paquete. El IV también es utilizado como una defensa contra los ataques de reenvío, por lo que el receptor no aceptará paquetes con un valor de IV menor o igual a los recibidos con anterioridad [18].

Para una mayor comprensión de WPA véase [18, 19]. En [14] se presenta un nuevo esquema de control de acceso que utiliza algunas de las propiedades descritas anteriormente.

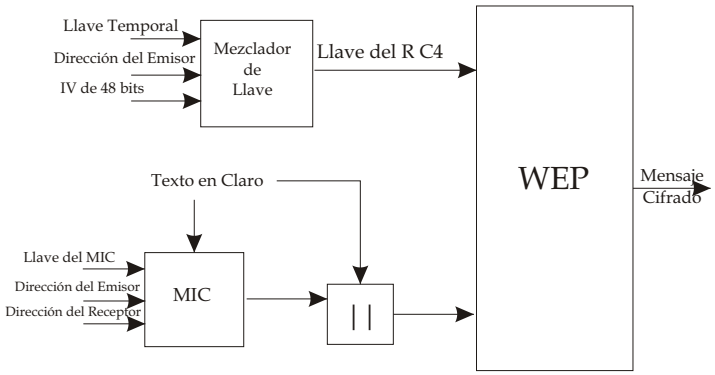


Figura 1.10: Proceso de Encapsulamiento del WPA

Algunos de los problemas que presenta el WPA son analizados en [18], entre los cuales se encuentra el hecho de requerir una actualización en el firmware de la tarjeta de red. Lo anterior debido a que está definido como una expansión al WEP.

Otro de los detalles que hacen deficiente al WPA es el hecho de requerir el uso de dos llaves, una de 128 bits utilizada por el mezclador de llave para generar una llave de cifrado, y otra de 64 bits utilizada por el *MIC*. Además que los procesos para realizar el cifrado y la autenticación de la información requieren de múltiples ciclos de reloj [19].

En [18] se da la descripción de un posible ataque al *WPA*. Los autores remarcan que su artículo no significa que el WPA haya sido roto, sino que destacan la importancia de una buena implementación y el conservar en completo secreto la llave de cada paquete.

El CCMP

CCMP son las siglas de *Counter-Mode-CBC-MAC Protocol* (véase la sección A.2 para una descripción de los algoritmos MAC). Al igual que el WPA, está pensado para sustituir al WEP, pero tiene la ventaja de ser implementado sin considerar el hardware existente. El AES (*Advanced Encryption Standard* véase el capítulo 2) ha sido escogido como el algoritmo para el cifrado [19].

Ninguno de los modos de operación desarrollados para el AES ofrecen un balance adecuado para las aplicaciones móviles. Algunas de las características deseables son [19]:

- Utilizar una llave única para proveer confidencialidad e integridad, y así evitar el manejo

de intercambio de llaves, y al mismo tiempo minimizar el tiempo del cálculo de la expansión de llaves para el AES.

- Proveer protección de integridad al encabezado del paquete, así como al resto de la información.
- Permitir pre-cómputo para reducir los retrasos.
- Soportar paralelización para incrementar el rendimiento.
- Implementaciones económicas en tamaño.
- Evitar modos de operación que estén bajo derechos de autor.

El nuevo modo denominado CCM, fue diseñado por Whiting et al. [41], y está concebido para cumplir con los puntos listados anteriormente. El CCM utiliza el Modo de Conteo (CTR) para el cifrado y el CBC-MAC para la autenticación. Ambos utilizan únicamente la primitiva de cifrado del AES tanto para el emisor como para el receptor.

El CCM utiliza la misma llave tanto para la confidencialidad como para la integridad, lo que es tradicionalmente inseguro, pero el CCM lo evita al garantizar que el espacio del modo del conteo jamás se mezcle con el vector de inicialización del CBC-MAC. La intuición detrás del CCM es que si el AES se comporta como una permutación pseudoaleatoria, entonces la salida del cifrador en cada uno de los modos deberá ser independiente [19]. La figura 1.11 ilustra el proceso del CCMP.

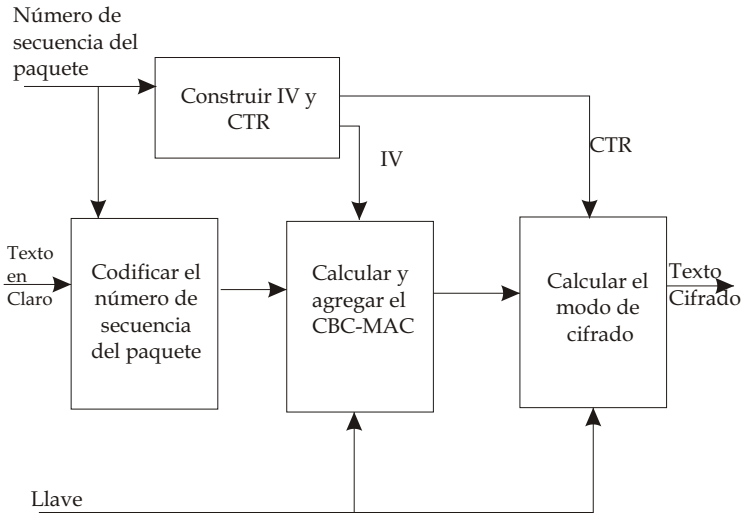


Figura 1.11: El Proceso del CCMP

La tabla 1.2 presenta una comparación entre el WEP, WPA y el CCMP.

	WEP	WPA	CCMP
Cifrador y tamaño de llave	RC4 40 ó 104 bits para el cifrado	RC4 128 bits para cifrado y 64 bits para autenticación	AES 128 bits
Vida útil de la llave	24 bits de protección en el IV	48 bits de IV	48 bits de IV
Llave por paquete	Concatenación de IV con la llave base	Función de mezclado del TKIP	No se necesita
Integridad del encabezado del paquete	Ninguna	Dirección Origen y Destino protegidas por la función hash	CCM
Función de integridad	CRC32	Función hash	CCM
Detección de ataques de reenvío	Ninguna	Secuencia del IV	Secuencia del IV
Manejo de llave	Ninguno	IEEE 802.1X	IEEE 802.1X

Cuadro 1.2: Diferencias entre los esquemas de seguridad para el IEEE802.11

En esta tesis, abordaremos el modo de operación CCM, y se darán los lineamientos de su implementación en una arquitectura de Hardware Reconfigurable, con la finalidad, de proporcionar una alternativa de implementación a bajo costo.

1.2. Seguridad en hardware reconfigurable

La elección de una plataforma de implementación de un sistema digital depende de muchos criterios, y también en gran parte del área de aplicación. Además de los aspectos referentes al algoritmo, la velocidad del sistema y los costos, existen factores específicos al cifrado: seguridad física (por ejemplo contra la recuperación de la llave y la manipulación del algoritmo), flexibilidad (sin importar los parámetros del algoritmo, la llave o incluso el propio algoritmo), el consumo de energía, y otros factores secundarios [42].

Los dispositivos de hardware reconfigurable, tales como los Arreglos de Compuertas de área Reprogramable (*FPGAs* por sus siglas en inglés), intentan combinar las ventajas de las implementaciones en Software y Hardware. Al mismo tiempo, existen todavía muchas preguntas sobre la utilidad de los FPGAs como módulos para funciones de seguridad [42].

La propiedad de reconfiguración de los FPGAs ofrece grandes ventajas cuando son usados en aplicaciones criptográficas. A pesar de que existe una gran cantidad de artículos sobre implementaciones criptográficas en FPGAs, casi no existen trabajos que analicen la conveniencia de los FPGAs en aplicaciones de seguridad desde un punto de vista del sistema. En particular muy poco trabajo se ha hecho sobre la resistencia de los FPGAs a ataques físicos o del sistema, que en general son más peligrosos que los ataques a los algoritmos [43].

En [42, 43] se da un panorama de los aspectos en cuanto a la seguridad en los FPGAs, y se brindan conclusiones sobre los posibles ataques que puede sufrir un Sistema basado en FPGAs. La conclusión a la que se llega es que si bien es cierto que las aplicaciones bajo FPGAs no son cien por ciento seguras, este tipo de tecnología brinda niveles de seguridad razonables cuando es utilizada adecuadamente y en las condiciones apropiadas.

1.2.1. Tecnología de hardware reconfigurable

La microelectrónica ha permitido el desarrollo de sistemas basados tanto en Software como en Hardware en los años recientes. El continuo crecimiento del nivel de integración de los dispositivos electrónicos en un bloque de silicio ha derivado en la producción de sistemas cada vez más complejos. Estos circuitos son comúnmente llamados circuitos Integrados de Gran Escala (*VLSI* por sus siglas en inglés). El diseño de circuitos integrados requieren de una considerable inversión de capital, debido al costo de refinar la precisión del proceso de manufactura [32].

Tradicionalmente, en el diseño de los sistemas empotrados los ASICs (*Application-Specific Integrated Circuit*, Circuito Integrado Específico para Aplicaciones) han sido los componentes comunes al proveer el alto rendimiento y/o el bajo costo que muchos sistemas requieren durante el largo y difícil ciclo de diseño. En 1980 se introdujo el uso de componentes reprogramables, en particular se introdujeron los FPGAs. Los FPGAs brindan un ciclo de diseño más corto porque permiten pruebas de funcionalidad de manera más rápida. Sin embargo, el rendimiento y tamaño de los FPGA ha impedido que éstos sustituyan a los ASICs en la mayoría de las aplicaciones, por lo que fueron utilizados principalmente para chips de prototipos lo suficientemente pequeños para un FGPA. En los años recientes, los fabricantes de FPGA han logrado reducir la brecha que existe entre los FPGAs y los ASICs, permitiéndoles, no únicamente ser utilizados como herramientas de prototipos, sino también participar de manera activa en los sistemas empotrados [43].

Por hardware reconfigurable nos referimos al tipo de circuitos integrados mayoritariamente conocidos como *FPGAs*, aunque existen también otros dispositivos como los "Dispositivos de Lógica Programable (*PLDs* por sus siglas en inglés) que se aproximan a la misma definición pero no a las mismas capacidades. Este tipo de dispositivos son también reconfigurados por el diseñador. Cada nueva configuración es realizada nuevamente en fracciones de segundo y con esto se logra que el FPGA sea capaz de llevar a cabo una función totalmente nueva [21].

Los FPGAs son considerados como dispositivos reconfigurables debido a que pueden ser configurados en tiempo de ejecución. En otras palabras, el diseño del circuito puede cambiar de acuerdo a las necesidades del sistema en el que esté siendo ejecutado.

Un FPGA es un circuito integrado que pertenece a la clase de dispositivos programables, el cual consiste en miles de bloques básicos, denominados Bloques Lógicos de Configuración (*CLBs* por sus siglas en inglés), los cuales están conectados a través de interconexiones pro-

gramables [28].

El funcionamiento y la cantidad de CLBs están definidos por las especificaciones de los fabricantes. Existen diversas familias y fabricantes de FPGAs, los más famosos son: Virtex, Spartan (Xilinx, esta tesis fue desarrollada en un FPGA de la familia Spartan), FLEX, APEX (Altera), ACT (Actel), pASIC (QuickLogic), LCA (Logic Cell Array) y ORCA (Lucent). En años recientes, se han obtenido grandes avances en la tecnología de FPGAs, logrando que los dispositivos más modernos operen con un reloj interno en una frecuencia superior a los 400 MHz con una densidad de más de 8 millones de compuertas en un solo encapsulado (FPGA Virtex-II). Las mejoras en la tecnología no se limitan a brindar más compuertas lógicas sino que han permitido la disponibilidad de muchos bloques funcionales como memorias de acceso más rápido y multiplicadores. Incluso, algunos FPGAs cuentan con procesadores PowerPC integrados.

Para una mejor comprensión de los FPGA es necesario estudiar su composición interna, para entender las propiedades que los hacen programables. Debido a que existe una amplia variedad de tecnologías para implementar dispositivos programables a continuación discutiremos algunas de las tecnologías disponibles en los FPGAs. Las tecnologías disponibles para las memorias son: Memoria de Acceso Aleatorio Estática (*SRAM* por sus siglas en inglés), Memoria de Solo Lectura Programable y Borrable (*EPROM*), Memoria de Solo Lectura Programable y Eléctricamente Borrable (*EEPROM*).

La tecnología específica define si un dispositivo es reprogramable o programable únicamente una vez, volátil o no. La mayoría de los FPGAs actuales son programables a través de una SRAM, lo que significa que los bits de la SRAM son conectados a los puntos de configuración en el FPGA, por lo que al programar las celdas de la SRAM configura al dispositivo. Algunas de las características de las memorias son listadas en la tabla 1.3 [32].

Tecnología	SRAM	EPROM	EEPROM
Reprogramable	√	√	√
Programable en el Sistema	√	-	√
Volátil	√	-	-

Cuadro 1.3: Propiedades de los diferentes tipos de Memorias

El diseño y la implementación de la arquitectura de un FPGA se encuentra escasamente descrita en la literatura abierta, debido a que mucha de la información es propietaria de los diferentes fabricantes. En general los FPGAs pueden ser vistos como un gran conjunto de compuertas lógicas programables, donde no únicamente la lógica sino también las conexiones son programables por el usuario. En FPGAs basados en RAM, las celdas de las SRAMs son usadas para almacenar los bits de configuración, los cuales programan los diferentes compo-

nentes del dispositivo (por ejemplo, bloques lógicos, bloques de conexión e intercambio). Lo anterior es ilustrado en la figura 1.12.

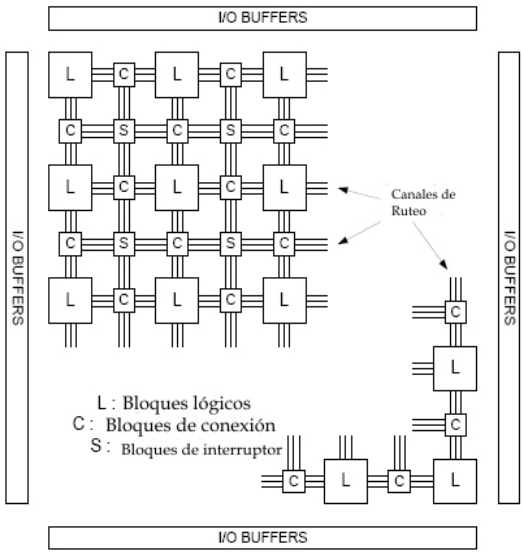


Figura 1.12: Los FPGAs vistos desde un alto nivel

El diseño de los bloques de cálculo en el hardware reconfigurable varía de sistema a sistema. Cada unidad de cálculo o bloque lógico, puede ser tan simple como una Tabla de Consulta (LUT por sus siglas en inglés) de 3 entradas, o tan compleja como una Unidad Aritmética y Lógica (ALU por sus siglas en inglés) de 4 bits. Esta diferencia en el tamaño del bloque es comúnmente denominada granularidad del bloque lógico. Los bloques con una granularidad más fina son útiles para las manipulaciones a nivel de bits, mientras que los que tienen una granularidad más gruesa están más optimizados para niveles más altos de manipulación. El nivel de granularidad en un FPGA tiene un gran impacto en el tiempo de configuración del dispositivo. Por ejemplo, un dispositivo con una granularidad fina, tiene muchos puntos de configuración para realizar cálculos muy pequeños, por lo que requiere de más bits de datos durante la reconfiguración [32].

La interconexión entre los bloques lógicos con el hardware reconfigurable es de gran importancia. Dicha conexión contribuye significativamente a la importancia del hardware reconfigurable. Cuando el porcentaje de bloques lógicos en un FPGA se incrementa significativamente, las herramientas de ruteo automáticas presentan dificultades para obtener los enlaces necesarios entre los bloques. Por lo anterior, con la finalidad de garantizar que un diseño puede ser implementado satisfactoriamente en el hardware reconfigurable es necesario tener buenas estructuras de interconexión.

A continuación describiremos los componentes internos de un FPGA con la finalidad de brindar un mejor panorama sobre la funcionalidad de estos dispositivos.

Bloques Lógicos Configurables (CLBs)

Los Bloques Lógicos Configurables (CLBs) constituyen el principal recurso de circuitos lógicos para la implementación de circuitos síncronos así como combinatorios. Cada CLB se compone de cuatro "*Slices*" (término propio de Xilinx para referirse a las unidades básicas de un FPGA) interconectados como se muestra en la figura 1.13. Los *slices* están agrupados en parejas y cada pareja está organizada como una columna con una cadena de acarreo independiente [44].

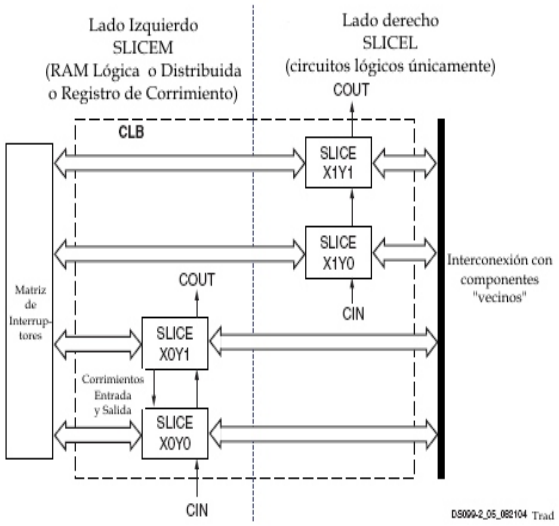


Figura 1.13: Arreglo de los Slices en un CLB

Los cuatro *slices* tienen los siguientes elementos en común: dos generadores de funciones lógicas, dos elementos de almacenamiento, multiplexores, circuitos lógicos para el manejo de acarreo, y compuertas lógicas aritméticas. Tanto la pareja de *slices* de la izquierda como la de la derecha utilizan estos elementos para proveer las funciones lógicas, aritméticas y de ROM. Además de esto, la pareja izquierda soporta dos funciones adicionales: almacenamiento de datos usando RAM Distribuida y el corrimiento de datos con registros de 16 bits. El generador de funciones basado en RAM también conocido como Tablas de consulta (LUTs) es el recurso principal para la implementación de funciones lógicas. Las LUTs en cada *slice* de la izquierda pueden ser configurada como una RAM distribuida o como un registro de corrimiento de 16 bits.

Desde la introducción de los FPGAs a mediados de 1980, se han realizado múltiples investigaciones con la finalidad de encontrar los componentes de cálculo adecuados para ser incluidos en el arreglo de compuertas lógicas. En la actualidad los componentes que se han colocado como los más útiles son las tablas de consulta (LUTs). Las LUTs de N entradas son básicamente una memoria que, cuando es programada apropiadamente, puede realizar cualquier función de hasta N bits de entrada. Un FPGA típico tiene CLBs con una o más LUTs

de 4 entradas, opcionalmente puede contar con Flip-Flops (registros de memoria) tipo "D" y con circuitos que implementen el cálculo de acarreos de forma rápida (ver figura 1.14). Las LUTs permiten que cualquier función booleana que no exceda el número de entradas pueda ser implementada, por lo que provee circuitos lógicos genéricos. Los Flip-Flops pueden ser usados para realizar un *pipeline* (técnica para paralelizar un proceso), registros o cualquier otra función en la que se requiera sincronización. Los circuitos para el cálculo de acarreos de forma eficiente son recursos especiales que se encuentran en la celda con la finalidad de acelerar los cálculos que requieran del acarreo. Debido a que las opciones de interconexión son muy pocas, y debido al poco retraso en los cálculos, el uso de estos recursos mejoran significativamente la velocidad en la propagación del acarreo. Frecuentemente en los Bloques Lógicos de los dispositivos más recientes, se incluyen componentes lógicos adicionales (como por ejemplo compuertas XOR, multiplexores), los cuales permiten que sean implementadas de manera eficiente una gran variedad de funciones.

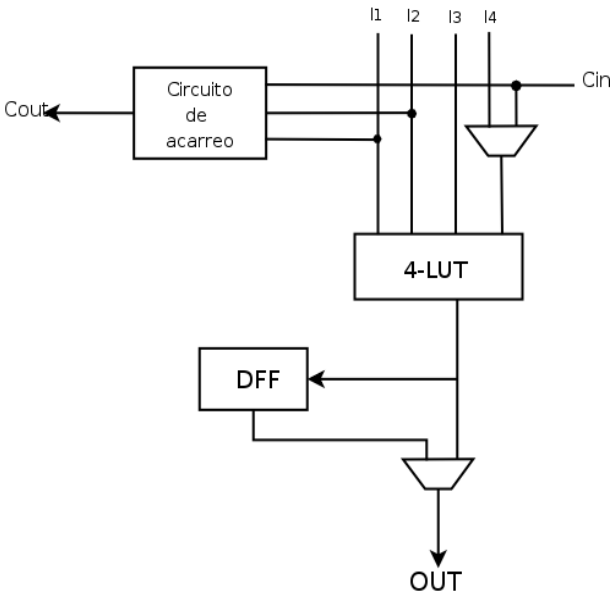


Figura 1.14: Un bloque lógico Básico

Un CLB puede ser utilizado para implementar cualquiera de las siguientes funciones [21]:

- Cualquier función de hasta 4 variables, además de una función secundaria de hasta 4 variables no relacionadas, además una tercera función de hasta 3 variables independientes.
- Cualquier función de hasta 5 variables.
- Cualquier función de 4 variables junto con algunas funciones de 6 variables.
- Algunas funciones de hasta 9 variables.

La implementación de una gran variedad de funciones en un solo bloque reduce tanto el número de bloques requeridos como el retraso en la propagación de la señal, logrando el incremento en la capacidad y en la velocidad. La versatilidad de los generadores de funciones de los CLB mejora de forma significativa la velocidad del sistema. Además de lo anterior, las herramientas de diseño disponibles pueden configurar cada generador de funciones de manera independiente lo que permite un mejor aprovechamiento de la celda.

Estructuras de interconexión

Así como los Bloques Lógicos de Configuración son importantes en los diseños en FPGAs, la interconexión tiene un papel muy importante debido a la necesidad de una comunicación rápida y eficiente a través de las columnas y los renglones de los bloques lógicos. Existen diferentes tipos de longitudes cuando se interconectan recursos. Algunas son interconexiones locales, las cuales son hechas en componentes adyacentes (por ejemplo circuitos de acarreo), lo que permite una comunicación más rápida. Las interconexiones de longitud media pueden ser hechas entre muchos bloques lógicos, lo que que provoca una mayor distancia de interconexión. Finalmente, las de tipo largo son las que se realizan a través de todo el *chip* y que normalmente son utilizadas para las señales globales.

Normalmente se consideran dos soluciones para la implementación de las conexiones. La primera solución utiliza interruptores (es decir transistores de paso controlados por una celda de RAM). Otra posibilidad es el uso de multiplexores. La principal ventaja de los multiplexores es que un solo bit de control puede manejar muchos dispositivos, por lo que el uso de la RAM es más eficiente. La principal desventaja es la ruta crítica de la señal a través de los multiplexores.

Los Bloques de Entrada/Salida

Los Bloques de Entrada/Salida (*IOB* por sus siglas en inglés), proveen una interfaz bidireccional programable entre una conexión de entrada/salida y la lógica interna de un FPGA. Existen tres tipos de señales de ruteo para un IOB: la señal de salida, la señal de entrada y la señal del tercer estado (alta impedancia). Cada una de estas señales tienen su propio par de elementos de almacenamiento que pueden comportarse tanto como registros o como "*latches*". Estas tres señales se comportan como sigue [44]:

- *La señal de Entrada* introduce información a los componentes internos del FPGA. Dicha información es alimentada a las líneas de entrada del FPGA para que sea procesada de acuerdo a las funciones programadas.
- *La señal de Salida* es la encargada de proporcionar la salida a los datos procesados en los componentes internos del FPGA para que sea proporcionada a la interfaz externa del FPGA.
- *La señal del Tercer Estado* determina cuando una conexión de salida está en estado de alta impedancia, con la finalidad de no causar interferencia con otras conexiones en la interfaz externa con el FPGA.

Bloques de RAM (BRAM)

Es una característica que no todos los modelos de FGPA tienen disponible. Sin embargo, debido a que en esta tesis se utiliza el dispositivo Spartan 3 de Xilinx, el cual sí tiene BRAMs, daremos una breve explicación de este componente.

Los BRAMs están organizados como bloques de 18 Kbit los cuales son configurables y síncronos. Los BRAMs almacenan grandes cantidades de datos de manera más eficiente que la RAM Distribuida, la cual es más recomendable para pequeñas cantidades de información.

La relación entre el ancho y la profundidad de cada bloque de RAM es configurable. Incluso múltiples bloques pueden ser conectados en cascada para crear memorias más anchas y/o más profundas. Una elección entre las primitivas de configuración determina si un bloque de RAM funciona como una memoria de puerto doble o simple. Un bloque llamado RAM16_S[w_A]_S[w_B] indica que se genere una memoria de doble puerto donde los enteros w_A y w_B especifican el total del tamaño de la memoria en los puertos w_A y w_B respectivamente. Un nombre de la forma RAM16_S[w] identifica una memoria de puerto simple, donde el entero w especifica el tamaño total del puerto único.

1.2.2. Propuestas de seguridad en hardware reconfigurable

Actualmente con el rápido crecimiento de la tecnología de Hardware Reconfigurable, es posible el desarrollo de nuevas propuestas para brindar un nivel de seguridad confiable en la implementación de soluciones reales, por lo que se hace necesario contar con buenos diseños que permitan obtener un alto grado de eficiencia en el sistema.

Diversos trabajos han destacado en el área, por lo que en esta sección evaluaremos algunas de las propuestas que se encuentran disponibles en la literatura abierta.

Saqib [28] brinda nuevas técnicas para la implementación eficiente de algoritmos criptográficos en FPGAs. En este trabajo, da los lineamientos para llevar a cabo un buen diseño de los cifradores por bloque más conocidos en la actualidad: DES y AES. Dicho diseño es conseguido de manera eficiente lo que permite que puedan ser utilizados para aplicaciones que requieran la protección de los datos a través de cifradores por bloques. En este trabajo Saqib brinda nuevas ideas de optimización en los algoritmos de los cifradores por bloque y brinda ideas para su implementación tanto en un modo secuencial como en modo paralelo.

Este antecedente es importante para nuestro trabajo de tesis debido a que tomamos algunas ideas de la implementación del AES de Saqib para poder desarrollar nuestra propia implementación.

Un trabajo similar al de Saqib lo reporta Standaert en [32], en el cual presenta en particular una muy buena implementación del cifrador por bloque AES, dicho cifrador está tomando una importancia relevante en la seguridad informática. Standaert se enfoca únicamente a la

implementación de algoritmos de Criptografía simétrica, en concreto en el DES y el AES.

Wollinger et al. [42] brindan el estado del arte de algunas implementaciones de algoritmos criptográficos tanto de criptografía simétrica como asimétrica. En este trabajo brindan el panorama de algunas de las implementaciones reportadas hasta esa fecha. Comparan las distintas implementaciones encontradas en la literatura y brindan ciertas observaciones a dichas implementaciones.

Como se muestra en [42, 43] los FPGAs son dispositivos que brindan un nivel de seguridad razonable cuando son utilizados apropiadamente, por lo que el desarrollo de implementaciones basadas en esta tecnología se hace más factible para los productos comerciales de hoy en día.

En este trabajo de tesis se brindan las bases para el diseño y la implementación de manera eficiente del nuevo método de protección para las redes inalámbricas, el estándar IEEE 802.11i, el cual implementa el denominado CCMP (vea sección 1.1.3).

Capítulo 2

El Cifrador por bloques AES

En este capítulo se dará la descripción del cifrador por bloques denominado AES (*Advanced Encryption Standard*). Iniciaremos con una breve historia de su desarrollo, para posteriormente enfocar nuestro estudio a sus principales características. Con el objetivo de obtener una implementación eficiente incluimos también algunas ideas de optimización encontradas en la literatura para el algoritmo del AES.

Como se mencionó en el capítulo anterior, el modo de operación CCM puede ser implementado con cualquier cifrador por bloques, tal como el AES, es por eso que en este capítulo nos enfocamos a su descripción, con la finalidad de brindar los aspectos teóricos que serán utilizados en la implementación del mismo. Debido a que el modo CCM, utiliza únicamente la primitiva de cifrado del AES, en este capítulo solamente se estudia la teoría referente a dicha primitiva. Para una descripción detallada del AES véase [6].

2.1. Breve historia del AES

En Enero de 1997, el Instituto Nacional de Estándares y Tecnología de los Estados Unidos (NIST), anunció una iniciativa para desarrollar un nuevo estándar de cifrado: El Estándar de Cifrado Avanzado (*AES* por sus siglas en inglés). Este estándar de cifrado se convertiría a su vez en un Estándar Federal de Procesamiento de Información (*FIPS* por sus siglas en inglés), y estaba pensado para reemplazar al viejo DES (*Data Encryption Standard*) y su variante el 3-DES.

A diferencia de otros estándares desarrollados anteriormente para el NIST, se anunció que el proceso de selección para el AES sería abierto a toda la comunidad criptográfica. Cualquiera podría enviar un cifrador candidato. El NIST recibió diferentes algoritmos de diversos grupos de trabajo alrededor del mundo. Esto permitió que el proceso de selección contara con las contribuciones de toda la comunidad criptográfica a nivel mundial.

2.1.1. El inicio del proceso del AES

En septiembre de 1997, los requerimientos finales para las nominaciones de candidatos para el AES fueron publicados. Los requerimientos funcionales mínimos solicitados para los cifradores simétricos por bloques, incluían la capacidad de ser utilizados con longitudes de bloque de 128 bits y longitudes de llave de 128, 192 y 256 bits.

El NIST anunció que su búsqueda se concentraba en encontrar un cifrador por bloques *tan seguro como el 3-DES, pero mucho más eficiente*. Otro de los requerimientos importantes fue que los diseñadores aceptaran hacer su algoritmo libre de derechos de autor, si era elegido como el nuevo AES. Para poder ser considerado como un candidato oficial para el AES, los diseñadores debían proveer [6]:

1. La especificación completa del cifrador por bloques en forma de algoritmo.
2. Una implementación de referencia en ANSI C, así como las implementaciones matemáticamente optimizadas tanto en ANSI C como en JAVA.
3. Implementaciones de pruebas diversas incluidas las denominadas "Monte Carlo", así como las respuestas esperadas para estas implementaciones para una correcta implementación del cifrador por bloques.
4. Análisis sobre la eficiencia computacional estimada tanto para hardware como para software, la fortaleza esperada contra ataques criptoanalíticos, y las ventajas y limitaciones del cifrador en diversas aplicaciones.
5. Un análisis de la fortaleza del cifrador contra los ataques criptoanalíticos más conocidos.

El esfuerzo por producir un paquete de propuesta "completo y adecuado", se convirtió en el primer filtro para muchas de las postulaciones candidatas, aún cuando uno de los equipos criptográficos anunció que su organización proporcionaría las implementaciones en Java de los algoritmos propuestos, así como la implementación para las pruebas estadísticas respectivas, incluidas las de Monte Carlo.

2.1.2. Criterios de evaluación

El criterio de evaluación para la primera ronda fue dividido principalmente en tres categorías: seguridad, costo y características del algoritmo y de la implementación. El NIST invitó a toda la comunidad criptográfica a montar ataques y tratar de criptoanalizar los diferentes candidatos, así como evaluar los costos de implementación. Los resultados serían enviados al NIST para seleccionar a cinco finalistas y posteriormente obtener al ganador.

El ganador fue anunciado en Octubre del 2000, siendo seleccionado como el AES el algoritmo "Rijndael", (el nombre es una combinación de los nombres de sus dos autores, Vincent "Rijmen" y Joan "Daemen"). Desde el proceso de evaluación hasta la fecha, Rijndael ha sido implementado en todo tipo de plataformas, tanto de Hardware como de Software. Algunas de

estas implementaciones, han manipulado las especificaciones del AES, con el fin de obtener un incremento en la eficiencia.

La aprobación por parte del gobierno norteamericano al algoritmo Rijndael como un estándar, le da una "certificación de calidad". El AES ha sido enviado a la Organización Internacional para la Estandarización (*ISO* por sus siglas en inglés) y al grupo de trabajo para la ingeniería de Internet (*IETF* por sus siglas en inglés), así como a la IEEE para que lo adopten como estándar [6].

Las características principales que favorecen la rápida adopción de Rijndael es el hecho de ser un código abierto libre de derechos de autor, y que gracias a su calidad, puede ser implementado de manera sencilla en una amplia variedad de plataformas sin que el rendimiento y la eficiencia se vean reducidos de manera considerable [6].

2.2. Los componentes del AES

En esta sección se explica la estructura del AES, así como los bloques que lo constituyen. Se describirán las rondas de transformación que conforman a este cifrador, así como el algoritmo para la expansión de la llave secreta.

2.2.1. Datos de entrada y salida

Los datos de entrada y salida del AES son consideradas como arreglos de bytes unidimensionales. Para el cifrado, la entrada es un bloque del texto en claro y una llave y la salida es un bloque de texto cifrado. Para el descifrado, la entrada es un texto cifrado y una llave; mientras que la salida es el texto en claro. Como se mencionó al inicio de este capítulo, en esta tesis, nos enfocaremos únicamente en la etapa de cifrado del AES.

La figura 2.1 ilustra el diagrama a bloques del proceso de cifrado con el AES. Como se muestra en la figura, la entrada corresponde a un bloque del texto en claro de 128 bits y a una llave de cifrado de la misma longitud. La salida corresponde también a un bloque del texto cifrado de 128 bits.

Cada ronda de transformación del AES, y los pasos que la conforman, operan con una matriz temporal, denominada *matriz de estado* de 4×4. Así mismo, la llave es organizada en una matriz de la misma dimensión. La figura 2.2 muestra la manera en que el texto en claro (a) y la llave (b) son acomodados en su respectiva matriz.

Todas las operaciones referentes al AES, son realizadas en el campo finito $GF(2^8)$, en el cual los operandos son considerados polinomios dentro de ese campo. En el apéndice B se dan las bases matemáticas necesarias para compreder dicha aritmética.

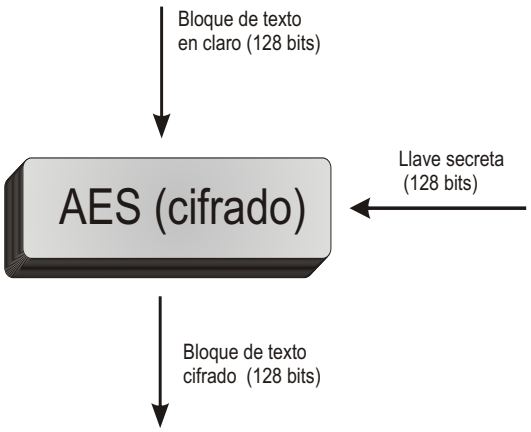


Figura 2.1: Digrama a Bloques del AES (cifrado)

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}

(a) Distribución de la matriz de entrada

K_0	K_4	K_8	K_{12}
K_1	K_5	K_9	K_{13}
K_2	K_6	K_{10}	K_{14}
K_3	K_7	K_{11}	K_{15}

(b) Distribución de la llave secreta

Figura 2.2: La organización de la información de entrada en el AES

2.2.2. Las rondas del AES

El AES es un cifrador por bloques iterativo, esto es, consiste en 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 llave de ronda para la i-ésima ronda se denota como *ExpandedKey[i]*, y *ExpandedKey[0]* denota la llave de entrada para la ronda inicial. La obtención de *ExpandedKey* de la llave de cifrado se denomina *KeyExpansion* y es explicada en las siguientes secciones.

La ronda de transformación se compone de 4 pasos: *SubBytes,ShiftRows,MixColumns y AddRoundKey*, la ronda final no incluye la aplicación de *MixColumns*. La figura 2.3 ilustra los pasos en el proceso de cifrado para el AES.

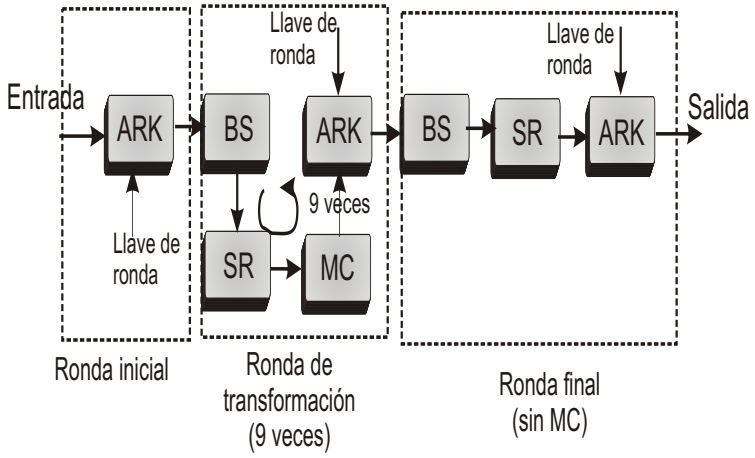


Figura 2.3: El proceso de Cifrado en el AES

Sustitución de Bytes: *SubBytes* **(BS)**

La sustitución de bytes o *SubBytes* es la única transformación no lineal del cifrador. *SubBytes* es una permutación basada en bloques pequeños, la cual consiste en la aplicación de una caja de sustitución (denominada *S-BOX*) a los bytes de la matriz de estado con el propósito de considerar todas las combinaciones, la caja S es una tabla de 256 bytes. En [6], se denota como S_{RD} a la caja de sustitución utilizada para el AES. La figura 2.4 ilustra el efecto de la transformación por *SubBytes* a una matriz de estado.

Conforme se describe en [6], el criterio seguido para diseñar S_{RD} se basó en las siguientes condieraciones listadas de acuerdo a su orden de importancia:

1. **No linealidad.**

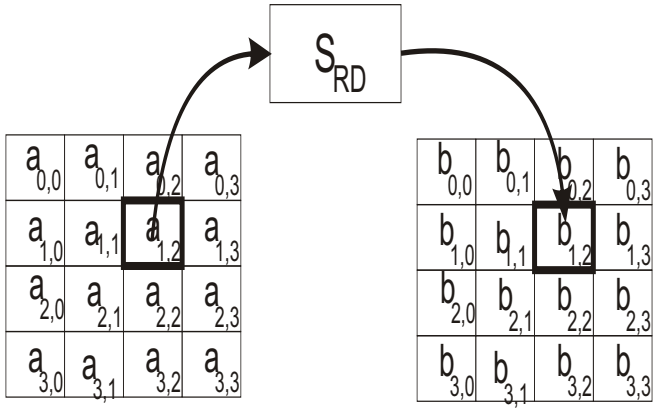


Figura 2.4: El paso SubBytes del AES

- **Correlación:** La máxima correlación entre la entrada y la salida debe tener la menor amplitud posible.
- **Probabilidad de la 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 $GF(2^8)$ (véase sección B.4) tiene que ser una expresión lo suficientemente fuerte para resistir ataques de interpolación.

La misma *S-BOX* es usada para cada uno de los bytes en la matriz de estado. Para una mejor descripción de la selección de la *S-BOX* véase [6]. La *S-BOX* consiste en un arreglo de 256 bytes.

Corrimiento de Renglones: *ShiftRows* (SR)

El paso de *ShiftRows* es una transposición de bytes que realiza corrimientos circulares con diferentes desplazamientos a los renglones de la matriz de estado. El renglón 0 es desplazado C_0 bytes, el 1 C_1 bytes, el renglón 2 C_2 bytes y el 3 C_3 bytes, por lo que cada byte en la posición j en el renglón i se desplaza a la posición $(j - C_i)$ mod 4.

El criterio seguido para los desplazamientos fue el de "difusión óptima" [28], el cual requiere que los cuatro desplazamientos sean diferentes.

Para el AES los desplazamientos son: $C_0 = 0, C_1 = 1, C_2 = 2$ y $C_3 = 3$.

La figura 2.5 muestra la operación de *ShiftRows* en una matriz de estado y el resultado obtenido.

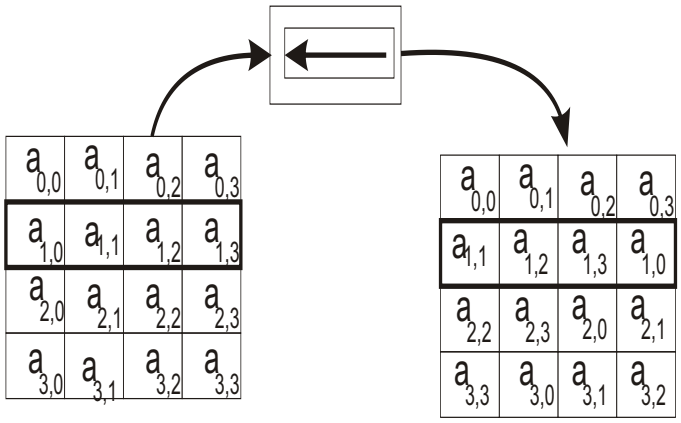


Figura 2.5: El paso ShiftRows del AES

El mezclado de columnas: *MixColumns* **(MC)**

El paso *MixColumns* es una permutación basada en bloques sobre la matriz de estado realizada columna a columna, la cual puede ser descrita como una multiplicación de la matriz de estado por una matriz constante. La aritmética del AES, debe hacerse en campos finitos binarios $GF(2^8)$, dado por el pentanomio irreducible $P(x) = x^8 + x^4 + x^3 + 1$. En este sentido los operandos se consideran polinimios (véase el apéndice B para las bases matemáticas correspondientes).

En esta transformación, cada columna de la matriz de estado es considerada un polinomio sobre $GF(2^8)$ (véase sección B.4) y es multiplicado por un polinomio $c(x)$ módulo $x^4 + 1$. El polinomio está dado por la ecuación 2.1:

$$c(x) \quad = \quad 03 \cdot x^3 + 01 \cdot x^2 + 01 \cdot x + 02 \tag{2.1}$$

Sea $b(x) = c(x) \cdot a(x) \bmod x^4 + 1$, entonces la multiplicación modular con un polinomio fijo puede ser escrita como se muestra en la ecuación 2.2.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{2.2}$$

MixColumns opera en las columnas de la matriz de estado como se muestra en la figura 2.6. El criterio de diseño para *MixColumns* incluye la dimensión, linealidad, difusión y rendimiento en un procesador de 8 bits. El criterio de dimensión define la operación de transformación en columnas de 4 bytes.

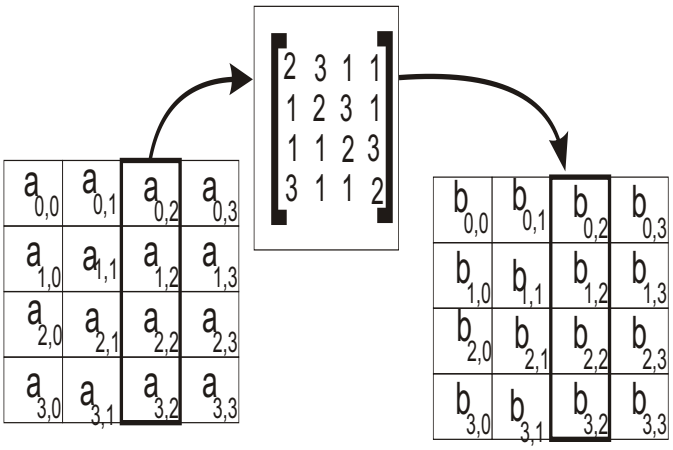


Figura 2.6: El paso MixColumns del AES

La suma de la llave: AddRoundKey (ARK)

Esta ronda consiste en la combinación de la llave de ronda con la matriz de estado mediante la operación lógica XOR. Una llave de ronda es denotada por *ExpandedKey[i]*, $0 \leq i \leq 10$. El arreglo de llaves de ronda es denominado *ExpandedKey* y es derivado de la llave de cifrado como se explica más adelante en la sección siguiente.

La llave de ronda tiene la misma longitud que la matriz de estado (128 bits). El *AddRoundKey* es ilustrado en la figura 2.7. En la figura $a_{i,j}$ denota el byte correspondiente a la matriz de estado, mientas que $k_{i,j}$ al de la llave de ronda, así mismo $b_{i,j}$ simboliza al byte de la matriz de estado resultante.

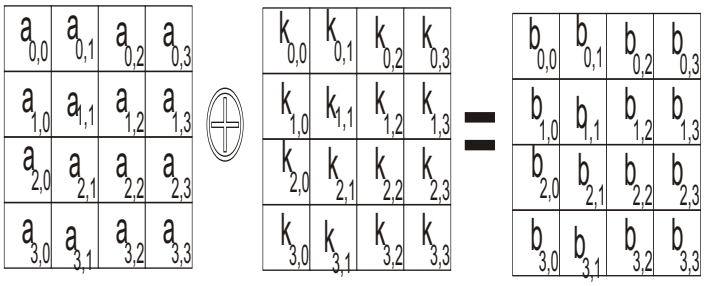


Figura 2.7: El paso AddRoundKey del AES

La generación de las llaves de ronda: *Key Schedule* **(KS)**

El KS consiste de dos componentes: la expansión de la llave y la selección de la llave de ronda. La expansión de la llave especifica cómo se deriva *ExpandedKey* a partir de la llave 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 cifrado; nunca debe ser especificada de manera directa.

El criterio de selección para el *KS* se basó en lo siguiente [6]:

- **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 2.3

$$w[i] = \begin{cases} w[i-4] \oplus w[i-1] & \text{si } i \bmod 4 \neq 0 \\ w[i-4] \oplus T(w[i-1]) & \text{de otra manera} \end{cases} \tag{2.3}$$

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:

$$(w(4i), w(4i+i), w(4i+2), w(4i+3)) \tag{2.4}$$

En la sección 2.3 se brindan algunas ideas para lograr una optimización los procesos de cálculo de *MC* y del *KS*.

2.3. Optimización de los pasos del AES

En [28] Saqib brinda algunas ideas para lograr un incremento en la eficiencia de algunos de los pasos que componen al AES. Dichas optimizaciones están pensadas para ser implementadas en hardware donde se puede aprovechar la paralelización inherente a los procesos.

Para los propósitos de esta tesis, utilizaremos las mejoras propuestas por Saqib para nuestra implementación del AES. Dicha implementación se describe en la sección 4.5. En esta sección daremos los fundamentos en los que se basan dichas implementaciones.

2.3.1. Optimización de *MixColumns* y *AddRoundKey*

En el proceso de cifrado, *MC* puede ser calculado en una manera eficiente utilizando únicamente tres pasos [6]: una suma, una multiplicación por 2 y una suma final. Sean $a[0]$, $a[1]$, $a[2]$ y $a[3]$ los elementos en una columna de la matriz de estado, entonces la columna transformada por *MC* sería: $a'[0]$, $a'[1]$, $a'[2]$ y $a'[3]$, la cual puede ser calculada como:

$$\begin{array}{c} t = a[0] \oplus a[1] \oplus a[2] \oplus a[3] \\ \begin{array}{lll} v = a[0] \oplus a[1]; & v = xtime(v); & a'[0] = a[0] \oplus v \oplus t; \\ v = a[1] \oplus a[2]; & v = xtime(v); & a'[1] = a[1] \oplus v \oplus t; \\ v = a[2] \oplus a[3]; & v = xtime(v); & a'[2] = a[2] \oplus v \oplus t; \\ v = a[3] \oplus a[0]; & v = xtime(v); & a'[3] = a[3] \oplus v \oplus t; \end{array} \end{array} \tag{2.5}$$

En la expresión ilustrada con la ecuación 2.5, $xtime(v)$ representa la multiplicación de campo de $02 \times v$, donde 02 es la constante polinomial x en $GF(2^8)$. Para obtener un mejor rendimiento, los cálculos de la expresión 2.5 pueden ser reestructurados y agregársele las operaciones del paso *ARK* con la finalidad de aprovechar los recursos del *FPGA*. La ecuación 2.6 ilustra el procedimiento:

$$\begin{array}{lll} v_0 = a[1] \oplus a[2] \oplus a[3]; & xt_0 = xtime(a[0]); & a'[0] = k[0] \oplus v_0 \oplus xt_0 \oplus xt_1; \\ v_1 = a[0] \oplus a[2] \oplus a[3]; & xt_1 = xtime(a[1]); & a'[1] = k[1] \oplus v_1 \oplus xt_1 \oplus xt_2; \\ v_2 = a[0] \oplus a[1] \oplus a[3]; & xt_2 = xtime(a[2]); & a'[2] = k[2] \oplus v_2 \oplus xt_2 \oplus xt_3; \\ v_3 = a[0] \oplus a[1] \oplus a[2]; & xt_3 = xtime(a[3]); & a'[3] = k[3] \oplus v_3 \oplus xt_3 \oplus xt_0; \end{array} \tag{2.6}$$

Con esta modificación se aprovecha la estructura del *FPGA*, pero dado que la ronda final (como se ilustra en la figura 2.3) no utiliza el paso de *MC*, *ARK* tiene que ser implementado de manera independiente para la ronda inicial y para la final.

2.3.2. Optimización en el Algoritmo de *KS*

Como se ha mencionado anteriormente, la llave original es representada como una matriz de bytes de 4×4, y las columnas de esta matriz son expandidas para poder obtener 40 columnas más.

Sean $w[0]$, $w[1]$, $w[2]$, $w[3]$ las columnas de la matriz de la llave representadas como:

$$\begin{array}{ll} w[0] = \begin{bmatrix} k_0 \\ k_1 \\ k_2 \\ k_3 \end{bmatrix} & w[1] = \begin{bmatrix} k_4 \\ k_5 \\ k_6 \\ k_7 \end{bmatrix} \\ w[2] = \begin{bmatrix} k_8 \\ k_9 \\ k_{10} \\ k_{11} \end{bmatrix} & w[3] = \begin{bmatrix} k_{12} \\ k_{13} \\ k_{14} \\ k_{15} \end{bmatrix} \end{array} \tag{2.7}$$

De acuerdo a la ecuación 2.7, las nuevas columnas $w'[0]$, $w'[1]$, $w'[2]$ y $w'[3]$ para la siguiente llave de ronda pueden ser calculadas conforme a la ecuación 2.8:

$$\begin{array}{cccc} \text{Paso 1} & \text{Paso 2} & \text{Paso 3} & \text{Paso 4} \\ k'_0 = k_0 \oplus SBox(k_{13}) \oplus r_{con}; & k'_4 = k_4 \oplus k'_0; & k'_8 = k_8 \oplus k'_4; & k'_{12} = k_{12} \oplus k'_8; \\ k'_1 = k_1 \oplus SBox(k_{14}); & k'_5 = k_5 \oplus k'_1; & k'_9 = k_9 \oplus k'_5; & k'_{13} = k_{13} \oplus k'_8; \\ k'_2 = k_2 \oplus SBox(k_{15}); & k'_6 = k_6 \oplus k'_2; & k'_{10} = k_{10} \oplus k'_6; & k'_{14} = k_{14} \oplus k'_{10}; \\ k'_3 = k_3 \oplus SBox(k_{12}); & k'_7 = k_7 \oplus k'_3; & k'_{11} = k_{11} \oplus k'_7; & k'_{15} = k_{15} \oplus k'_{11}; \end{array} \tag{2.8}$$

En [28] Saqib analiza algunas optimizaciones que pueden ser realizadas al algoritmo del *KS*. Reduce los cuatro pasos de la ecuación 2.8 a únicamente dos pasos. Esta optimización utiliza cálculos redundantes y paraleliza el proceso, por lo que tenemos:

$$\begin{array}{cc} \text{Paso 1} & \text{Paso 2} \\ k'_0 = k_0 \oplus SBox(k_{13}) \oplus rcon; & \begin{array}{l} k'_4 = k_4 \oplus k'_0; \\ k'_8 = k_8 \oplus k_4 \oplus k'_0; \\ k'_{12} = k_{12} \oplus k_8 \oplus k_4 \oplus k'_0; \end{array} \\ k'_1 = k_1 \oplus SBox(k_{14}); & \begin{array}{l} k'_5 = k_5 \oplus k'_1; \\ k'_9 = k_9 \oplus k_5 \oplus k'_1; \\ k'_{13} = k_{13} \oplus k_9 \oplus k_5 \oplus k'_1; \end{array} \\ k'_2 = k_2 \oplus SBox(k_{15}); & \begin{array}{l} k'_6 = k_6 \oplus k'_2; \\ k'_{10} = k_{10} \oplus k_6 \oplus k'_2; \\ k'_{14} = k_{14} \oplus k_{10} \oplus k_6 \oplus k'_2; \end{array} \\ k'_3 = k_3 \oplus SBox(k_{12}); & \begin{array}{l} k'_7 = k_7 \oplus k'_3; \\ k'_{11} = k_{11} \oplus k_7 \oplus k'_3; \\ k'_{15} = k_{15} \oplus k_{11} \oplus k_7 \oplus k'_3; \end{array} \end{array} \tag{2.9}$$

Con la optimización mostrada en la ecuación 2.9, la implementación de la expansión de llave, puede ser realizada de una manera eficiente.

Algunas otras optimizaciones pueden ser encontradas en [32] donde Standaert brinda ideas de optimización para la implementación del AES en FPGAS. El análisis hecho por Standaert puede ser utilizado para mejorar la implementación del AES conseguida en esta tesis, con la finalidad de obtener un mejor rendimiento del modo de operación CCM.

El siguiente capítulo brinda la explicación de la implementación, tanto del CCM como del AES.

Capítulo 3

El modo de operación CCM

En este capítulo se darán los detalles sobre este modo de operación genérico. Se explicará la manera en que la autenticación y el cifrado son realizados, así como los detalles necesarios para llevar a cabo el descifrado y la verificación. El capítulo está organizado de la siguiente forma, primero damos una introducción a la teoría del CCM, especificaremos los parámetros de entrada, y finalmente cubriremos la autenticación y el cifrado junto con el descifrado y la verificacion.

3.1. Introducción al modo de operación CCM

Los esquemas de autenticación cifrada (AE, por sus siglas en inglés) son mecanismos de llave simétrica, por medio de los cuales un mensaje M es transformado en un texto cifrado C de tal forma que C protege tanto la privacidad como la autenticidad. El algoritmo de cifrado utiliza una llave, un texto en claro y un vector de inicialización (IV), y regresa un texto cifrado. El algoritmo de descifrado utiliza una llave, un texto cifrado válido y un IV, y regresa un texto en claro o bien un símbolo especial llamado *Inválido* [25].

En este capítulo nos referiremos al IV como *nonce* (un número aleatorio), con la finalidad de reflejar los requerimientos que se darán para dicho *nonce*.

Es necesario destacar que además de cumplir con el objetivo de privacidad un AE debe cumplir, con la propiedad de autenticidad: si un adversario trata de crear un texto cifrado, el algoritmo de descifrado deberá identificarlo como *inválido* con una certeza muy confiable [25].

Un esquema AE puede ser construido mediante la combinación apropiada de un esquema de cifrado y un código de autenticación de mensaje (MAC) . Esta estrategia es utilizada en la práctica y en los estándares [25].

En esta sección explicaremos la teoría sobre el funcionamiento del esquema de autenti-

cación cifrada, conocido como el modo de operación CCM.

CCM es el nombre corto para *Counter with CBC-MAC* (Contador con CBC-MAC [*Cipher Block Chaining*, cifrado Encadenado de Bloques]). Esto significa que dos modos de operación diferentes son combinados en uno, el modo CTR y el CBC-MAC. El modo CCM es un modo genérico de cifrado y autenticación para un cifrador por bloques y está definido para ser utilizado con cifradores por bloques de 128 bits, tal como el AES . Las ideas planteadas en el CCM pueden ser fácilmente extendidas para otros tamaños de bloque, pero esto requeriría definiciones adicionales.

El modo CCM combina la privacidad del modo de Conteo (*Counter CTR*) y la autenticación del CBC-MAC. Estos modos han sido usados y estudiados desde hace bastante tiempo, y tienen propiedades criptográficas bien entendidas. Proveen una buena seguridad y rendimiento tanto en hardware como en software [11]. Este modo fue propuesto por Whiting et al. [41] en el año 2002. Su artículo original fue enviado al Instituto Nacional de Estándares y Tecnología de los Estados Unidos (*NIST* por sus siglas en inglés), para ser aceptado como un modo de operación genérico. La IEEE lo ha adoptado en su estándar 802.11 en su nueva versión "i".

Como todo estándar que sea aceptado, el CCM ha sido objeto de múltiples análisis para encontrar sus fortalezas y debilidades. Ejemplos de dichos análisis pueden ser encontrados en [11], donde se hace un análisis de la seguridad que brinda este modo de operación. En [35] Struick realiza un análisis de la especificación liberada por el *NIST*, así mismo describe algunas desventajas encontradas en este modo de operación.

De acuerdo al análisis de Jonsson en [11], el nuevo modo de operación CCM, cuenta con las siguientes ventajas:

1. Maneja mensajes en los cuales una parte de la información se desea únicamente autenticar y no cifrar, lo cual es realizado de manera sencilla sin incurrir en una pérdida en la eficiencia del sistema. Para algunos otros modos de operación es necesario implementar mejoras para poder realizar esta operación.

2. El CCM utiliza únicamente la operación para el ciframiento del cifrador por bloques, esto aplica tanto para el proceso de cifrado como descifrado del CCM. Esta característica hace atractivo al CCM, para aplicaciones en las cuales se desea un tamaño de código pequeño, además de ser posible utilizar una función que genere permutaciones pseudoaleatorias, la cual no sea necesariamente reversible, lo que convierte al CCM en uno de los modos más versátiles.

3. Está basado en dos tecnologías (CTR y CBC-MAC) ampliamente utilizadas y documentadas alrededor del mundo, lo que permite ahorrar tiempo en implementaciones, pues existen estrategias eficientes reportadas para su implementación en diferentes plataformas de desarrollo.

4. Según [41], todos los derechos intelectuales para el CCM han sido liberados para el uso público.

Nombre	Descripción	Tamaño del campo	Codificación del campo
M	Número de bytes del campo de autenticación	3 bits	$(M-2)/2$
L	Número de bytes en el campo de longitud de mensaje	3 bits	$L-1$

Cuadro 3.1: Descripción de los parámetros del CCM [41]

Sin embargo, según [35], el CCM tiene ciertas desventajas cruciales, las cuales son listadas a continuación:

- El criterio seguido por el *NIST* para la selección del CCM no fue claro.
- Está definido únicamente para cifradores de 128 bits.
- Requiere que la longitud de los datos de entrada sean conocidos con anterioridad.
- El modo original del CCM no define la opción de brindar únicamente confidenciabilidad, sino que tiene que existir autenticación de datos de manera obligatoria, lo cual no es útil o necesario si una entidad externa brinda la autenticación de la información.
- Es susceptible a ciertos ataques, si se utiliza con campos de autenticación de tamaño variable, en lugar de usarlo con campos de longitud fija.

3.1.1. Los parámetros del CCM

Para el modo genérico del modo CCM existen dos parámetros que tienen que ser elegidos por el usuario. El primer parámetro es M, que representa el tamaño en bytes del campo de autenticación. La elección del valor para M involucra un compromiso entre la expansión del mensaje y la probabilidad que un atacante pueda modificar el mensaje sin ser detectado. Los valores válidos son $4, 6, 8, 10, 12, 14$ y 16 bytes. Este parámetro es codificado como $(M-2)/2$

El segundo parámetro es L, que representa el tamaño del campo de longitud ($l(m)$) de mensaje. Este valor requiere un compromiso entre la extensión máxima del mensaje y el largo del *Nonce*. Los valores válidos para L están en el rango entre 2 y 8 bytes (el valor $L = 1$ está reservado). L se codifica como $L-1$. La tabla 3.1[41] resume el significado de los parámetros.

3.1.2. Las entradas en este modo de operación

Para enviar un mensaje, el emisor debe proveer la siguiente información:

- Una llave de cifrado K, la cual debe ser adecuada para el cifrador por bloques que se utilice.
- Un número de inicialización *Nonce* N de longitud $15 - L$ bytes. El valor del *nonce* debe ser único, esto significa que los valores de N utilizados con la llave K no deben repetirse.
- El mensaje m, que consiste en una cadena de $l(m)$ bytes donde $0 \leq l(m) < 2^{8L}$.
- Datos adicionales (opcionales) a, que consisten de una cadena de $l(a)$ bytes donde $0 \leq l(a) < 2^{64}$. Estos datos adicionales son autenticados pero no cifrados, y no son incluidos en la salida de este modo. $l(a)$ es codificado con 2 a 10 bytes según la tabla descrita en [41]. Para este trabajo el campo $l(a)$ siempre ocupará únicamente 2 bytes.

La figura 3.1(a), muestra el diagrama a bloques del modo de operación CCM del lado del Emisor. Por otro lado, en (b), se ilustran los procesos a seguir para el receptor. En la ilustración se muestran las diferentes etapas que conforman a este modo de operación, las cuales serán explicadas en las siguientes secciones.

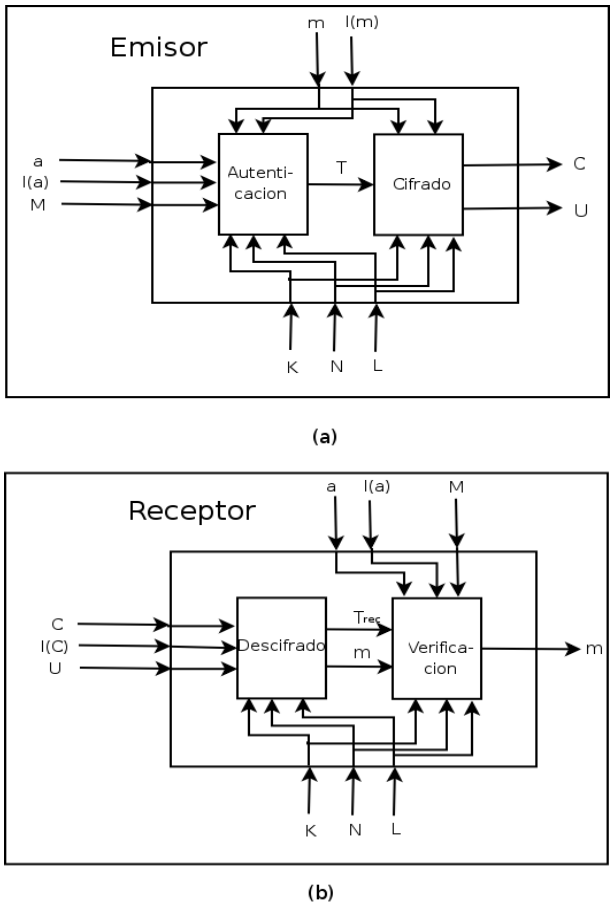


Figura 3.1: Digrama a Bloques del Modo CCM

3.2. Autenticación

La autenticación de la información es el primer paso en el proceso del CCM. Para realizar la autenticación del mensaje, es necesario utilizar el CBC-MAC para obtener el campo de autenticación T. Debido a que el CBC-MAC procesa bloques de información de una longitud kb fija, no puede ser aplicado de forma directa a los datos de entrada del modo CCM (la cual puede tener una longitud arbitraria y puede no ser múltiplo de kb).

Conforme a lo anterior es necesario definir una secuencia de bloques $B_0, B_1, ..., B_n$ y calcular el CBC-MAC de estos bloques.

De acuerdo a las especificaciones dadas en [41], el primer bloque B_0 se forma como se indica en la figura 3.2.

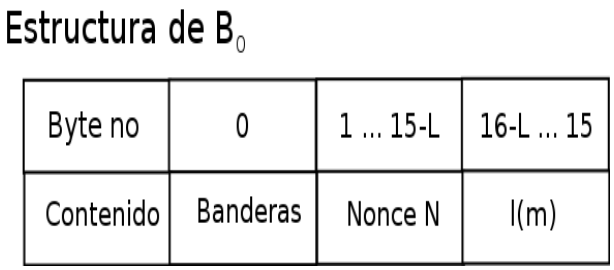


Figura 3.2: Estructura del Bloque B_0

Donde, $l(m)$ está codificado con el orden de primero el bit más significativo. El contenido del byte de banderas para B_0 se ilustra en la figura 3.3.

Figura 3.3: Estructura de las banderas para B_0

Como se aprecia en la figura 3.3, el bit 7 está reservado para futuras expansiones y debe ser puesto siempre en 0. El bit Adata (6) debe ser cero si $l(a) = 0$ o bien uno si $l(a) > 0$.

Una vez que se ha formado el bloque B_0, es necesario formar los bloques restantes para aplicarles el CBC-MAC. Para hacerlo deben realizarse los siguientes pasos:

1. Para formar los bloques B_1 a B_k, es necesario concatenar a $l(a)$ (codificado) la información adicional a y dividir el resultado en bloques de 16 bytes, agregando ceros al último bloque en caso de ser necesario.

2. Cuando los bloques B_1 a B_k han sido formados con los datos (opcionales) para autenticar, es necesario agregar los bloques del mensaje. Estos bloques de mensaje se construyen mediante la división de m en bloques de 16 bytes, agregando ceros, si es necesario, al final. Si el mensaje m es una cadena vacía, entonces no se agregan bloques en este paso.

El resultado obtenido es la secuencia de bloques $B_0, B_1, ..., B_n$. Entonces el CBC-MAC se calcula como sigue:

$$X_1 = E_k(B_0) \tag{3.1}$$
$$X_{i+1} = E_k(X_i \oplus B_i) \text{ for} i = 1, ..., n \tag{3.2}$$
$$T = firstMBytes(X_{n+1}) \tag{3.3}$$

Donde $E()$ representa la operación de cifrado con el cifrador por bloques. T es el campo de autenticación de M bytes. Debe notarse que el último bloque (B_n), es sumado mediante la operación de XOR, con X_n y el resultado es cifrado con el cifrador por bloques. En caso de ser necesario, el texto cifrado es truncado con la función firstMBytes para obtener T de la longitud deseada.

Las ecuaciones 3.1 a 3.3, pueden ser implementadas con el algoritmo 6.

Algoritmo 6 Algoritmo de autenticación

1: Sean $B_0, B_1, ..., B_n$ los bloques de autenticación
2: $X_1 = E_k(B_0)$
3: **for** $i = 1$ to n **do**
4: $\quad X_{i+1} = E_k(B_i \oplus X_i)$
5: **end for**
6: $T = firstMBytes(X_{n+1})$

Como puede observarse, el algoritmo del CBC-MAC es bastante sencillo de implementar. Sin embargo, una característica importante es el hecho de ser un algoritmo que utiliza el encadenamiento, es decir, que cada nuevo texto cifrado depende del anterior. Gracias a este encadenamiento, la autenticación de la información se hace más confiable, por lo que una modificación mínima en el mensaje, provocará que el CBC-MAC, calcule erróneamente el parámetro de autenticación, con lo que se puede detectar cualquier alteración mal intencionada o accidental.

Lo anterior ocasiona que el proceso de autenticación no pueda ser paralelizado en su implementación, debido a la gran dependencia de datos que existe, lo que lo hace un algoritmo más lento que si fuese paralelizable. Las consecuencias de no poder paralelizar el algoritmo de autenticación serán explicadas en la sección 4.4.

En la figura 3.4 se ilustra el proceso de autenticación en el modo CCM, se muestra la manera en que se da el encadenamiento de la información y por lo tanto, la dependencia de datos que existe.

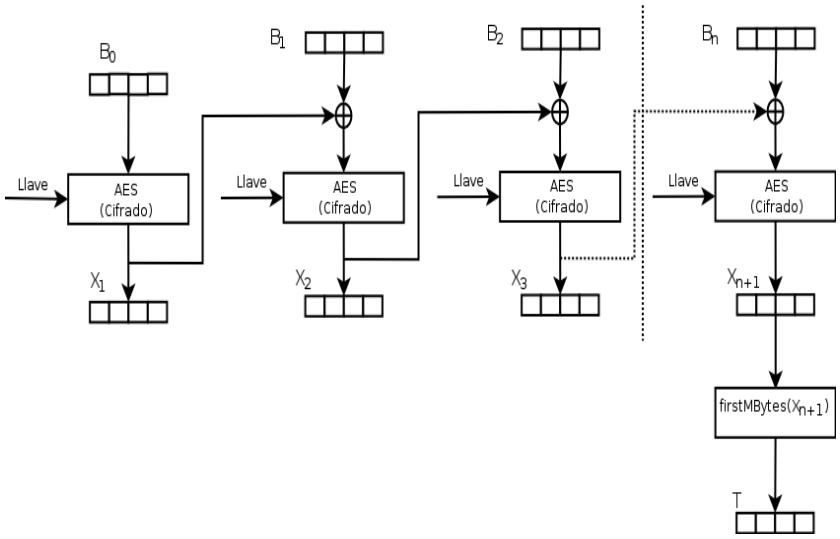


Figura 3.4: El proceso de autenticación en CCM

El CBC-MAC tiene algunas propiedades de seguridad que son atractivas, en [11] se brinda un análisis de algunas de las consideraciones de seguridad a ser observadas.

3.3. Cifrado

Para realizar el cifrado de un mensaje en el modo CCM, se utiliza el modo de Conteo (*CTR* por sus siglas en inglés), para lo cual es necesario definir un flujo de bloques como sigue:

$$S_i := E(K, A_i) \text{ for } i = 0, 1, 2, ..., \tag{3.4}$$

De acuerdo a [41], los bloques A_i se forman como se ilustra en la figura 3.5.

Los valores para i están codificados de la forma del byte más significativo primero, y representa una función de conteo establecida por el usuario. Dicho conteo se incrementa

Estructura de A_i

Byte no	0	1 ... 15-L	16-L ... 15
Contenido	Banderas	Nonce N	Contador i

Figura 3.5: Estructura del Bloque A_i

conforme se crean bloques A_i.

Para cada bloque A_i, el campo de banderas está constituido por los valores ilustrados en la figura 3.6.

Estructura de las banderas para A_i

Bit numero	7	6	5	4	3	2	1	0
Contenido	Reservado	Reservado	0			L		

Figura 3.6: Estructura de las Banderas para los Bloques A_i

Al igual que en las banderas del bloque B_0, los bits que aparecen como reservados, serán utilizados en futuras expansiones por lo que deben ser puestos en cero. Nótese también que los bits $5, 4$ y 3, son puestos a cero, lo que garantiza que todos los A_i, serán distintos de B_0 y así asegurar que los dominios de la autenticación y cifrado sean diferentes y por lo tanto no comprometan la seguridad.

Para llevar a cabo el cifrado es necesario realizar los siguientes pasos:

1. El mensaje es cifrado mediante la operación XOR de m con los primeros $l(m)$ bytes de la concatenación de $S_1, S_2, S_3, ...$ El resultado de esta operación es el texto cifrado C. Debe notarse que S_0 no es utilizado para cifrar el mensaje.

2. El parámetro de autenticación T es cifrado con S_0 como se muestra en la ecuación 3.5, y se trunca a la longitud deseada es decir el valor de M (véase tabla 3.1).

$$U := T \oplus firstMbytes(S_0) \tag{3.5}$$

El resultado del proceso de cifrado es el texto cifrado C seguido por el parámetro de autenticación cifrado U, los cuales son enviados al receptor para que los descifre y los verifique.

El algoritmo 7 muestra el proceso para cifrar un mensaje m. Debe notarse que en el cifrado no se utilizan los datos a, puesto que son únicamente para ser autenticados, y no para ser cifrados.

Algoritmo 7 Algoritmo del proceso de cifrado

Requiere: m dividido en bloques de 16 bytes, formando $m_0, m_1, ..., m_n$, k y N

1: Sean $A_0, A_1, ..., A_n$ los bloques de conteo
2: $S_0 = E_k(A_0)$
3: **for** $i = 1$ to n **do**
4: $\quad S_i = E_k(A_i)$
5: $\quad C_{i-1} = S_i \oplus m_{i-1}$
6: **end for**
7: $U = T \oplus firstMBytes(S_0)$

Al igual que el algoritmo de autenticación con el CBC-MAC, el proceso de cifrado es bastante sencillo de implementar. A diferencia de la autenticación, el cifrado de la información sí puede ser paralelizado, puesto que no existe dependencia entre los datos.

Sin embargo, el hecho que el proceso pueda ser paralelizado, no puede ser explotado en su totalidad porque depende de la autenticación para ser completado. Lo anterior implica que ambos procesos deben terminar para proporcionar la información suficiente al receptor.

Al cifrar el campo de autenticación T, se evitan los ataques de colisión contra el CBC-MAC. Si el cifrador por bloques se comporta como una permutación pseudoaleatoria, entonces el bloque llave es indistinguible de una cadena aleatoria, por lo que el atacante no obtiene ninguna información útil de los resultados del CBC-MAC [41].

La figura 3.7 muestra la manera en que se lleva a cabo el proceso de cifrado en el modo CCM. Se ilustra la interacción de los datos y los pasos a seguir para obtener la cifra.

De acuerdo a las propiedades de seguridad, el *nonce* no debe repetirse, para garantizar que todos los bloques A_i de entrada del CTR y todos los bloques B_0 utilizados durante el tiempo de vida de la llave serán diferentes y de este modo evitar fisuras de seguridad.

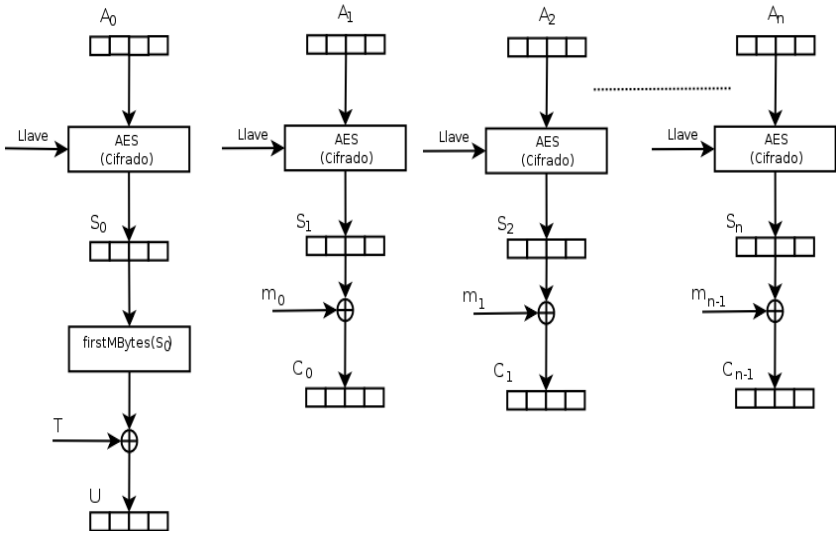


Figura 3.7: El proceso de cifrado en CCM

3.4. Descifrado

Para poder descifrar un mensaje, el receptor necesita la siguiente información:

- La llave K del cifrador.
- El *Nonce* N.
- El mensaje cifrado C.

El proceso de descifrado comienza con el cálculo del flujo de llaves para el CTR, es decir los bloques S_i, mediante el uso de la ecuación 3.4. Con los bloques S_i, se recupera el mensaje m, mediante la operación descrita en la ecuación 3.6.

$$m_{i-1} \quad = \quad S_i \oplus C_i \text{ for } i = 1, ..., n \tag{3.6}$$

En el proceso de descifrado también es necesario obtener el parámetro de autenticación T_{rec} a partir de U, para lo cual se realiza la ecuación 3.7.

$$T_{rec} \quad = \quad U \oplus firstMBytes(S_0) \tag{3.7}$$

El algoritmo 8 muestra los pasos a seguir para descifrar la información. Dicho algoritmo es el proceso inverso del utilizado para cifrar (véase el algoritmo 7).

En la figura 3.8 se ilustra la forma de llevar a cabo el proceso de descifrado. Como puede apreciarse, es muy similar al cifrado, con la diferencia de que el descifrado utiliza los datos

Algoritmo 8 Algoritmo del proceso de descifrado

Requiere: C dividido en bloques de 16 bytes, formando $C_0, C_1, ..., C_n$, k, T y N

1: Sean $A_0, A_1, ..., A_n$ los bloques llave para el cifrado

2: $S_0 = E_k(A_0)$

3: **for** $i = 1$ to n **do**

4: $\quad S_i = E_k(A_i)$

5: $\quad m_{i-1} = S_i \oplus C_{i-1}$

6: **end for**

7: $T_{rec} = U \oplus firstMBytes(S_0)$

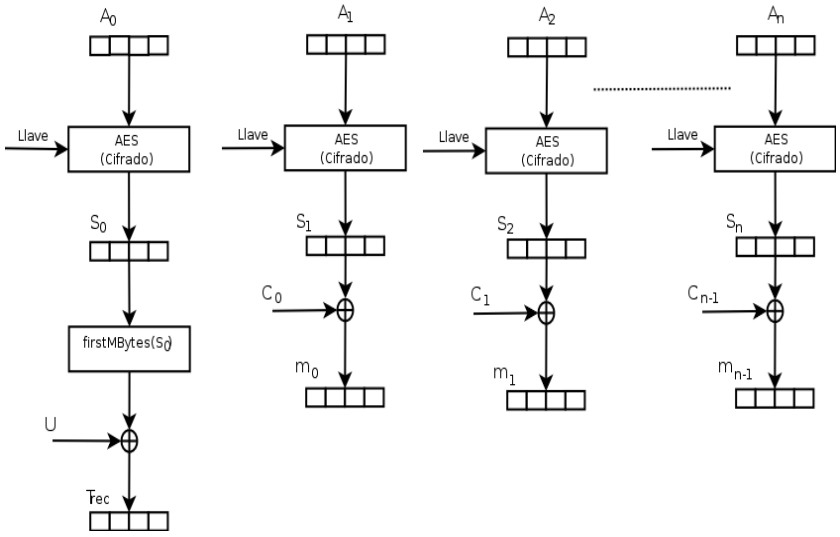


Figura 3.8: El proceso de descifrado en CCM

cifrados recibidos y obtiene el mensaje original.

Con la información obtenida en el proceso de descifrado, se procede a realizar la verificación de los datos recibidos.

3.5. Verificación

Una vez que el mensaje m y el parámetro de autenticación T_{rec}, han sido descifrados, es necesario realizar la operación opuesta a la autenticación, es decir, la verificacion.

La verificación de la información es importante debido a que con ella se garantiza que los datos no hayan sufrido modificaciones, tanto mal intencionadas como accidentales, durante

la transmisión.

En el modo CCM el proceso de verificación se lleva a cabo de la siguiente manera:

1. Con el mensaje m calculado durante el descifrado y los datos adicionales para autenticar a, se calcula el CBC-MAC como se describió en la sección 3.2, para obtener el parámetro de autenticación T_{calc}.

2. Realizamos la verificación de la información, comparando T_{rec} y T_{calc}. Si son iguales, entonces la información no sufrió modificaciones y puede ser utilizada, de lo contrario, es desechada y se solicita un reenvío de la misma.

El algoritmo 9, describe el proceso de verificación.

Algoritmo 9 Algoritmo de verificacion

Requiere: m descifrado, T_{rec}, k.

Sean $B_0, B_1, ..., B_n$ los bloques de autenticación

$X_1 = E_k(B_0)$

for $i = 1$ to n **do**

    $X_{i+1} = E_k(B_i \oplus X_i)$

end for

$T_{calc} = firstMBytes(X_{n+1})$

if $T_{calc} = T_{rec}$ **then**

    La información es aceptada y utilizada.

else

    Los datos son desechados y se solicita sean reenviados

end if

La figura 3.9 ilustra la manera en que se lleva a cabo la verificación de la información recibida.

Con el proceso de verificación termina el CCM. A continuación se dará un breve análisis de la seguridad de este modo de operación.

3.6. La seguridad del modo CCM

Debido a que el modo CCM, está pensado para ser utilizado en medios inseguros, es necesario que su seguridad esté garantizada.

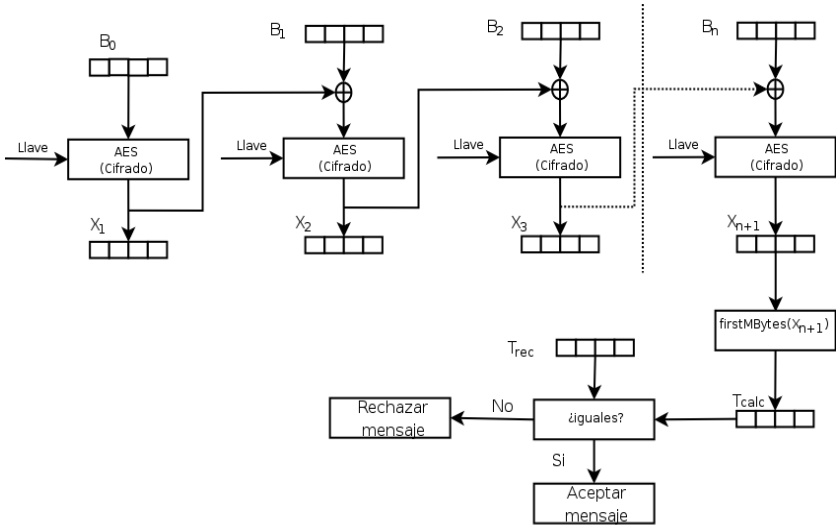


Figura 3.9: El proceso de verificacion en CCM

En [11] Jonsson hace un análisis del método combinado denominado CCM, para el cual el hecho de estar basado en dos tecnologías ampliamente utilizadas no garantiza que sea seguro.

Uno de las principales fortalezas recalcadas por Jonsson, es que el CCM resuelve el problema de garantizar que la modificación del texto cifrado sea difícil para un oponente, a través de un parámetro de autenticación cifrado, a diferencia de algunos otros modos de operación que utilizan sumas de verificación. El agregar dicho parámetro de autenticación hace menos eficiente a este modo de operación, pero a cambio brinda los beneficios listados en la sección 3.1.

Jonsson centra su análisis en dos aspectos fundamentales:

1. **Privacidad:** Debe ser imposible para un adversario obtener información útil a partir del texto cifrado, sin el conocimiento de la llave secreta.

2. **Autenticidad:** Debe ser imposible para un adversario generar un texto cifrado válido sin el acceso a la llave secreta.

La conclusión mostrada por Jonsson implica que el modo CCM, brinda un nivel de seguridad adecuado para cumplir con sus objetivos de diseño. Jonsson muestra que de acuerdo a sus experimentos la probabilidad de un atacante para comprometer la seguridad del CCM, tanto en privacidad y autenticidad, se encuentra en los límites aceptados para ser considerado seguro.

Para una mejor comprensión de los análisis de seguridad realizados al modo CCM, véase [11].

3.7. Pruebas de rendimiento del CCM en tarjetas comerciales

Con la finalidad de obtener un parámetro de referencia para el desempeño del modo de operación CCM, realizamos pruebas de transmisión con una tarjeta comercial que tiene soporte para este nuevo esquema de seguridad. En esta sección se describen las tarjetas utilizadas y el experimento realizado. Se presentan además los resultados obtenidos.

En el apéndice C se da la información referente al driver utilizado, así como la instalación del mismo. Se presentan las aplicaciones que fueron programadas para realizar los experimentos.

3.7.1. Descripción de las tarjetas inalámbricas

Para esta prueba se utilizaron dos tarjetas inalámbricas, las cuales corresponden al modelo 3CRDW696 de 3COM, cuyas características se enlistan a continuación [3]:

- Conexiones seguras con cifrado WEP de 40 y 128 bits
- El soporte 802.1x proporciona autenticación de usuario mediante servidor RADIUS
- Características similares a las de una red cableada con velocidades de hasta 11 Mbps y distancias de hasta 100 metros (328 pies) en interiores
- Sencilla instalación con funcionamiento plug-and-play y utilidad de configuración fácil de usar
- El cambio dinámico de velocidad garantiza la fiabilidad de las conexiones, incluso en condiciones de ruido
- El networking ad hoc permite conectar múltiples clientes inalámbricos, sin usar puntos de acceso
- Firmware actualizable a futuros estándares de seguridad y mejoras de funcionalidades
- La certificación Wi-Fi garantiza la interoperabilidad multiproveedor

3.7.2. Pruebas realizadas

Las pruebas que se llevaron a cabo consistieron en el envío de información a través de las tarjetas inalámbricas, utilizando las aplicaciones cliente y servidor. Dicha transmisión se hizo con diferentes esquemas de cifrado.

Se trabajó con 4 esquemas de cifrado:

1. Sin cifra
2. cifrado con WEP de 40 bits
3. cifrado con WEP de 104 bits
4. cifrado con AES-CCM (tamaño de llave de 128 bits)

Las pruebas consistieron en enviar un buffer de información de 1024 bytes. Dicho buffer fue enviado varias veces a través de las tarjetas inalámbricas. El número de veces que se envió el buffer fueron : $10,000$, $20,000$, $30,000$, $40,000$, $50,000$, $100,000$, $200,000$, $300,000$, $400,000$ y $500,000$.

Los experimentos se llevaron a cabo ejecutando la aplicación "cliente" (véase la sección C.5.2) dos veces para cada número de veces que se envió el buffer. Los tiempos registrados por la aplicación fueron promediados para obtener el tiempo final que se reporta en la sección 3.7.3.

Este experimento se repitió para cada uno de los esquemas de cifrado. Las tarjetas se configuraban con el correspondiente algoritmo de cifrado (véase la sección C.4).

3.7.3. Resultados de las pruebas

Una vez concluidas las pruebas, se recopiló la información contenida en los diferentes archivos de reporte que fueron generados.

La tabla 3.2 resume los tiempos promedio obtenidos en las pruebas.

La figura 3.10 muestra los tiempos de transmisión registrados en la tabla 3.2.

Como puede apreciarse en la figura 3.10, el método más rápido en la transmisión, es obviamente, el no utilizar ningún mecanismo de cifrado de la información, sin embargo, es también el más inseguro. Los algoritmos WEP, son prácticamente igual de rápidos, por lo que en la figura, sus gráficas se ven empalmadas.

Por otro lado, el modo CCM, es el más rezagado en esta gráfica comparativa, lo cual se debe a que el proceso de cifrado con el AES es considerablemente más lento que el realizado con RC4. Otro factor importante a considerar es el hecho que para las tarjetas utilizadas en esta prueba, la implementación del AES, es llevada a cabo en software, lo que lo hace todavía más lento. En relación con el WEP, el CCM, agrega 16 bytes extras a un paquete del estándar IEEE 802.11 sin cifrar, esto debido a la inclusión de la autenticación llevada a cabo con el CBC-MAC.

En el capítulo 5 se analizan más detalladamente los resultados obtenidos en esta prueba.

Número de Veces	Sin Cifrar	WEP 40 bits	WEP 104	AES-CCM
10,000	4.195 s	4.795 s	4.815 s	5.38 s
20,000	8.47 s	9.65 s	9.53 s	10.555 s
30,000	12.665 s	14.46 s	14.33 s	15.985 s
40,000	16.88 s	19.13 s	19.29 s	21.335 s
50,000	21.135 s	23.885 s	23.875 s	26.51 s
100,000	42.355s	47.675 s	47.88 s	53.425s
200,000	84.775 s	95.875 s	95.845 s	106.38 s
300,000	126.91 s	143.855 s	143.845 s	159.725 s
400,000	168.98 s	191.52 s	191.93 s	212.805
500,000	211.34 s	239.245 s	239.4 s	265.894 s

Cuadro 3.2: Tabla de resultados obtenidos

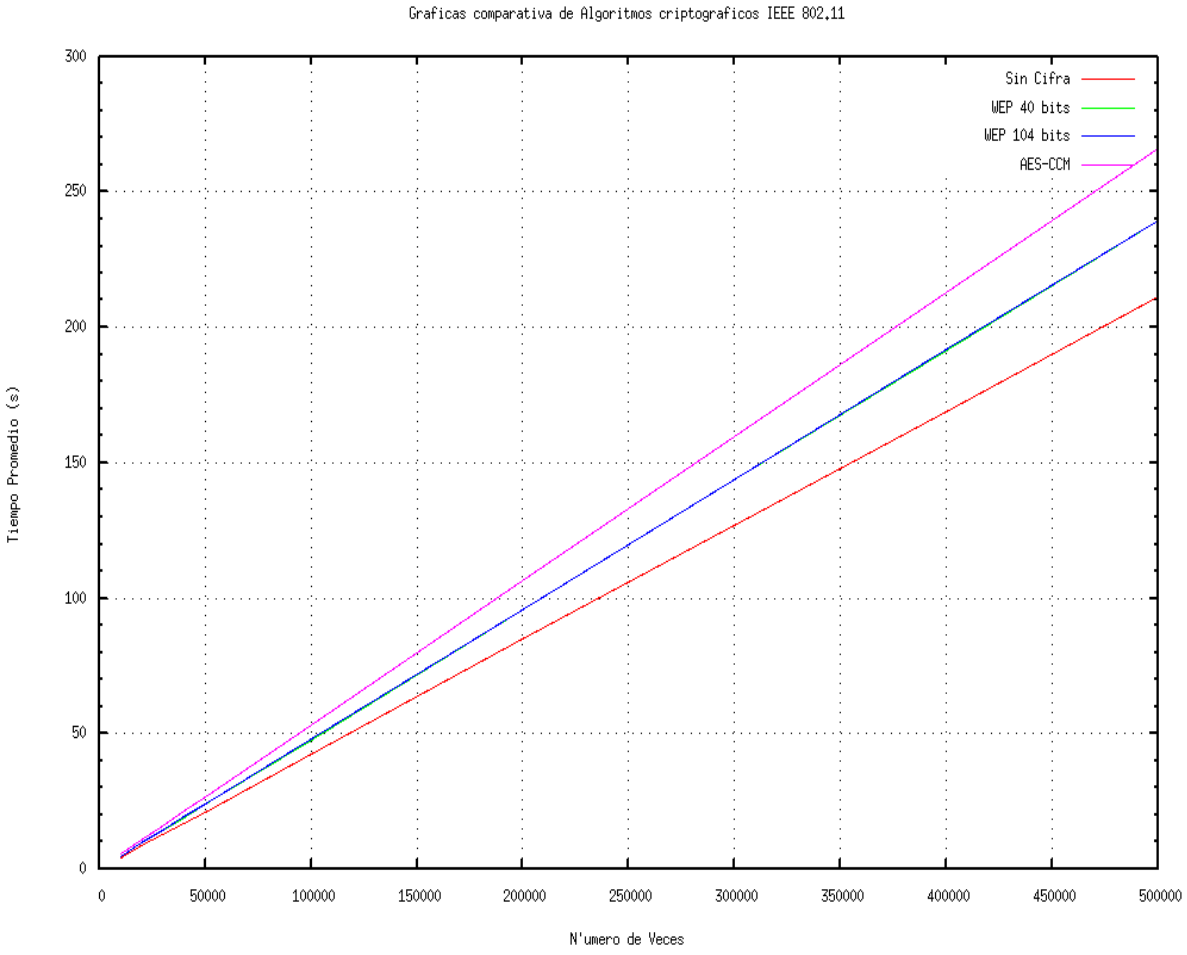


Figura 3.10: Gráfica comparativa de los algoritmos de cifrado

Capítulo 4

La implementación del CCM en FPGAs

En este capítulo se dan los lineamientos seguidos para llevar a cabo la implementación del modo de operación CCM en un FPGA. Iniciamos con una descripción de la plataforma de desarrollo, que incluye la descripción del chip utilizado, las herramientas y el lenguaje de programación. Posteriormente se dan las consideraciones que fueron tomadas en cuenta para desarrollar la implementación y finalmente se describe la manera en que se realizaron las implementaciones del AES y del modo CCM.

4.1. La plataforma de desarrollo

Para realizar el desarrollo de esta tesis, se utilizaron las siguientes herramientas:

- Las herramientas de desarrollo provistas por Xilinx para sus FGPAs, las cuales consisten en el *ISE Project Navigator* en su versión 6.3 para el sistema operativo Microsoft Windows. Así mismo se utilizó la herramienta de simulación *ModelSim Xilinx edition* versión 5.8, para el mismo sistema operativo.
- El proyecto se desarrolló bajo el lenguaje de programación VHDL (véase sección 4.2), utilizando el editor de texto de la herramienta descrita anteriormente.
- Se utilizaron las optimizaciones de la herramienta *Core Generator* para la construcción de las memorias BRAMs.
- El chip utilizado corresponde a la familia Spartan 3 (XC3S4000), y cubre las necesidades de desarrollo, tanto en área como en velocidad.

Es importante señalar que los resultados reportados en este trabajo, fueron obtenidos a través de la herramienta de síntesis incluida en el *Project Navigator* de Xilinx.

Dispositivo XC3S4000			
Compuertas lógicas	4M	Celdas lógicas equivalentes	62,208
CLBs	6912	RAM Distribuida	432 Kb
BRAMs	1728Kb	I/O disponibles	712

Cuadro 4.1: Descripción del FPGA utilizado

La tabla 4.1 muestra las características del chip seleccionado para llevar a cabo la implementación del modo de operación CCM.

4.2. El Lenguaje de programación VHDL

VHDL es un lenguaje para describir sistemas electrónicos digitales. Surgió a partir del programa de Circuitos Integrados de muy alta velocidad (*VHSIC* por sus siglas en inglés) del gobierno de los Estados Unidos, el cual inició en 1980. Durante dicho programa, se hizo notoria la necesidad de contar con un lenguaje estándar para describir la estructura y la función de los circuitos integrados (IC por siglas en inglés). Esto derivó en el desarrollo del Lenguaje de Descripción del Hardware VHSIC (VHDL por sus siglas en inglés), el cual fue posteriormente adoptado por la IEEE como un estándar [2].

El lenguaje VHDL está diseñado para satisfacer determinadas necesidades en el proceso de diseño [2]:

1. Permite la descripción de la estructura del diseño, lo que representa la manera en que está dividida en sub-diseños, y cómo esos sub-diseños son interconectados.

2. Permite que las funciones sean especificadas mediante el uso de estructuras de lenguajes de programación que son familiares a los desarrolladores.

3. Como resultado, es posible realizar la simulación del diseño antes de ser fabricado, por lo que los diseñadores son capaces de comparar las alternativas y probar que sean correctos sin el retraso y el costo del desarrollo de prototipos de hardware.

VHDL divide las entidades (componentes, circuitos o sistemas) en una parte visible o externa (nombre de la entidad y conexiones) y una parte oculta o interna (algoritmo de la entidad e implementación). Después de definir la interfaz externa, otras entidades pueden usarla cuando haya sido completamente desarrollada. Este concepto de vista externa e interna es central para una perspectiva de diseño de sistemas en VHDL. Una entidad es definida, en relación a otras, por sus conexiones y comportamiento. Se pueden explotar implementaciones externas (arquitecturas) de una de ellas sin cambiar el resto del diseño. Después de definir una entidad para un diseño, se puede reutilizar en otros diseños tanto como sea

necesario [38]. La figura 4.1 ilustra un modelo de hardware en VHDL.

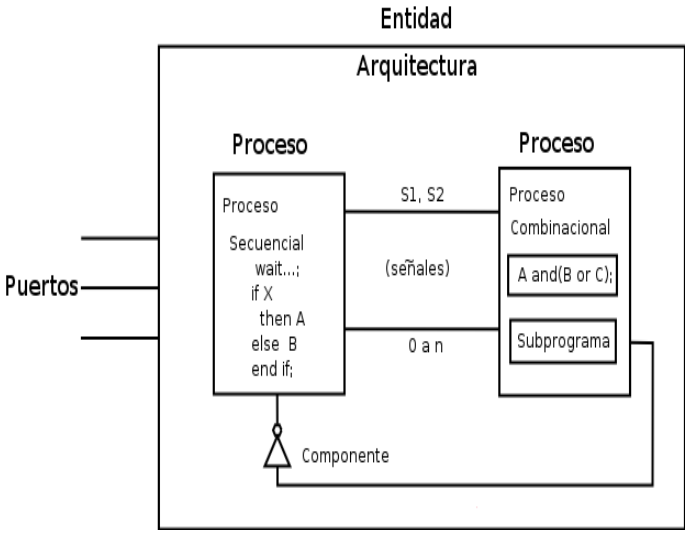


Figura 4.1: Un modelo VHDL de Hardware

Una entidad (diseño) VHDL tiene uno o más puertos de entrada, salida o entrada/salida, los cuales son conectados a sistemas vecinos. Está formada de *procesos* y *componentes* interconectados, todos operando concurrentemente. Cada entidad es definida por una *arquitectura* particular, que se compone de construcciones VHDL tal como operaciones aritméticas, asignaciones de señal, o declaraciones de instanciación de componentes.

En VHDL, los circuitos de modelo secuencial en procesos independientes (síncronos), usan flip-flops y latches, y los circuitos combinacionales (asíncronos) utilizan típicamente compuertas lógicas. Los procesos pueden definir y llamar (instanciar) *subprogramas* (diseños secundarios). Los procesos se comunican con cada uno de los otros procesos mediante *señales*. Una señal tiene una fuente (manejador), uno o más destinos (receptores), y un tipo definido por el usuario [38].

Las construcciones del lenguaje VHDL están divididas en tres categorías de acuerdo a su nivel de abstracción:

- **Comportamiento:** Es posible describir un circuito electrónico, generalmente digital, simplemente describiendo el comportamiento funcional o algorítmico del diseño, expresado en un proceso secuencial VHDL.
- **Flujo de datos:** Se ven a los datos como un flujo a través del diseño, de la entrada a la salida. Una operación es definida en términos de una colección de transformación de datos, expresado como una sentencia concurrente.
- **Estructural:** La visión más cercana al hardware; un modelo donde los componentes de un diseño se enumeran y se interconectan. Está expresada por instanciación de componentes.

Para el desarrollo de la implementación presentada en esta tesis, la programación de los componentes se llevó a cabo utilizando una construcción de tipo *estructural*.

4.3. Ciclo del diseño en FPGA

El ciclo general del diseño en un FPGA para este trabajo consistió en los siguientes pasos:

1. *Análisis de los algoritmos a implementar.* Esta fase fue crucial, pues se tuvieron que determinar los bloques básicos a implementar, con la finalidad de obtener los resultados esperados.

2. *Optimización de la arquitectura.* En este paso, se realizó el análisis de la secuencia del proceso, con la finalidad de encontrar posibles mejoras para el desempeño. Se analizaron algunas ideas encontradas en la literatura, así como las características específicas del FPGA seleccionado.

3. *Implementación de los bloques básicos en VHDL.* Una vez identificados los componentes básicos que constituyen el sistema completo, se implementaron dichos bloques partiendo desde los más sencillos hasta agruparlos para formar los más complejos. Toda la implementación se realizó mediante la descripción en el lenguaje VHDL.

4. *Simulación de cada uno de los componentes implementados.* Cuando se terminaba la implementación de cada bloque, se realizó la simulación del funcionamiento, con la finalidad de comprobar que la implementación generara los resultados esperados. La simulación se llevó a cabo con la herramienta *Model Sim*, con la que se analizaron las formas de onda obtenidas, así como diversas pruebas para validar el funcionamiento.

5. *Validación del diseño mediante la síntesis.* Cuando la simulación funcional era satisfactoria, se procedió a realizar la síntesis de dicho diseño, para de esta manera, validar que la implementación no contuviera errores de sintaxis.

4.4. Consideraciones para la implementación

Con la finalidad de obtener una buena implementación del modo CCM, que cumpliera con las necesidades del diseño, se hizo necesario tomar en cuenta algunas limitantes necesarias para acotar los alcances de esta tesis.

En primer lugar debe mencionarse, que la implementación aquí presentada es un prototipo de laboratorio, el cual tiene ciertas limitaciones inherentes a los recursos disponibles y a los alcances planteados.

La implementación llevada a cabo se basa en las siguientes consideraciones:

1. Debido a que el modo de operación CCM únicamente utiliza la primitiva de cifrado del AES, en este trabajo se reporta únicamente la implementación de dicha primitiva.

2. El tamaño máximo del mensaje es de 1024 bytes, lo cual fue basado en que una transmisión típica a través de una tarjeta inalámbrica comercial el tamaño máximo del buffer de salida es de 1048 bytes de acuerdo a las pruebas presentadas en la sección 3.7. Los 1024 bytes no incluyen los datos de autenticación adicionales.

3. La información opcional de autenticación considerada para esta implementación está basada en el tamaño del encabezado típico de una trama IEEE 802.11, por lo que su tamaño fijo es de 32 bytes (2 bloques de 128 bits).

4. Toda la información necesaria para el CCM, es proporcionada por una entidad externa, la cual suministra los datos en el momento en que son requeridos.

5. Para fines de prueba, tanto el mensaje como los datos de autenticación se fijan a un valor constante durante todo el tiempo de ejecución.

6. La validación de la ejecución se realizó con los programas hechos en lenguaje C y en Maple.

7. Como se explicó en la sección 3.2, el algoritmo de autenticación no puede ser paralelizado, lo que ocasiona que la implementación del cifrador por bloques no pueda ser hecha en forma de *pipeline* por lo que realizamos la implementación utilizando la aproximación iterativa.

4.5. Implementación del AES

Para llevar a cabo la implementación del AES, se siguió la metodología descrita en la sección 4.3, por lo que se diseñaron los componentes que integran al AES, conforme a la operación descrita en el capítulo 2. En las siguientes sub-secciones, se describe la manera en que se implementaron cada uno de los componentes.

4.5.1. Arquitectura General del AES

El AES en modo de cifrado está compuesto de tres componentes básicos: la expansión de llave, la unidad de control y los tres tipos de rondas del AES. La figura 4.2 ilustra la arquitectura general de la implementación del AES.

La expansión de llave se encarga de suministrar al cifrador, la llave de ronda correspondiente, de acuerdo al algoritmo presentado en la sección 2.2.2, mientras que la unidad de control es la responsable de sincronizar los procesos de generación de llave y cifrado, para así obtener un texto cifrado válido cuando el proceso termine, lo cual es indicado con la señal "cifraLista".

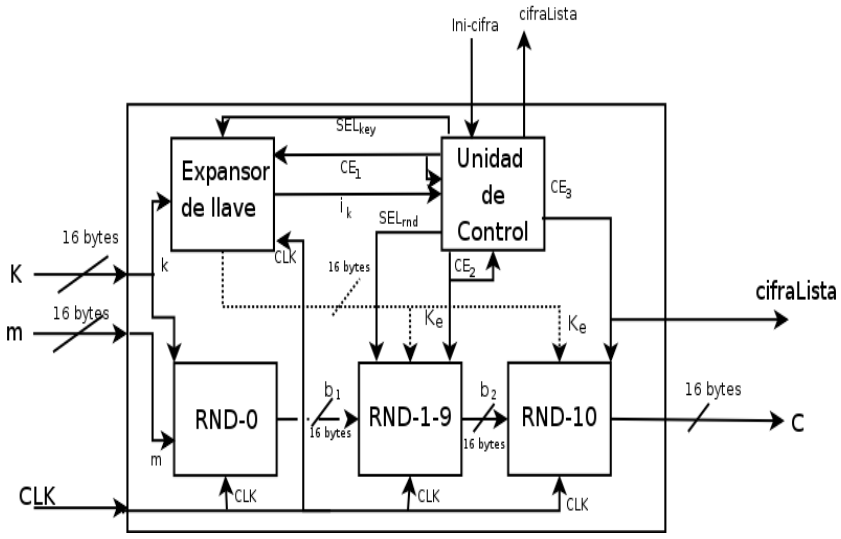


Figura 4.2: Arquitectura general de la implementación del AES

El resultado final de la implementación del AES, fue el contar con un componente denominado "AES", el cual consta de tres entradas (además de la del Reloj (CLK)), como se aprecia en la figura 4.2, el texto en claro, la llave secreta, y la señal de control que indica el inicio del proceso de cifrado (ini-Cifra en la figura), el cual debe estar en alto durante un pulso de reloj, para comenzar la cifra. AES proporciona a su salida el texto cifrado correspondiente. Este bloque es utilizado en la implementación del CCM como cifrador por bloques.

En las siguientes secciones se describe la implementación de cada uno de estos bloques.

4.5.2. Implementación de las Rondas del AES

Para realizar la implementación de las rondas de transformación del AES, fue necesario implementar cada uno de los pasos que las conforman. La figura 4.3 muestra la arquitectura diseñada para implementar las 10 rondas. A continuación describiremos cómo se implementó cada uno de los componentes.

Implementación de ARK

Como se mencionó en la sección 2.2.2, el *AddRoundKey* consiste únicamente en la "suma" de la matriz de estado con la matriz de la llave de ronda mediante una XOR. Para esta implementación, se considera que la entrada de los datos se da en formato de matrices.

Para implementar ARK fue necesario construir un bloque básico que realiza la operación XOR entre dos bytes. Dicho bloque se agrupó en bloques de cuatro bytes, para posteriormente implementar un bloque que realizara la operación de los 16 bytes de cada una de las matrices.

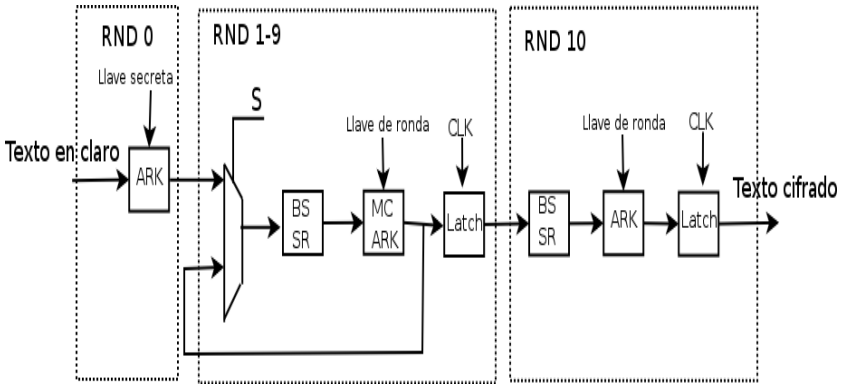


Figura 4.3: Implementación de las rondas de transformación

La figura 4.4 muestra el bloque resultante para la implementación del ARK. Es necesario destacar que esta implementación se utiliza para la ronda inicial y la ronda 10, puesto que las rondas 1-9 utilizan la optimización descrita en la sección 2.3.1.

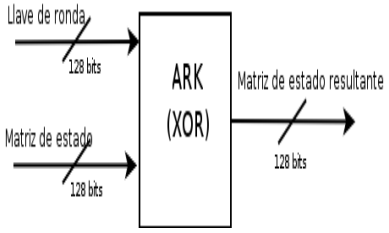


Figura 4.4: Implementación del Bloque ARK

Implementación de BS y SR

La implementación de *ByteSubstitution y ShiftRows* se hizo de manera simultánea, con la finalidad de incrementar la eficiencia del Sistema. Para llevar a cabo esta implementación se realizaron los siguientes pasos:

1. Se configuró una BRAM de 256 bytes, para almacenar la *SBOX*. Las BRAM fueron configuradas con la herramienta *CoreGenerator* incluida en las librerías de desarrollo de Xilinx.

2. Con la BRAM configurada, se construyó un componente que utiliza 4 de estas BRAMs, y que realiza la sustitución de 4 bytes al mismo tiempo. Éste mismo componente es reutilizado para la implementación del expansor de llaves.

3. Para realizar la sustitución de toda la matriz, se crean 4 componentes de los descritos en el paso anterior, para formar con ello un componente que lleva a cabo la sustitución de los 16 bytes de la matriz de estado.

4. Finalmente, debido a que *ShiftRows* consiste únicamente en un reorganización de los bytes, su implementación fue muy sencilla, puesto que lo único que se realizó fue el redireccionamiento de las salidas de *SBOX* al orden especificado por los offsets. Lo anterior, permite obtener un mejor rendimiento al evitar los retrasos por interconexión.

La figura 4.5 ilustra gráficamente la implementación de los pasos BS y SR. En ella se muestra la manera en que se realiza el redireccionamiento de los bytes sustituidos en la *SBOX*.

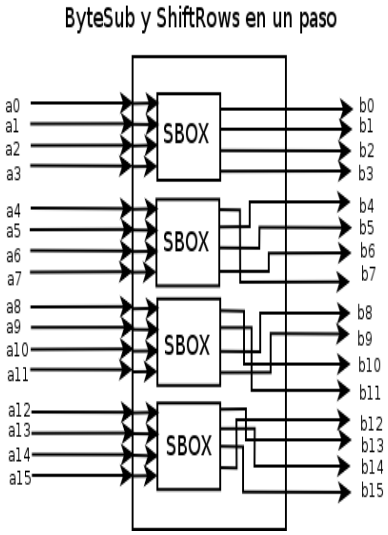


Figura 4.5: Implementación de SubBytes y ShiftRows en un solo paso

En la figura a_i representa el byte i de la matriz de estado de entrada, mientras que b_i es el byte i de la matriz de estado resultante.

Implementación de MC y ARK

Como se mencionó en la sección 2.3.1, la implementación de *Mix Columns y AddRoundKey* puede hacerse en un solo paso, con la finalidad de reducir el tiempo de cómputo y así obtener un incremento en la eficiencia del sistema.

Para la implementación de estos pasos en FPGA, se utiliza la ecuación 2.6, la cual consta de compuertas XOR, y tablas de consulta implementadas mediante BRAMs.

La tabla correspondiente a *xtime*(), requirió 16 BRAMs, de 256 bytes, acomodados en una matriz de 16×16. Al igual que las BRAMs utilizadas anteriormente, fueron construidas utilizando la herramienta *CoreGenerator*.

La figura 4.6, muestra el circuito que implementa estos dos pasos. En la figura, *xtime*() es representado por un cuadro que simboliza la consulta a la memoria RAM, para obtener el dato buscado.

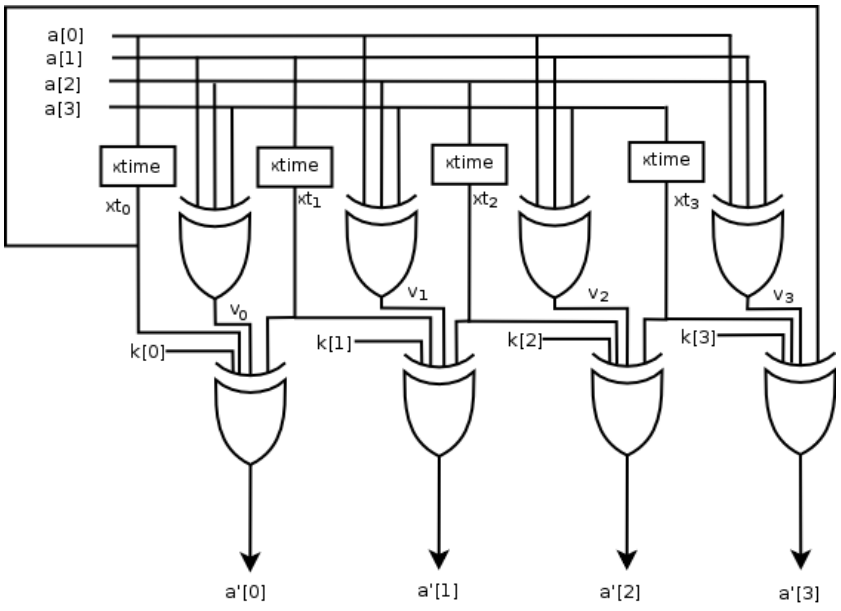


Figura 4.6: Circuito de la implementación de ARK y MC

4.5.3. Implementación de la expansión de llave

El algoritmo de expansión de llave descrito en la sección 2.2.2, se implementó conforme se ilustra en la figura 4.7, en la cual se muestran los diferentes componentes que integran la implementación.

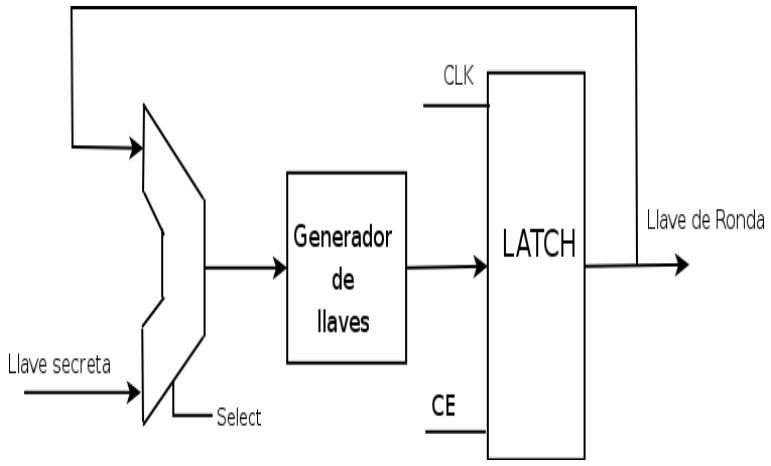


Figura 4.7: Arquitectura General de la expansión de llave

Como se ilustra en la figura, la implementación de la expansión de llave, se realiza de manera iterativa, es decir, que el proceso *"Generador de llave"* se repite 10 veces para suministrar las llaves de ronda correspondientes. El multiplexor inicial, realiza la selección de la llave a utilizar en el generador de llaves. En un primer paso, se usa la llave secreta, mien-

tras que en los pasos subsecuentes, se retroalimentan las llaves calculadas en el paso anterior.

El generador de llaves se implementó, mediante la utilización de compuertas XOR, como se expresa en la ecuación 2.7 de la sección 2.3.2. La implementación se muestra en la figura 4.8.

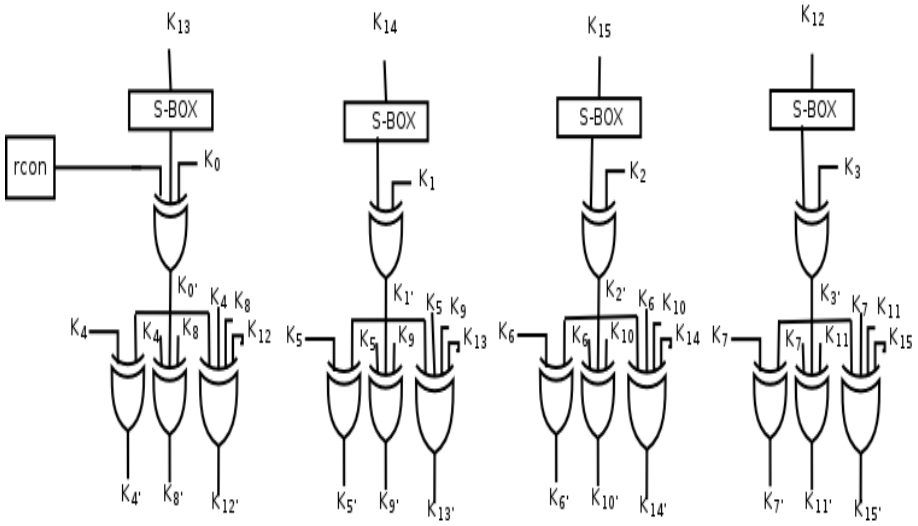


Figura 4.8: Circuito del Generador de llaves

En la figura puede observarse la utilización de las funciones *SBOX* y *rcon*, las cuales consisten en tablas de consulta implementadas mediante una BRAM incluida en el FPGA elegido para el desarrollo. Estas funciones toman un byte de entrada como dirección de la memoria, y a la salida proporcionan el valor almacenado en dicha dirección. Este modo de implementación permite obtener un buen rendimiento del diseño. Para el caso de *SBOX* se reutilizó el componente que sustituye 4 bytes de manera paralela, el cual se desarrolló para la implementación del paso *ByteSubstitution* descrito anteriormente.

La tabla para *SBOX* está constituida por 256 bytes, organizados en una matriz de 16×16 bytes. Por otro lado, la tabla correspondiente para *rcon* es una matriz de 4×8 bytes, la cual almacena el cálculo de las constantes de ronda calculada como:

$$r_{con}(i) = 02^{(i-4)/4} \text{ en } GF(2^8) \tag{4.1}$$

Ambas tablas se utilizan para realizar el cálculo de la expansión de llave como se ilustró en la figura 4.8.

El *latch* al final del componente se utiliza principalmente para almacenar los resultados parciales, y de este modo simplificar la sincronización entre los componentes restantes del AES.

4.6. Implementación del CCM

En esta sección se describe el diseño realizado para el modo de operación CCM. Se da una descripción detallada de la arquitectura general propuesta, para posteriormente describir la implementación de cada uno de los bloques que conforman la arquitectura.

4.6.1. Las parámetros e información de entrada y salida del CCM

Con la finalidad de hacer más fácil la lectura de esta sección, presentamos un resumen de los datos de entrada y salida del modo CCM, así como los parámetros utilizados. En la tabla 4.2 se encuentra el nombre y descripción de dicha información correspondientes a la implementación que se presenta en este capítulo.

4.6.2. Arquitectura General del Sistema

La figura 4.9 muestra la arquitectura general del sistema, en ella se ilustra la interconexión que existe entre los componentes del modo CCM.

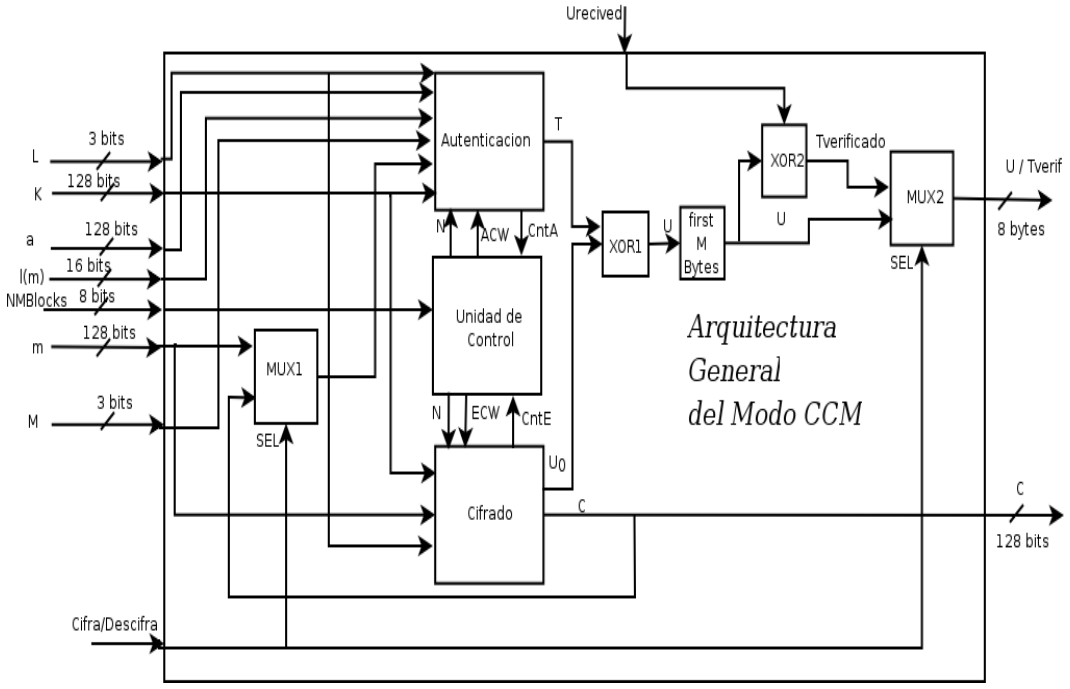


Figura 4.9: Arquitectura General de la implementación del CCM

Como se muestra en la figura 4.9, el diseño consta de los módulos de autenticación y cifrado, así como la unidad de control general, la cual se encarga de sincronizar los procesos a través de las palabras de control ACW y ECW. Así mismo, se pueden apreciar algunos

Nombre	Descripción	Tamaño del campo	Codificación del campo
	Parámetros		
M	Número de bytes del campo de autenticación	3 bits	$(M-2)/2$
L	Número de bytes en el campo de longitud de mensaje	3 bits	$L-1$
	Entradas		
K	Llave secreta para el AES	16 bytes	Cadena de bits
N	Número aleatorio (Nonces)	$15-L$ bytes	Cadena de bits
m	Mensaje a cifrar y enviar	Máximo 1024 bytes	Cadena de bits
a	Información adicional a autenticar (no se cifra)	32 bytes	Cadena de bits
	Auxiliar		
T	Campo de autenticación sin cifrar	M bytes	Cadena de bits
	Salidas		
U	Campo de autenticación cifrado	M bytes	Cadena de bits
C_i	Bloque i-ésimo del texto cifrado	16 bytes	Cadena de bits

Cuadro 4.2: Descripción de los parámetros y entradas del CCM

componentes externos, los cuales realizan el descifrado y la verificación conforme se describe en la sección 4.6.6. En las siguientes subsecciones se explican a detalle los diferentes componentes.

4.6.3. Implementación de la autenticación en CCM

Para realizar la implementación de la autenticación, se diseñó la arquitectura ilustrada en la figura 4.10, la cual muestra los componentes que conforman el bloque de autenticación, los cuales son: El generador de bloques, CBC-MAC y la unidad de control para la autenticación. A continuación explicaremos la manera en que cada uno de los componentes fueron implementados.

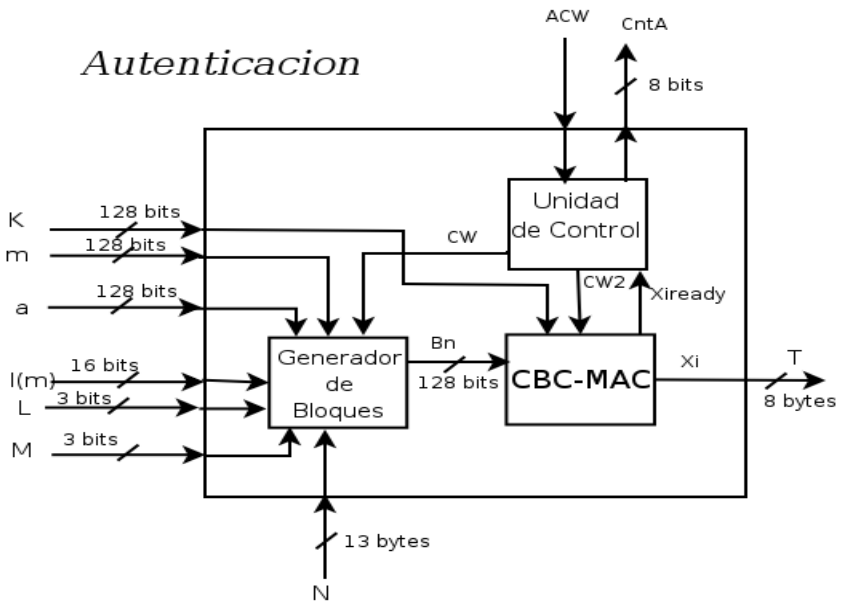


Figura 4.10: Arquitectura del módulo de autenticación

Implementación del Generador de bloques

El Generador de bloques es el encargado de construir los bloques $B_0, B_1, ..., B_n$ como se especificó en la sección 3.2. Estos bloques se generan con la información proporcionada por el usuario, así como con los parámetros del sistema.

La implementación se realizó mediante un proceso codificado en VHDL, el cual recibe una palabra de control (2 bits) proveniente de la unidad de control de la autenticación. Dicha palabra de control se describe en la tabla 4.3. El algoritmo 10 muestra el algoritmo implementado para el generador de bloques.

Valor	Significado
00	Genera B_0 sin datos de autenticación adicionales (Adata=0)
01	Genera B_0 con datos de autenticación adicionales (Adata=1)
10	Genera B_i a partir del mensaje a enviar (m)
11	Genera B_i con los datos de autenticación adicionales (a)

Cuadro 4.3: Descripción de la palabra de control del Generador de bloques

Algoritmo 10 Algoritmo del Generador de bloques para la autenticación

Requiere: $CW, m, a, l(m), nonce, M, L, contador$

if Ocurrió un cambio en CW y contador **then**
  if $CW = 00$ o $CW = 01$ **then**
    if $CW = 00$ **then**
      $Bloque \leftarrow B_0$ con $Adata = 0$
    else
      $Bloque \leftarrow B_0$ con $Adata = 1$
    end if
  else if $CW = 10$ **then**
    $Bloque \leftarrow B_i$ en base al mensaje m
  else if $CW = 11$ **then**
    $Bloque \leftarrow B_i$ en base a los datos de autenticación a.
  end if
end if
$BlockGen \leftarrow Bloque$

Las señales *CW* y *contador*, provienen de la unidad de control. La primera sirve para especificar el tipo de bloque que se generará, mientras que la segunda indica el número de bloque que se está generando. Debido a las consideraciones hechas para el diseño (véase la sección 4.4), el número de bloques máximo a generar para la autenticación son 67, de los cuales 64 corresponden al mensaje a transmitir (1024 bytes en bloques de 16 bytes cada uno), 2 para los datos a autenticar (a) y 1 para B_0.

Una vez que el bloque B_i es generado, se direcciona a las salidas de la entidad, con la finalidad de ser proporcionado a la siguiente fase, el CBC-MAC, que explicaremos a continuación.

El CBC-MAC

Como se mencionó en la sección 3.2, el CBC-MAC, es el componente básico de la autenticación y es quien lleva a cabo la autenticación de la información proporcionada por el Generador de bloques explicado en los párrafos anteriores. La figura 4.11 muestra la arquitectura diseñada para la implementación del CBC-MAC, la cual consta de 4 componentes: Una operación XOR, un multiplexor, el cifrador AES y dos latches.

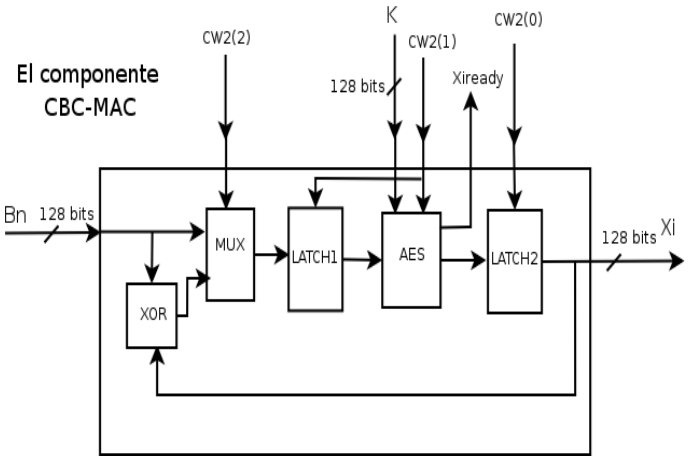


Figura 4.11: Arquitectura del CBC-MAC

A continuación explicaremos la implementación de cada componente:

- **La operación XOR:** Su función es la de realizar la operación lógica XOR ($\oplus$), entre el bloque de entrada y el bloque recién procesado, con la finalidad de cumplir con la ecuación $B_i \oplus X_i$ (véase ecuaciones 3.1 y 3.2). Debido a que esta operación no utiliza una cantidad de recursos considerable, la operación se realiza con cada bloque, siendo el multiplexor quien decide si al AES ingresa B_i o $B_i \oplus X_i$. Para implementar este bloque se utilizó el mismo componente que se creó para el ARK del AES, puesto que ambos trabajan con matrices de 16 bytes.

- **Multiplexor:** La finalidad del multiplexor es controlar la información que ingresa al AES, es decir, si se alimentará el bloque B_0 (primer bloque construido por el Generador de bloques) o bien, se procesarán los bloques correspondientes a la retroalimentación propia del CBC-MAC. El multiplexor es controlado por el bit más significativo de la palabra de control recibida para el CBC-MAC. Si es '0' se utiliza la información correspondiente al generador de bloques (sin la XOR), en caso contrario, se utiliza el resultado de la operación lógica XOR.

- **Latch1:** El propósito de este latch es ayudar a la sincronización de los procesos. Debido a que el retraso en las conexiones generaba que el AES no obtuviera los datos adecuados, se hizo necesario almacenar la información parcial antes de ser proporcionada al AES. Es controlado con el mismo bit que el AES.

- **AES:** La función del AES, es cifrar la información que es provista a su entrada, con la llave secreta proporcionada por el usuario. El segundo bit de la palabra de control le indica el momento en que debe comenzar el proceso de cifrado. A su salida se obtiene el texto cifrado $E_k(B_0)$ o $E_k(X_i \oplus B_i)$, según sea el caso. La señal *cifraLista* generada por el AES, se convierte en *XiReady*, la cual es utilizada por la unidad de control de la autenticación para determinar que el bloque B_i ha sido procesado y puede iniciar el procesamiento de B_{i+1}.

- **Latch2:** Este último latch tiene como finalidad primordial, el almacenar los resultados parciales, que serán proporcionados a los bloques restantes que componen la arquitectura general de la autenticación y con ello facilitar la sincronización de los procedimientos. Es controlado con el bit menos significativo de la palabra de control, el cual permite la escritura en este registro.

Implementación de la unidad de control para la autenticación

La unidad de control para la autenticación es la encargada de sincronizar los procesos que intervienen en la autenticación de la información. Recibe la palabra de control de la unidad de control General, la cual consta de un bit. Si es '0' el proceso está deshabilitado y no se realiza operación alguna, en caso contrario, se inicia el proceso de autenticación y se continúa hasta que la palabra de control sea '0'.

El algoritmo 11, muestra el algoritmo que se utilizó para implementar la unidad de control.

La unidad de control genera las palabras de control necesarias para coordinar el proceso de autenticación. Los valores y sus significados se muestran en la tabla 4.4.

La UC para la autenticación, envía el número de bloque que está siendo procesado a la unidad de control General, la cual, sincroniza el proceso de cifrado con el de autenticación.

Algoritmo 11 Algoritmo de la unidad de control para la autenticación

Requiere: $CW, xiReady, estado$

1: $estado = inicial$
2: **for** $i = 0$ to $i = n$ **do**
3: **if** $xiReady =' 1'$ y $CW =' 1'$ **then**
4: **if** $estado = inicial$ **then**
5: $CWOut \leftarrow GenB0$ {Generar B_0}
6: $estado = iniciado$
7: **else if** $i < 3$ **then**
8: $CWOut \leftarrow GenB1y2$ {Generar B_1 y B_2 con a}
9: **else**
10: $CWOut \leftarrow GenBk$ {Generar B_k con m}
11: **end if**
12: **end if**
13: **end for**

Valor	Significado
Bits de control para el Generador de bloques	
$00XXX$	Genera B_0 sin datos de autenticación adicionales (Adata=0)
$01XXX$	Genera B_0 con datos de autenticación adicionales (Adata=1)
$10XXX$	Genera B_i a partir del mensaje a enviar (m)
$11XXX$	Genera B_i con los datos de autenticación adicionales (a)
Bits de control para el CBC-MAC	
$XX1XX$	control del multiplexor, usar B_i
$XX0XX$	control del multiplexor, usar $B_i \oplus X_i$
$XXX0X$	control del latch1 y AES, no cifrar y desactivar latch1
$XXX1X$	control del latch1 y AES, cifrar y activar latch1
$XXXX0$	control del latch2, desactivarlo.
$XXXX1$	control del latch2, activarlo

Cuadro 4.4: Descripción de la palabra de control de la UC de la autenticación

4.6.4. Implementación del cifrado en CCM

La figura 4.12 muestra la arquitectura diseñada para la implementación del cifrado en el modo CCM. En ella se pueden apreciar los componentes que integran el diseño propuesto: un generador de bloques, el cifrador CTR, y una unidad de control. A continuación se explica la implementación de cada uno de estos componentes.

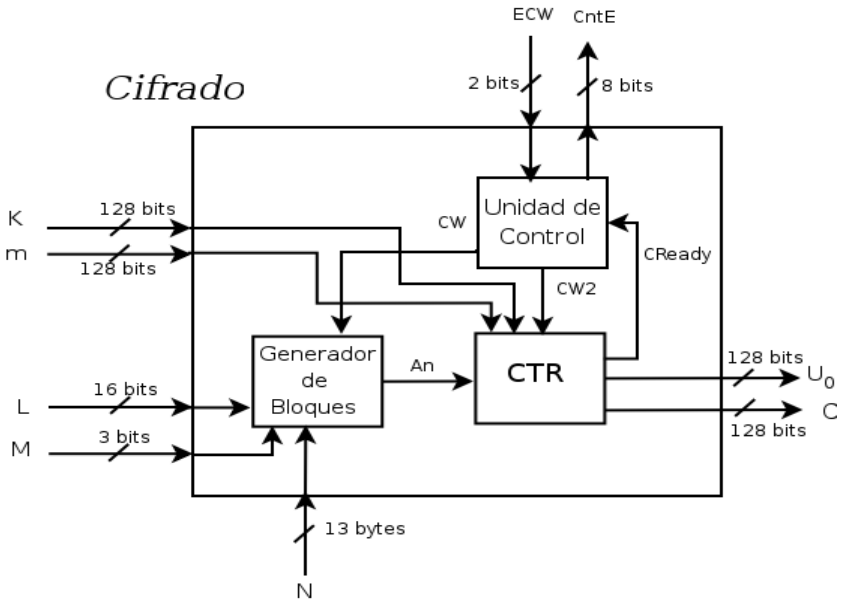


Figura 4.12: Arquitectura del cifrado en CCM

El generador de bloques para el cifrado

El generador de bloques es el encargado de proporcionar el conjunto de bloques $A_0, A_1, A_2, ...$ los cuales son formados de acuerdo a la figura 3.5, que se encuentra en la sección 3.3.

Los bloques se generan a partir de la función de conteo, la cual es proporcionada por la unidad de control del cifrado. En esta implementación el conteo se inicia en 1, para generar así el bloque A_1. La implementación del Generador de bloques, es sencilla, debido a que únicamente se cambia el valor del contador. El algoritmo 12 muestra el procedimiento que se implementó para este componente.

La generación de los bloques se implementó utilizando un proceso, el cual responde a los cambios en el contador, para construir el bloque A_i correspondiente. La salida del generador de bloques es procesada por el bloque CTR.

El generador de bloques es habilitado a través de una palabra de control proveniente de la unidad de control del cifrado. Si es '1', el proceso puede llevarse a cabo, de lo contrario no se realiza ninguna operación.

Algoritmo 12 Algoritmo del Generador de bloques para el cifrado

Requiere: $nonce, L, contador, CW$

1: **if** $CW = 1$ **then**
2: $i \leftarrow contador$
3: **if** $i \geq 0$ **then**
4: Construye A_i
5: **end if**
6: **end if**
7: $Salida \leftarrow A_i$

Cifrado con el modo CTR

Una vez que el bloque A_i es generado, la información es cifrada mediante el modo CTR. La arquitectura diseñada para la implementación de este modo de operación se ilustra en la figura 4.13.

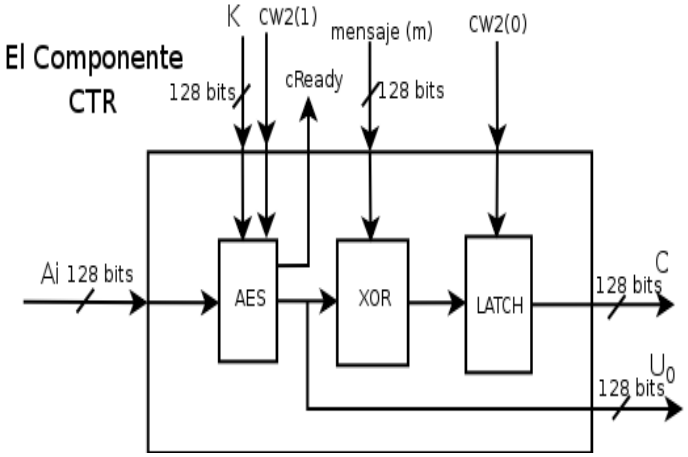


Figura 4.13: Arquitectura del módulo CTR

Como puede apreciarse en la figura 4.13, el bloque A_i es cifrado con AES para obtener S_i, con el cual se cifra el mensaje m, a través de la operación XOR como se mostró en el algoritmo 7 (véase la sección 3.3).

La salida U_0 que se muestra en la figura, corresponde a S_0, el cual conforme a la ecuación 3.5 (véase la sección 3.3) es utilizado para cifrar el parámetro de autenticación T, por lo que no pasa por la fase de la operación XOR con el mensaje, ni el resultado es almacenado en el LATCH que se encuentra al final del circuito.

Valor	Significado
0XX	Generador de bloques deshabilitado
1XX	Generador de bloques habilitado
X1X	Iniciar cifrado con AES
X0X	Detener cifrado con AES
XX0	latch Deshabilitado
XX1	latch Habilitado

Cuadro 4.5: Descripción de la palabra de control del cifrado

El módulo CTR recibe de la unidad de control para el cifrado una palabra de control de dos bits, la cual es utilizada para controlar el AES (el bit más significativo) y para activar o desactivar el latch (el bit menos significativo).

Cuando un bloque ha sido cifrado, la señal de aviso del AES, se convierte en la señal "cReady", indicándole a la unidad de control del cifrado que ha concluido el cifrado del bloque de información y que puede iniciarse el cifrado de uno nuevo.

La unidad de control para el cifrado

La Unidad de control para el cifrado, se implementó mediante un proceso que proporciona el índice del bloque a procesar al generador de bloques, a través de la función de conteo definida por el usuario. Para el desarrollo de esta tesis, la función de conteo consiste en incrementar en una unidad el índice del bloque a procesar.

La unidad de control recibe una palabra de control de la Unidad de control General, la cual consta de dos bits, e indica el procedimiento que tiene que llevarse a cabo. El significado de cada bit se explica en la sección 4.6.5 en la tabla 4.6.

Así mismo esta unidad de control genera una palabra de control de 3 bits, para controlar los diferentes componentes. El significado de cada bit se explica en la tabla 4.5.

Como ha sido mencionado anteriormente, la unidad de control es la encargada de proporcionar el índice del bloque a generar, por lo que dentro de la implementación se realiza el conteo de los bloques, incrementando en uno el contador cada vez que se ha terminado de procesar el bloque anterior. El algoritmo 13 muestra el procedimiento que se sigue en la unidad de control.

La señal de RESET es general y sirve para reinicializar todo el sistema a un estado inicial. Como puede apreciarse en el algoritmo, la palabra de control de la unidad de control general

Algoritmo 13 Algoritmo de la unidad de control para el cifrado

Requiere: $CWGen, RESET, CE, cReady$

1: **if** $RESET = 1$ **then**
2: $contadorAux \leftarrow 0$
3: $CW \leftarrow$ Desactivar procesos
4: **else if** $CE = 1$ **then**
5: $CW \leftarrow$ Activar procesos
6: **if** $CWGen = 1$ y $cReady = 1$ **then**
7: $contadorAux+ = 1$
8: **else**
9: $contadorAux = 0$
10: **end if**
11: **end if**
12: $ContadorSalida \leftarrow contadorAux$

($CWGen$) determina qué valor del contador se utilizará, si se utiliza el incremento en una unidad del contador o bien el valor inicial de la función de conteo, que en la implementación es cero. La señal CE indica si el proceso está habilitado, o se encuentra en modo de espera, lo cual es útil para coordinar los procesos de autenticación y cifrado como se explica en la sección 4.6.5.

Para la unidad de control "cReady=1" indica que el bloque de información ha sido cifrado por completo, por lo que puede comenzar el procesamiento del siguiente bloque.

4.6.5. La unidad de control General

Para que los procesos de autenticación y verificacion procesen los datos de forma adecuada, es necesario que exista un módulo capaz de sincronizar el intercambio de información entre ellos, por lo cual se realizó la implementación de una unidad de control General, con la que se logra la interacción de los componentes descritos anteriormente.

La unidad de control General utiliza una palabra de control de 3 bits para coordinar la autenticación y el cifrado. La tabla 4.6 muestra el significado de cada uno de los bits de la palabra de control.

La unidad de control se diseñó con la finalidad de obtener una implementación eficiente del modo de operación CCM, por lo que se llevó a cabo un procedimiento que permite realizar la etapa de autenticación al mismo tiempo que el cifrado, lo que brinda un mejor

Valor	Significado
1XX	Utilizar el valor incrementado de la función de conteo
0XX	Utilizar el valor inicial de la función de conteo
X0X	Proceso de cifrado deshabilitado
X1X	Proceso de cifrado Habilitado
XX0	Deshabilitar el proceso de autenticación
XX1	Habilitar el proceso de autenticación

Cuadro 4.6: Descripción de la palabra de control General

desempeño del sistema.

La figura 4.14 muestra el diagrama de tiempos del proceso de autenticación y cifrado. Como se ilustra en dicha figura, el procesamiento de la información consiste en iniciar la fase de autenticación con el bloque B_0, seguido por el procesamiento del bloque B_1 el cual se forma con el primer bloque de los datos a autenticar (a). Durante ese tiempo el cifrado permanece en estado de espera. Cada unidad de tiempo representa 12 ciclos de reloj.

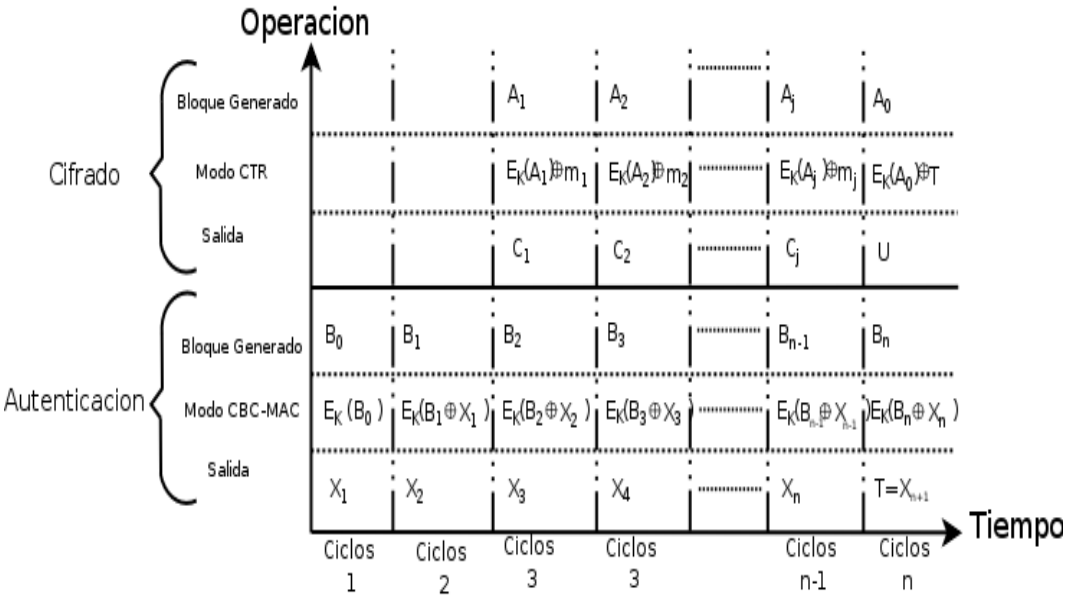


Figura 4.14: Diagrama de tiempos para la autenticación y cifrado

Como se especificó en la sección 4.4, el número de bloques con datos de autenticación opcionales se fijó en 2, por lo que cuando se termina con el procesamiento de B_1, el proceso

de autenticación inicia con B_2 (el segundo bloque conformado por a). En este punto el cifrado comienza con el procesamiento del primer bloque del mensaje m a cifrar. Cuando ambos terminan, la autenticación procesa el primer bloque del mensaje m, mientras que el cifrado inicia ahora con el segundo bloque del mensaje m.

Ese desfasamiento se conserva hasta que el cifrado termina con el procesamiento de m_n, y la autenticación con m_{n-1}. Finalmente, el cifrado genera el bloque S_0, mientras la autenticación procesa el bloque correspondiente a m_n para calcular así el parámetro de autenticación T, el cual se cifrará con la compuerta XOR1 de la figura 4.9, para ser truncado a M bytes con la función "firstMBytes" para tener como resultado el parámetro de autenticación cifrado U, que será enviado al receptor junto con los bloques de la cifra calculados con el módulo de cifrado.

La unidad de control general realiza esta sincronización a través del conteo de los bloques que procesa tanto la autenticación como el cifrado.

Mediante el procedimiento descrito en los párrafos anteriores, la implementación propuesta permite que los procesos de autenticación y cifrado sean realizados casi al mismo tiempo, con lo que el desempeño del sistema mejora considerablemente. Los resultados obtenidos de esta implementación son discutidos en el capítulo 5.

4.6.6. Implementación del descifrado y verificacion

Para llevar a cabo el descifrado y la verificación, se diseñó el Hardware extra que se aprecia en la figura 4.9, el cual consta de los multiplexores 1 y 2, las compuertas XOR 1 y 2, así como el módulo "firstMBytes".

El diseño propuesto permite controlar las operaciones de descifrado y verificacion de manera independiente a la implementación descrita en las secciones anteriores, debido a que como se explicó en las secciones 3.4 y 3.5, los procesos de descifrado y verificación son los inversos de cifrado y autenticación respectivamente.

Por lo anterior, ambos procesos se pueden realizar únicamente con la modificación de que el descifrado utiliza la cifra C enviada por el emisor, en lugar del mensaje m, mientras que la verificación utiliza el mensaje descifrado.

El multiplexor 1, tiene como objetivo realizar la selección de la entrada correspondiente para la autenticación/verificacion, por lo que si la entrada de selección es 0, se utilizará el mensaje m a cifrar, por lo tanto se realiza la autenticación de la información, lo que es realizado por el emisor. En caso contrario, se utiliza la información proporcionada por el módulo de cifrado, el cual estará realizando el descifrado de la información recibida.

La verificacion se lleva a cabo exactamente igual a la autenticación con la diferencia que al final del procesamiento de la información, mediante la compuerta XOR2, se valida la

información recibida, el multiplexor 2, brinda a su salida, el valor de U calculado para ser enviado o bien el valor de verificación, el cual tiene que ser 0, para comprobar que el mensaje recibido no sufrió modificaciones durante la transmisión.

Mediante este diseño, es posible utilizar el hardware diseñado para la autenticación y el cifrado de forma transparente, pues el procedimiento es el mismo, por lo que la gráfica de tiempos es similar, lo cual se ilustra en la figura 4.15. Al igual que en la de autenticación y cifrado, cada unidad de tiempo representa 12 ciclos de reloj.

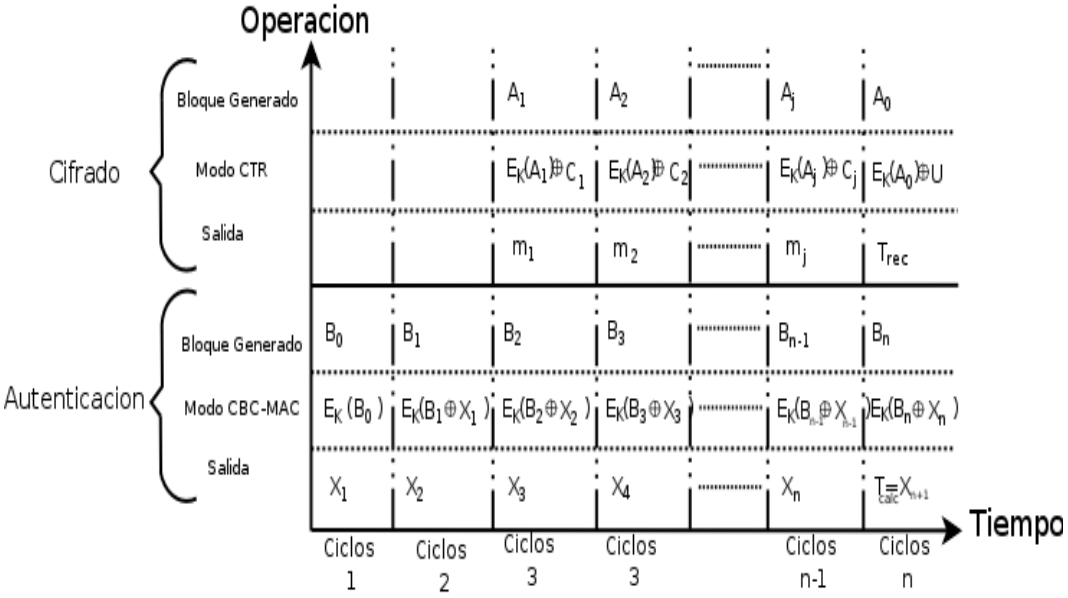


Figura 4.15: Diagrama de tiempos para el descifrado y la verificacion

Capítulo 5

Resultados Comparativos

En este capítulo se presentan los resultados obtenidos en el desarrollo de esta tesis. Iniciamos el capítulo con una descripción de las métricas utilizadas para FPGAs, para posteriormente mostrar los resultados obtenidos. Presentamos primeramente, un análisis de los resultados que se presentaron en la sección 3.7.3.

En segundo lugar se reportan los resultados correspondientes a la implementación del AES. Finalmente se muestran los resultados obtenidos en la implementación del modo de operación CCM.

5.1. Medidas de rendimiento en FPGA

Con la finalidad de determinar la calidad del trabajo realizado, es necesario definir los criterios de medición, los cuales fueron divididos básicamente en dos aspectos: El área utilizada y la velocidad de procesamiento (Throughput).

En [28] se definen como:

- **Área:** Es el espacio ocupado por el diseño, el cual es expresado en términos de slices de CLBs. Algunos FPGAs tienen otros recursos como BRAMs, Multiplicadores, etc. Si estos recursos son usados es necesario también reportarlos puesto que son recursos dedicados y no ocupan CLBs. Si no son mencionados no puede justificarse el ahorro de espacio en un diseño. El número de entradas y salidas también son mencionadas en algunos casos. Un diseño es económico si ocupa poco espacio.
- **Throughput:** Es un factor muy importante para medir el rendimiento en tiempo de un diseño. El throughput de un diseño es obtenido mediante la multiplicación de la frecuencia permitida para el reloj por el número de bits procesados por ciclo. Para los algoritmos criptográficos, el throughput es definido como:

$$Throughput = \frac{\text{Frecuencia permitida } \times \text{ Número de Bits}}{\text{Número de rondas}}(bits/s)$$

Número de Veces	Sin Cifrar	WEP 40 bits	WEP 104	AES-CCM
10,000	4.195 s	4.795 s	4.815 s	5.38 s
20,000	8.47 s	9.65 s	9.53 s	10.555 s
30,000	12.665 s	14.46 s	14.33 s	15.985 s
40,000	16.88 s	19.13 s	19.29 s	21.335 s
50,000	21.135 s	23.885 s	23.875 s	26.51 s
100,000	42.355s	47.675 s	47.88 s	53.425s
200,000	84.775 s	95.875 s	95.845 s	106.38 s
300,000	126.91 s	143.855 s	143.845 s	159.725 s
400,000	168.98 s	191.52 s	191.93 s	212.805
500,000	211.34 s	239.245 s	239.4 s	265.894 s

Cuadro 5.1: Tabla de resultados obtenidos

Mientras más elevado sea el Throughput de un diseño, mayor es su eficiencia.

5.1.1. Herramientas para realizar la medición

Para el desarrollo de esta tesis, los resultados fueron obtenidos a través de la herramienta de síntesis integrada a la herramienta *ISE Project Navigator*, la simulación se realizó con el software *Model Sim Xilinx Edition II v5.8c.*

Los resultados reportados en este capítulo, fueron calculados utilizando los datos de la herramienta de síntesis, así como la gráfica de simulación.

5.2. Análisis de las pruebas con tarjetas inalámbricas

En la sección 3.7 se presentan las pruebas realizadas con las tarjetas inalámbricas 3COM 3CRDW696. En esta sección hacemos un análisis de los resultados obtenidos y brindamos una estimación del tiempo de procesamiento necesario para llevar a cabo las operaciones necesarias para el modo de operación CCM.

En la tabla 5.1 se presentan los resultados reportados en la sección 3.7.3 para la transmisión sin utilizar esquemas de cifrado, con WEP de 40 y 104 bits y para el cifrado con el CCM, que es el esquema de interés para la tesis.

Como puede observarse, la transmisión de la información es más rápida si no se utiliza esquema de cifrado alguno. También puede notarse que el modo CCM es el más lento de los esquemas de cifrado, lo cual se debe en buena medida a que la operación de cifrar con AES, es significativamente más lenta que hacerlo con el RC4. Además, las operaciones del modo CCM son realizadas a través del software implementado en el driver *Host AP - Linux driver for Prism2/2.5/3* (véase el apéndice C).

Si consideramos que el tiempo de transmisión sin cifra es la referencia del tiempo que se tarda en realidad para enviar la información, podemos hacer una aproximación teórica, del rendimiento que tiene cada esquema de cifrado, para comparar la velocidad de las operaciones en cada uno.

Supongamos que podemos calcular la velocidad de procesamiento mediante las siguientes ecuaciones:

Sean:

t_{tsc} el tiempo de transmisión sin cifra
t_{tcc} el tiempo de transmisión con cifra
t_{proc} el tiempo de procesamiento del esquema
n_{veces} el número de veces que se envió el buffer
$v_{cifrado}$ la velocidad de cifrado en *bps*

$$t_{proc} = t_{tcc} - t_{tsc} \tag{5.1}$$

$$v_{cifrado} = \frac{n_{veces} \times 1024 \times 8}{t_{proc}} \tag{5.2}$$

Por lo tanto, tomando como ejemplo el primer renglón de la tabla 5.1, para el WEP con 40 bits, usando las ecuaciones 5.1 y 5.2 los cálculos quedarían de la siguiente forma:

$$t_{proc} = 4{,}795 - 4{,}195 = 0{,}6 \tag{5.3}$$

$$v_{cifrado} = \frac{10000 \times 1024 \times 8}{0{,}6} = 136{,}53 \tag{5.4}$$

Si realizamos los cálculos anteriores para los datos reportados en la tabla 5.1, podemos generar la tabla 5.2. Las operaciones se realizaron para el WEP de 40 bits y para el CCM. Se excluyó al WEP de 104 bits, pues su rendimiento es muy similar a la versión de 40 bits.

Como puede verse en la tabla 5.2, la velocidad de procesamiento del CCM es casi del 50 % con respecto al WEP. Lo anterior se debe en gran parte, como se ha mencionado anteriormente, a que las operaciones correspondientes del CCM, son realizadas a través del software implementado en el controlador utilizado, mientras que el WEP se realiza a través

Número de Veces	WEP 40 bits (Mbps)	CCM (Mbps)
10,000	136.53	69.13
20,000	138.84	78.58
30,000	136.9	74.02
40,000	145.63	73.55
50,000	148.94	76.20
100,000	153.98	74.00
200,000	147.6	75.76
300,000	145.03	74.89
400,000	145.37	74.77
500,000	146.78	75.05

Cuadro 5.2: Velocidad de procesamiento aproximada para WEP y CCM

del Hardware incluido en la tarjeta inalámbrica.

Otro de los aspectos importantes a considerar es que las operaciones de Cifrado y Autenticación con CCM, requieren cifrar información con AES, el cual es más lento que el RC4. Debido a esto, es importante contar con una implementación en Hardware, que sea capaz de llevar a cabo las operaciones correspondientes en menor tiempo.

En las siguientes secciones se brindan los resultados obtenidos en la implementación del AES y el CCM.

5.3. Resultados de la implementación de AES

La tabla 5.3 muestra los datos obtenidos con la herramienta de síntesis, para nuestra implementación del AES.

Como puede observarse en la tabla 5.3, el diseño presentado en esta tesis, es económico en términos de área, sin embargo, utiliza una cantidad considerable de BRAMs, las cuales permiten realizar las operaciones de consulta más eficientemente. A continuación describimos el cálculo del Throughput del diseño.

De acuerdo a la tabla 5.3, el periodo mínimo del reloj del sistema es: 9.992 ns. Por lo tanto, la frecuencia máxima del sistema podemos calcularla como se muestra en la ecuación

Recurso	Valor utilizado
Slices	633
Puertos de E/S	387
BRAMs	53
Periodo Mínimo del reloj	9.992ns

Cuadro 5.3: Resultados de la implementación del AES

5.5:

$$f_{Max} = \frac{1}{9{,}992} = 100{,}080MHz \tag{5.5}$$

Con los datos de la ecuación 5.5, podemos calcular el Throughput del diseño. Si consideramos que el AES procesa 128 bits de información, y que de acuerdo a la simulación realizada, proporciona un texto cifrado cada 12 ciclos de reloj. El Throughput se calcula como se expresa en la ecuación 5.6:

$$Throughput = \frac{100{,}080MHz \times 128bits}{12} = 1067{,}52Mbps \tag{5.6}$$

El throughput de nuestro diseño se encuentra entre los más competitivos encontrados hasta el momento en la literatura actual. En la sección 5.5 se brinda una comparación de los resultados de la implementación presentada.

5.4. Resultados de la implementación del modo CCM

Debido a que el objetivo central de la tesis radica en la implementación del modo CCM, es importante establecer la calidad de dicha implementación. Por lo anterior, al igual que en el AES, presentamos los resultados obtenidos con la herramienta de síntesis, los cuales se encuentran en la tabla 5.4.

La tabla 5.4 muestra la cantidad de slices y BRAMs utilizadas por cada uno de los bloques que componen al modo de operación CCM. En estos resultados se encuentran incluidos los valores que corresponden a la implementación del AES, razón por la cual, los bloques de Autenticación y Cifrado utilizan 53 BRAMs cada uno, las cuales representan las utilizadas por el componente AES que tiene cada uno de dichos bloques.

Puede verse que el diseño implementado es relativamente pequeño en términos de área, sin embargo, debido al AES utiliza una cantidad considerable de BRAMS.

Bloque	**Slices**	**BRAMs**
Autenticación	1031	53
Cifrado	713	53
Unidad de Control y Hardware extra	410	0
Modo CCM Completo	2154	106
Periodo mínimo del reloj		9.992 ns

Cuadro 5.4: Resultados de la implementación del modo CCM

Al igual que en el caso del AES, el periodo mínimo del Reloj del Sistema es 9.992 ns, de modo que calculamos el Throughput de la implementación bajo las siguientes consideraciones:

1. Debido a que el tamaño máximo del mensaje (m) que se considera en esta implementación es de 1024 bytes, el número máximo de bloques para el mensaje es de 64 cada uno de 16 bytes (véase la sección 4.4).

2. De la misma forma, el tamaño de los datos a autenticar (a), se fijó en 32 bytes, por lo que únicamente utiliza 2 bloques.

3. Por lo tanto, el número total de bloques que el sistema procesa en su máxima capacidad es de 66 más el bloque correspondiente a B_0, el cual es siempre procesado, lo que da un total de 67 bloques de 16 bytes que el Sistema puede procesar.

4. Cada bloque se procesa en 12 ciclos de reloj. Por lo tanto, procesar los 67 bloques toma en total 804 ciclos de reloj.

5. El número total de bits que la implementación puede procesar es de 66×128 bits $= 8448$ bits. Nótese que se toman los 66 bloques efectivos de información, por lo que no se considera el bloque extra que se calcula por ser considerado como *overhead*.

Con los datos anteriores calculamos el Throughput como se muestra en la ecuación 5.7:

$$Throughput = \frac{66 * 128bits}{804 * 9{,}992ns} = 1{,}05158Gbits/s \tag{5.7}$$

Como puede verse en la ecuación 5.7, el Throughput alcanzado con la implementación presentada en esta tesis es considerablemente alto. Sin embargo es necesario establecer un parámetro de medida para determinar la calidad de la implementación. En la siguiente sección, realizamos la comparación de los resultados presentados.

5.5. Comparación de Resultados

En esta sección presentamos la comparación de los resultados obtenidos con el diseño presentado en esta tesis. Comenzamos con una proyección teórica del rendimiento del diseño basados en las pruebas realizadas con las tarjetas inalámbricas (véase la sección 3.7). Posteriormente, realizamos la comparación de la implementación en FGPA con implementaciones similares encontradas en las especificaciones de algunos productos comerciales, así como las encontradas en la literatura abierta.

5.5.1. Comparación con la implementación en software

Como se mencionó en las sección 5.2, el modo CCM es implementado en software en el controlador que se utilizó para las pruebas con las tarjetas inalámbricas (véase la sección 3.7). Por lo anterior, es necesario destacar el rendimiento que podría obtenerse en la transmisión de la información si la información se protegiera con CCM, utilizando una implementación en Hardware.

Para realizar este análisis, tomamos los resultados presentados en la tabla 5.1. Si consideramos que podemos calcular el tiempo de procesamiento del modo de operación CCM en el controlador de la tarjeta como se muestra en la ecuación 5.1, podemos establecer una comparación teórica del tiempo de procesamiento entre la implementación en el controlador de la tarjeta y nuestra implementación en Hardware Reconfigurable.

Los cálculos para el tiempo de procesamiento en la arquitectura presentada en esta tesis, pueden darse como sigue:

Sea
Th el throughput de nuestra implementación.
n_{veces} el número de veces que se envía el buffer
ov el overhead introducido en cada paquete
$nbits$ la cantidad de bits que se procesarán.
t_{proc} el tiempo que tarda el Sistema en procesar la información.

Calculamos el número de bits que se van a procesar como se muestra en la ecuación 5.8:

$$nbits = n_{veces} \times (1024 + ov) \times 8 \tag{5.8}$$

Si consideramos que por cada paquete que envía en un buffer de 1024 bits de información, debe procesar tres bloques adicionales (uno correspondiente a B_0 y dos para los datos adicionales a autenticar) de 16 bytes cada uno, el *overhead* del sistema es de 48 bytes por cada paquete de 1024 bytes.

Por lo tanto podemos calcular el tiempo de procesamiento como se indica en la ecuación 5.9:

Número de Veces	CCM Controlador (s)	CCM FPGA (s)
10,000	1.185	0.081
20,000	2.085	0.16
30,000	3.32	0.24
40,000	4.455	0.33
50,000	5.375	0.40
100,000	11.07	0.82
200,000	21.605	1.63
300,000	32.815	2.45
400,000	43.825	3.26
500,000	54.544	4.10

Cuadro 5.5: Tiempo de procesamiento teórico para CCM en software y Hardware

$$t_{proc} = \frac{nbits}{Th} \tag{5.9}$$

De esta manera, tomando las ecuaciones 5.8 y 5.9, podemos tomar el tiempo teórico que tardaría la implementación presentada en procesar la información para enviar 10000 veces el buffer de información, lo cual se muestra en la ecuación 5.10:

$$\begin{aligned} nbits &= 10000 \times (1024 + 48) \times 8 = 81{,}92Mb \\ t_{proc} &= \frac{81{,}92Mb}{1{,}05158Gbps} = 0{,}081s \end{aligned} \tag{5.10}$$

Repitiendo el proceso mostrado en la ecuación 5.10, para los valores restantes de las pruebas, y la ecuación 5.1 para hacer el cálculo del tiempo de procesamiento para la implementación del controlador, podemos obtener la tabla 5.5 en la que se muestra la comparación de los tiempos de procesamiento.

Como se puede apreciar en la tabla 5.5, la implementación en Hardware es considerablemente más rápida que la correspondiente en software. Lo anterior se debe a diversos factores, tales como la velocidad de procesamiento que es posible obtener en un FPGA, respecto a las implementaciones en software, lo cual es motivado a ciertas operaciones que en software tienen un costo computacional elevado, mientras que en Hardware su implementación es eficiente, tal como las operaciones XOR, la consulta a tablas y las permutaciones de arreglos.

En la gráfica 5.1 se muestra la gráfica correspondiente a la tabla 5.5. En ella se puede apreciar que el comportamiento de ambas implementaciones es casi lineal, lo cual es lógico, pues el mismo número de operaciones se llevan a cabo únicamente con el incremento de información a manejar.

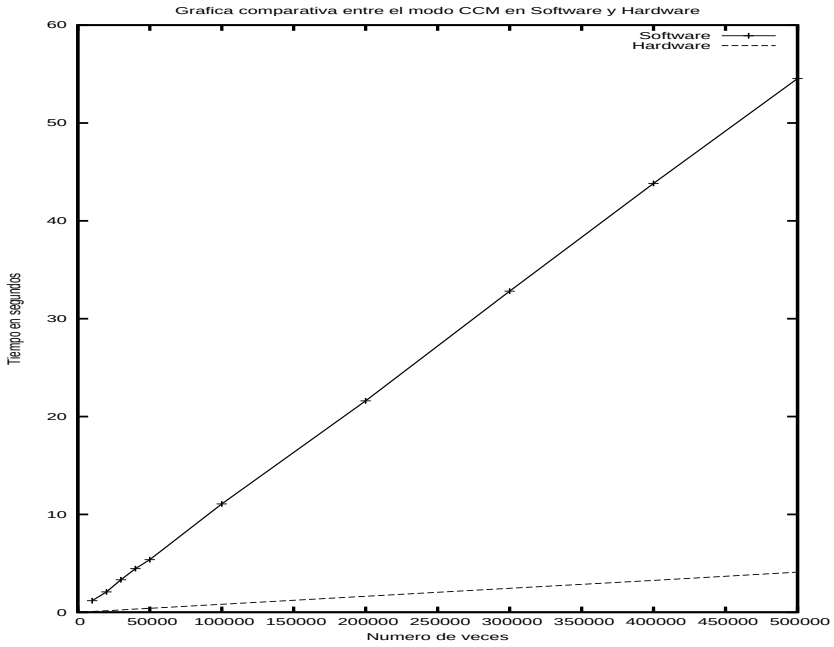


Figura 5.1: Gráfica comparativa de las implementaciones en software y Hardware

Es importante destacar que los resultados presentados son teóricos y corresponden a experimentos de laboratorio. Sin embargo, permiten realizar una aproximación razonable a implementaciones que puedan ser utilizadas en aplicaciones comerciales tales como una tarjeta inalámbrica.

En la siguiente sección, se presenta la comparación con implementaciones semejantes sobre FPGAs.

5.5.2. Comparación con implementaciones en FPGAs

Para poder obtener una comparación más aproximada a la realidad, realizamos una investigación para encontrar implementaciones similares a la presentada en este trabajo, para de esta manera evaluar la calidad de nuestro diseño.

Producto	Empresa	Arquitectura	Throughput(Mbps)
HTRU Aerolink	Ntru	VHDL/software	0.25
SCAES-CCM	SiWorks	RTL, ASIC, FPGAs	582 at 100MHz 192 at 33 MHz
AES-CCM core	Helion	FPGAs y ASIC	Tiny Version 15, Standard 200
AES-CCM	Nuestro diseño	VHDL - FPGA	1,051

Cuadro 5.6: Comparación de la implementación con aplicaciones comerciales

Durante nuestra investigación no encontramos trabajos académicos que reportaran implementaciones completas del modo CCM, por lo que en la tabla 5.6, presentamos una comparación con algunos productos comerciales que se encuentran ya a la venta.

Como se muestra en la tabla 5.6 la implementación desarrollada obtiene el mejor rendimiento sobre las aplicaciones comerciales. A pesar de tratarse de resultados experimentales a nivel prototipo, pueden considerarse como importantes por la ventaja obtenida sobre productos comerciales.

Con la finalidad de obtener una mejor comparación, presentamos la tabla 5.7, la cual recopila algunas de las mejores implementaciones encontradas en la literatura abierta, de las implementaciones de AES en modo iterativo en diferentes modos de operación. Una comparación de esta naturaleza, es considerada como válida, debido a que la mayor parte de los cálculos inherentes al modo CCM son realizados con el AES.

Como se muestra en la tabla 5.7, nuestro diseño es el segundo más rápido para un AES iterativo. Así mismo es también el segundo diseño más económico, pues la relación Throughput/Área (T/A), tiene el segundo mejor coeficiente de acuerdo a la tabla comparativa. Sin embargo, es también el diseño que más utiliza BRAMs.

Conforme a los resultados presentados en este capítulo, podemos decir que el diseño descrito en esta tesis, tiene un nivel competitivo, y puede ser catalogado como eficiente en términos de los recursos utilizados.

Autor	Dispositivo	Modo	Slices(BRAMs)	Throughput(Mbps)	T/A
Charot et al.[34]	Altera APEX	CTR	N/A	512	N/A
Weaver et al.[20]	XVE600-8	ECB	460(10)	690	1.5
Labbé et al.[15]	XCV1000-4	ECB	2151(4)	390	0.18
Saggese et al.[27]	XCVE2000-8	ECB	446(10)	1000	2.3
Chodwiec et al.[5]	XC2530-5	ECB	222(3)	139	0.62
Chodwiec et al.[5]	XC2530-6	ECB	222(3)	166	0.74
Standaert et al.[33]	VIRTEX2300E	ECB	542(10)	1450	2.6
Gaj et al.[8]	XCV1000	ECB	2902	331.5	0.11
Saqib [24]	XCV812E	ECB	2744	258.5	0.09
Amphion CS5220 [1]	XVE-8	ECB	421(4)	290	0.69
Amphion CS5230 [1]	XVE-8	ECB	573(10)	1060	1.9
Segredo et al. [31]	XCV-100-4	ECB	496(10)	417	0.84
Segredo et al. [31]	XCV600E-8	ECB	496(10)	743	1.49
Calder et al. [13]	Altera EPF10K	ECB	1584	637.24	0.40
Nuestro Diseño	Spartan 3 3s4000	ECB	633(53)	1067	1.68
Nuestro Diseño	Spartan 3 3s4000	CBC	1031(53)	1067	1.03
Nuestro Diseño	Spartan 3 3s4000	CTR	731(53)	1067	1.45

Cuadro 5.7: Comparación con implementaciones de AES iterativo

Capítulo 6

Conclusiones

En este capítulo se dan las conclusiones obtenidas durante el desarrollo de esta tesis. Se presenta en primer lugar una síntesis de los resultados alcanzados en este trabajo. Posteriormente se brindan las conclusiones obtenidas mediante el análisis de dichos resultados y finalmente proporcionamos una proyección del trabajo futuro que puede ser realizado para mejorar la calidad del trabajo.

6.1. Resumen de los resultados obtenidos

Durante el desarrollo de esta tesis, se obtuvieron los siguientes resultados:

- Una implementación eficiente de la primitiva de cifrado del cifrador por bloques AES, en su implementación iterativa, la cual es competitiva con respecto a implementaciones similares encontradas en la literatura abierta.
- Una implementación completa del modo de operación genérico denominado CCM, el cual es utilizado en el estándar IEEE 802.11i. Dicha implementación se hizo de manera eficiente, y es considerablemente más rápida que las implementaciones comerciales encontradas en la actualidad.
- Se realizó el análisis de los resultados obtenidos, comparados respecto al experimento efectuado con tarjetas inalámbricas comerciales, en el cual se pudo observar que la eficiencia de la implementación en hardware reconfigurable, es considerablemente mayor a la alcanzada en su contraparte de software.
- Se brindaron las ideas para realizar una paralelización eficiente al proceso de autenticación y Cifrado del modo CCM, con lo cual se disminuyó el tiempo de procesamiento de la información.
- Se obtuvo la implementación de las operaciones realizadas por el emisor (autenticación y cifrado) así como las correspondientes al receptor (descifrado y verificación), en un mismo diseño, el cual es capaz de llevar a cabo ambos procedimientos.

6.2. Conclusiones del trabajo realizado

Durante el desarrollo de esta tesis se han presentado las amenazas existentes para una red inalámbrica y hemos descrito los diversos servicios que tienen que ser satisfechos para que una aplicación pueda ser considerada segura.

Se han descrito también, las alternativas existentes para la protección de la información, y así mismo, hemos mostrado las vulnerabilidades que dichas propuestas tienen. Presentamos las nuevas ideas que se han implementado para garantizar la seguridad de los datos que son transmitidos.

Presentamos la alternativa del modo de operación genérico denominado CCM implementado en una arquitectura de hardware reconfigurable. Describimos los módulos que componen el diseño realizado, y explicamos la manera en que fueron implementados.

Presentamos los resultados obtenidos por el diseño del modo de operación CCM implementado. Consideramos que los resultados mostrados cumplen con las expectativas planteadas, pues se obtuvo una alta eficiencia en el Sistema desarrollado.

Debido al amplio impacto que las redes inalámbricas tienen en la actualidad, resulta indispensable contar con esquemas de seguridad que garanticen la protección de la información que viaja a través de ellas.

La implementación descrita en esta tesis, muestra la conveniencia de contar con una implementación en hardware del modo de operación CCM. Como se reporta en la tabla 5.5, la velocidad de procesamiento es mucho mayor que la alcanzada en la implementación en software, lo que nos permite concluir que es provechoso contar con el modo de operación implementado en hardware.

El trabajo desarrollado representa una contribución significativa, pues no existen implementaciones académicas del modo CCM que hayan sido reportadas hasta el momento, lo cual nos permite situarnos como pioneros en esta área. Es preciso remarcar que el modo de operación fue propuesto en el 2002, y ha sido aceptado desde entonces como un esquema de protección altamente confiable.

La importancia del modo de operación CCM, puede verse en su inclusión al estándar IEEE 802.11i, que se presenta como sucesor del estándar dominante en la actualidad, el IEEE 802.11b. Lo anterior es motivado por la gran cantidad de vulnerabilidades encontradas en el protocolo WEP, el cual es utilizado como método de protección de datos para la versión “b”.

Creemos que el modo de operación CCM, utilizando al AES como cifrador por bloques, será el esquema de seguridad básico en la siguiente generación de tarjetas para redes inalámbricas, por lo que es de gran interés su estudio, así como tratar de obtener implementaciones cada vez más eficientes. Debido a que muchos de los futuros desarrollos para el

CCM serán llevados a cabo en plataformas de hardware, consideramos indispensable contar con una buena implementación en hardware Reconfigurable.

6.3. Trabajo Futuro

A pesar de considerar que los resultados reportados en este trabajo, tienen un alto nivel competitivo, creemos que el diseño está sujeto a futuras modificaciones con la finalidad de obtener implementaciones cada vez más eficientes.

Uno de los campos donde existe más oportunidad para el incremento de eficiencia se encuentra en la implementación del AES. Desde su desarrollo a finales del año 2000, el AES ha sido objeto de análisis con la finalidad de lograr implementaciones más eficientes, lo que ha permitido a diversos autores, reportar cada vez mejores resultados.

En las plataformas de hardware reconfigurable, se pueden encontrar diversas ideas que incrementan el rendimiento del AES, por lo que lograr un incremento en la velocidad de procesamiento de este cifrador, permitirá en consecuencia, obtener un incremento en el Throughput de la implementación del modo de operación CCM, puesto que el costo computacional del CCM está estrechamente ligado al del AES.

La implementación del AES mostrada en esta tesis, utiliza una cantidad importante de BRAMs. Sin embargo, Standaert [32] consigue una implementación con un Throughput más elevado, con un menor número de BRAMs, por lo que una posibilidad para mejorar el diseño presentado consiste en incluir algunas de sus ideas a las implementaciones aquí presentadas.

En cuanto a la implementación del modo CCM se refiere, un trabajo a futuro incluye la posibilidad de agregar una interfaz para el intercambio de datos que permita hacer variables los parámetros, así como la cantidad de información que es procesada.

Consideramos que el rápido desarrollo de la tecnología de hardware Reconfigurable, permitirá en un futuro conseguir una implementación más eficiente, la cual pueda ser incluida en las tarjetas inalámbricas de la próxima generación.

Apéndice A

Primitivas Criptográficas

En este apéndice se describen algunas de las Primitivas Criptográficas que se mencionaron a lo largo de los diferentes capítulos de esta tesis.

A.1. Las Funciones Hash

Una Función Hash Criptográfica h toma un mensaje como entrada, el cual tiene una longitud arbitraria y produce a la salida un "digesto" del mensaje de una longitud fija, por ejemplo 128 bits, como se ilustra en la figura A.1.

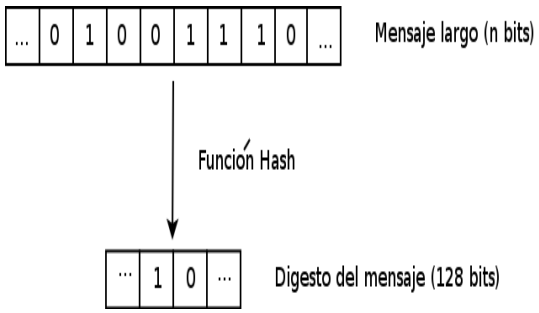


Figura A.1: Una Función Hash

No es sencillo diseñar una función Hash, puesto que tiene que satisfacer ciertas propiedades [39]:

1. Dado un mensaje m, el digesto del mensaje $h(m)$ puede ser calculado rápidamente.

2. Dado un digesto de mensaje y, es computacionalmente imposible encontrar un m con $h(m) = y$. Lo que significa que h es una función de **solo-ida**.

3. Es computacionalmente imposible encontrar mensajes m_1 y M_2 con $h(m_1) = h(m_2)$. Esta condición requiere que h sea una función **libre de colisiones**.

Las entradas en una función Hash son el mensaje M dividido en m bloques de longitud n (por ejemplo 128 bits), y la salida de los bloques anteriores. La salida final es el digesto de todos los bloques, lo cual se puede expresar como la ecuación A.1 [29].

$$h_i = f(M_i, h_{i-1}) \text{ para } i = 0, ..., m \tag{A.1}$$

Debe notarse que dado que el conjunto de mensajes posibles es mucho más grande que el conjunto de posibles digestos de mensaje, deben existir muchos ejemplos de mensajes m_1 y m_2 con $h(m_1) = h(m_2)$. La propiedad 3 debe comprenderse en el sentido de que es extremadamente difícil encontrar ejemplos. En particular, si B produce un mensaje m y su digesto es $h(m)$, A quiere tener la seguridad de que B no conoce otro mensaje m' con $h(m') = h(m)$, incluso si m y m' pueden ser cadenas aleatorias de símbolos.

Una función Hash puede ser impráctica debido a la cantidad de operaciones que son necesarias para implementarla. Sin embargo puede ser calculada de forma eficiente a través de una sucesión de elevaciones al cuadrado, aunque incluso esta técnica puede ser muy lenta para aplicaciones prácticas.

Existen muchas funciones Hash que cuentan con un nivel de fortaleza considerado profesional. Por ejemplo, existe la popular familia MD creada por Rivest. En particular, MD4 y su versión más robusta el MD5 son ampliamente utilizadas para producir un digesto de mensaje de 128 bits con una entrada de longitud arbitraria. Otra alternativa es el Algoritmo Hash Seguro (*SHA* por sus siglas en inglés) del Instituto Nacional de Estándares y Tecnología de los Estados Unidos, (*NIST* por sus siglas en inglés), el cual genera una salida de 160 bits. Una descripción de estos algoritmos puede ser encontrada en [17, 29].

A.2. Los algoritmos MAC (*Message Authentication Code*)

Un Código de Autenticación de Mensaje (*MAC* por sus siglas en inglés), es una etiqueta de autenticación (también llamada suma de verificación) aplicada a un mensaje, la cual es creada a partir de la combinación de un esquema de autenticación (como una función Hash) con una llave secreta. A diferencia de una firma digital, los algoritmos MAC sólo pueden ser calculados y verificados con la misma llave, por lo que únicamente el receptor autorizado puede verificarlos. Existen cuatro tipos de algoritmos MAC [26]:

1. **Incondicionalmente Seguros:** Este tipo de algoritmos están basados en el cifrado del mensaje, donde el texto cifrado autentica al texto en claro, debido a que ninguna entidad externa tiene acceso al cifrador.

2. **Basados en funciones Hash (HMACs):** Utilizan una o varias llaves junto con una función Hash para producir una suma de verificación que es agregada al mensaje.

3. **Basados en Cifradores por flujo de datos:** Estos MAC utilizan un cifrador por flujo de datos para calcular el valor de autenticación del mensaje.

4. **Basados en Cifradores por Bloque:** Utilizan un cifrador por bloques para autenticar el mensaje. Un ejemplo de este tipo de MAC es el CBC-MAC descrito en las secciones 1.1.3 y 3.2.

Los MAC son muy útiles para brindar autenticación sin privacidad. Una forma sencilla de convertir a una función Hash en un algoritmo MAC, es mediante el cifrado del valor Hash con un algoritmo simétrico. Por otro lado, cualquier algoritmo MAC puede ser convertido a un función Hash tradicional haciendo la llave de cifrado pública [29].

Apéndice B

Campos Finitos

En este apéndice damos una explicación sobre los campos Finitos, con la finalidad de comprender algunas de las operaciones realizadas durante la tesis, tales como el paso *Mix-Columns* del AES. Inicamos con algunas definiciones básicas y posteriormente se explican las operaciones sobre los campos finitos.

B.1. Grupo

Un grupo Abeliano $< G, + >$ consiste de un conjutno G y una operación definida en sus elementos, que hemos denotado como "+":

$$+ : G \times G \;\; \rightarrow \;\; G : (a, b) \mapsto a + b \tag{B.1}$$

Con la finalidad de ser considerado un grupo Abeliano, la operación tiene que cumplir las siguientes condiciones:

$$\begin{aligned}
cerrada: &\quad \forall a, b \in G : a + b \in G && \text{(B.2)}\\
asociativa: &\quad \forall a, b, c \in G : (a + b) + c = a + (b + c) && \text{(B.3)}\\
conmutativa: &\quad \forall a, b \in G : a + b = b + a && \text{(B.4)}\\
elemento\ neutro: &\quad \exists\ 0 \in G, \forall a \in G : a + 0 = a && \text{(B.5)}\\
elemento\ inverso: &\quad \forall a \in G, \exists\ b \in G : a + b = 0 && \text{(B.6)}
\end{aligned}$$

El mejor ejemplo de un grupo Abeliano es $< \mathbb{Z}, + >$: el conjunto de los enteros, con la operación "Suma". Dado que los enteros son el mejor ejemplo de un grupo, normalmente el símbolo "+" se utiliza para denotar la operación de grupo arbitraria.

B.2. Anillos

Un anillo R es un conjunto de aquellos elementos que pueden ser sumados y multiplicados, cumpliendo las siguientes condiciones:

- Bajo la operación suma, R es un grupo (Abeliano) aditivo.
- Para cada $x; y; z \in R$ tenemos que $x(y + z) = xy + xz; (y + z)x = yx + zx$:
- Para todo $x; y; z \in R$, tenemos $(xy)z = x(yz)$.
- Existe un elemento $e \in R$ tal que $ex = xe = x$ para todo $x \in R$.

Los números enteros, racionales, reales y complejos pertenecen a la clasificación de anillos. Un elemento x de un anillo es invertible si x tiene un inverso multiplicativo en R, esto es, si existe un $u \in R$ único tal que: $xu = ux = 1$. Al número 1 se le denomina el elemento *unidad* del anillo.

B.3. Campos

Un campo es un anillo en el cual la multiplicación es conmutativa y cada elemento excepto 0 tiene un inverso multiplicativo. Podemos definir el campor F con respecto a la adición y a la multiplicación si:

- F es un grupo conmutativo respecto a la suma.
- $F\backslash\{0\}$ es un grupo conmutativo con respecto a la multiplicación.
- La leyes distributivas mencionadas para el anillo se mantienen.

B.4. Campos Finitos

Un *campo finito* o *Campo de Galois* denotado por $GF(q = p^n)$, es un campo con p característico, y un número q de 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 $\alpha \in GF(q)$ no cero, la identidad $\alpha^{q-1} = 1$ se mantiene.

En criptografía dos casos son los más utilizados: $q = p$, con p siendo un número primo y $q = 2^m$. El primero, $GF(p)$, se denota como "campo primo", mientras que $GF(2^m)$, es conocido como un campo finito de característica 2 o simplemente como "campo de extensión binaria". Este campo es también denotado como F_{2^m}, sus elementos son cadenas de m bits. Las reglas para la aritmética en F_{2^m} pueden ser determinadas tanto por representación polinomial o por representación canónica. Dado que las operaciones en F_{2^m} son en cadenas de bits, las computadoras las pueden realizar de manera muy eficiente. La siguiente sección describe brevemente las bases polinomiales y sus operaciones artiméticas.

B.5. Polinomio sobre un campo

Representamos cada elemento del campo finito binario (F) como una cadena de bits ($a_{m-1}...a_2a_1a_0$), el cual puede ser un polinomio de grado menor que m:

$$a(x) \quad = \quad a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_2x^2 + a_1x + a_0 \tag{B.7}$$

donde,

- x es denomidado la incógnita del polinomio, m es un entero no negativo, y $b_{m-1}, ..., b_0$ ($b_i \in F$) son escalares constantes, denominados los coeficientes del polinomio.
- La potencia más alta de x ($m-1$, si el coeficiente a_{n-1} no es cero) se denomina el grado del polinomio.

B.6. Operaciones sobre polinomios

Definimos dos operaciones, la suma y la multiplicación.

- **Suma:** Sea $c(x)$ la suma de dos polinomios $a(x)$ y $b(x)$, por lo que la adición de polinomios consiste en la suma de los coeficientes con la misma potencia de x, donde dicha operación se realiza sobre el campo F:

$$c(x) = a(x) + b(x) \quad \Leftrightarrow \quad c_i = a_i + b_i, 0 \leq i < n \tag{B.8}$$

  - En la suma, el elemento neutral 0 es el polinomio con todos sus coeficientes iguales a cero.
  - El elemento inverso se encuentra reemplazando cada coeficiente con su inverso en F.
  - 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 $F = GF(2)$.

- **Multiplicación:** Si $m(x)$ es el polinomio de reducción, entonces la multiplicación de dos polinomios $a(x)$ y $b(x)$ es el producto algebraico de los polinomios módulo el polinomio $m(x)$:

$$c(x) = a(x) \cdot b(x) \quad \Leftrightarrow \quad c(x) = a(x) \times b(x)(\text{ mod } m(x)) \tag{B.9}$$

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

Apéndice C

Detalles de las pruebas del CCM con las tarjetas 3COM

C.1. La plataforma de instalación

Las tarjetas fueron instaladas en dos computadoras de escritorio con las siguientes características:

- Procesador: Pentium III a 800 MHz.
- Memoria RAM: 128 Mb
- Disco duro: 30 Gb
- Sistema Operativo: Linux Suse versión 9 distribución profesional.

C.2. Instalación del driver para las tarjetas

Para la realización de estas pruebas se utilizó el driver *Host AP - Linux driver for Prism2/2.5/3* [12], el cual, como su nombre lo indica es utilizado con tarjetas basadas en el chipset Prism2/2.5/3 de la compañía Intersil.

El driver permite operar la tarjeta inalámbrica en el modo *Host AP*, con el cual, la tarjeta se comporta como si fuera el punto de acceso, operación para la cual no es necesaria la actualización del firmware del chip. Así mismo, realiza la implementación del modo CCM en software, lo que permite utilizar este nuevo modo de operación con las tarjetas inalámbricas.

Para instalar el driver mencionado, es necesario que los archivos fuentes del Kernel del Sistema Operativo se instalen, puesto que son necesarios para la compilación del driver.

El driver tradicionalmente se distribuye en tres componentes básicos [12]:

- *hostap-driver*: Driver para Linux para los chips Prism de Itersil, brinda el soporte para el modo Host AP, el manejo del IEEE 802.11 para Puntos de Acceso.
- *hostapd*: Emulador del espacio de usuario para el manejo del IEEE 802.11 extendido, por ejemplo, Autentificador IEEE 802.1X, WPA/WPA2, RADIUS.
- *wpa supplicant*: Programa adicional necesario para las operaciones de cliente de WPA y WPA2.

En los archivos de ayuda del fabricante [12] se proporciona más información referente a cada uno de los paquetes listados anteriormente.

C.3. Compilación del driver

Para la compilación del driver es necesario como primer paso configurar los archivos fuente del kernel. Para hacerlo, se ejecutan las siguientes instrucciones desde la línea de comandos, con privilegios de ROOT:

```
>/usr/src/linux/make cloneconfig
>/usr/src/linux/make dep
```

Una vez que la compilación de los archivos del Kernel ha terminado es necesario cambiar los permisos del directorio /usr/src/linux-2.4.21-99/ para poder realizar la compilación del driver.

Finalmente el driver debe ser compilado desde el directorio donde hayan sido extraídos los archivos de distribución. En el directorio correspondiente al driver "hostap-driver-0.2.4", se debe teclear lo siguiente:

```
>/hostap-driver-0.2.4/make
Posteriormente, cambiar a ROOT y teclear
>/hostap-driver-0.2.4/make install
```

Para los directorios restantes, se omite la instrucción "make install", y únicamente se ejecuta el "make" para compilar los archivos fuente.

C.4. Configuración de las Tarjetas

Después de compilar el driver, se debe cargar el módulo del driver, lo cual se lleva a cabo con la siguiente instrucción (todos los comandos de configuración requieren privilegios de ROOT):

```
>modprobe hostap_pci
```

Cuando el módulo ha sido cargado por el Kernel, el driver crea múltiples instancias de las tarjetas de red, de las cuales *WiFi0* es la interfaz maestra para el dispositivo instalado. Dichas interfaces se encargan de administrar las tramas IEEE 802.11 que son generadas y recibidas por la tarjeta inalámbrica.

Por otra parte "wlan0" es la interfaz de datos por default, la cual es utilizada para realizar la mayoría de las configuraciones, por ejemplo, con una dirección IP así como con los parámetros del comando *"iwconfig"*.

Para las pruebas realizadas fue necesario ejecutar los siguientes comandos de configuración con cada una de las tarjetas inalámbricas a través del comando *"iwconfig"*:

- Se configuró la tarjeta para que trabajara en modo Ad-Hoc, mediante el comando: iwconfig wlan0 mode ad-hoc
- Se le asignó una IP estática con el comando: ifconfig wlan0 193.169.101.1 para la máquina cliente y 193.169.101.2 para la máquina que se usó como servidor.
- Para probar la comunicación entre las tarjetas, se hizo un ping a la otra computadora con la finalidad de verificar que la comunicación pudiera ser establecida.

Con el objetivo de realizar una comparación entre los distintos métodos de cifrado, se configuraron las tarjetas inalámbricas para que llevaran a cabo el cifrado de la información con WEP de 40 y 104 bits y con el CCM, así como el envío de información en claro. Lo anterior fue hecho mediante los siguientes comandos:

- **WEP 40 bits:**iwconfig wlan0 key 0123456789
- **Para el WEP de 104 bits:** iwconfing wlan0 key 00112233445566778899aabbcc
- **AES-CCM:** hostap_crypt_conf wlan0 FF:FF:FF:FF:FF:FF CCMP '0011223344556677 8899aabbccddeeff'. Dicho comando se encuentra en el directorio "hostap-utils-0.2.4" que se genera al extraer los archivos del driver.

C.5. Aplicación de Software para las pruebas

Para poder realizar las pruebas con los diferentes algoritmos criptográficos se programó una aplicación basada en Sockets y bajo el esquema de cliente-servidor. Dicha aplicación fue programada en GNU-C y consta de dos archivos.

C.5.1. Programa Servidor

Este programa se ejecuta en una de las computadoras y es utilizado como "servidor". Su tarea se puede resumir en los siguientes puntos:

- Abre un socket para establecer una comunicación con el cliente.
- Es un servidor dedicado, por lo que no crea "hijos" para cada conexión.
- Una vez aceptada la conexión proveniente del programa cliente, recibe la información enviada. El buffer de transmisión tiene un tamaño de 1024 bytes.
- Cuando el buffer se llena, escribe la información en un archivo llamado "salidaServer.elt".
- Envía un acuse al cliente de que la información ha sido recibida y almacenada y que puede seguir transmitiendo. Este acuse consta de una letra "A".
- Cuando la transmisión de toda la información ha terminado envía el mensaje "recibí la información" al cliente con lo que cierra dicha conexión.
- El servidor vuelve a su estado inicial en espera de una conexión entrante.

Para usar el programa servidor, es necesario enviar desde la línea de comando como parámetro el número de puerto donde se abrirá el socket. Un ejemplo de uso es el siguiente:

```
>./server 2004
Esperando una Conexión.....
```

Cuando una conexión se establece el programa despliega el mensaje:

```
Conexión aceptada, esperando información
```

Cuando la transmisión termina, el programa regresa al estado de esperar una conexión.

C.5.2. Programa Cliente

Este programa se ejecuta en la otra computadora y es utilizado como "cliente". Su funcionamiento se puede resumir en los siguientes puntos:

- Se crea un proceso hijo, que se encargará de realizar las operaciones de envío de información.
- Inicializa el contador de tiempo de transmisión
- Se ejecuta el proceso hijo, el cual realiza las siguientes operaciones:

- Abre un socket para establecer una comunicación con el servidor a través del puerto indicado.
- Intenta establecer una conexión con el servidor indicado en la dirección y por el puerto dado en los parámetros.
- Una vez aceptada la conexión comienza la transmisión.
- Cuando la información contenida en el buffer de transmisión ha sido enviada, espera el acuse del servidor antes de leer el siguiente bloque de información.
- Al recibir el acuse continúa el envío de información hasta enviar el buffer el número de veces especificado en los parámetros de entrada.
- Al terminar el envío de información, espera el acuse final con el mensaje "Recibí la información"

- Al finalizar la espera por el proceso hijo, contabiliza el tiempo de transmisión y escribe el archivo de reporte especificado en los parámetros.

Para usar el programa cliente, es necesario proporcionar desde la línea de comando como parámetro el nombre o dirección del host que ejecuta el programa "servidor", el número de puerto donde se abrirá el socket, el número de veces a enviar el buffer (1024 bytes) y el nombre del archivo de reporte a generar. Un ejemplo de uso es el siguiente:

```
>./client 193.169.101.2 2004 10000 reporteWEP40.elt
Intentando establecer una conexión........
```

Cuando una conexión se establece el programa despliega el mensaje:

```
Conexion establecida, listo para transmitir
```

Cuando la transmisión termina, el programa espera el acuse del servidor. Una vez recibido, genera el archivo de reporte correspondiente y termina la transmisión.

C.5.3. Archivo de Reporte

El formato del archivo de reporte es muy sencillo. Únicamente almacena el número de Kb que fueron enviados, y el tiempo que duró la transmisión. Cada vez que el programa cliente se ejecuta, trata de abrir el archivo de reporte indicado, si el archivo existe, escribe los datos al final del mismo. De lo contrario crea uno nuevo.

Bibliografía

[1] *CS5210-40: High Performance AES Encryption Cores*, 2003.

[2] Ashenden Peter J. *The VHDL Cookbook*. Dept. Computer Science University of Adelaide South Australia, First edition, 1990.

[3] 3COM, http://www.3com.com/other/pdfs/products/en_US/3CRDW696_Enterprise_DS.pdf. *3COM 11 Mbps Wireless LAN PCI Adapter*, 2004.

[4] Borisov Nikita, Goldberg Ian and Wagner David. Intercepting Mobile Communications: The Insecurity of 802.11. In *The proceedings of the 7th anual International Conference on Mobile Computing*, pages 180–189, 2001.

[5] Chodowiec Pawel and Gaj Kris. Very Compact FPGA Implementation of the AES Algorithm. In *Cryptographic Hardware and Embedded Systems-CHES 2003*, pages 319–333, 2003.

[6] Daemen Joan, Rijmen Vincent. *The Design of Rijndael: AES The Advanced Encryption Standard*. Springer-Verlag, First edition, 2002.

[7] Erickson Jon. *HACKING the art of exploitation*. No Starch Press, First edition, 2003.

[8] Gaj Kris and Chodowiec Pawel. Comparison of the hardware performance of the aes candidates using recon gurable hardware. In *In The Third AES Candidate Conference, New York*, 2000.

[9] Housley Russ and Arbaugh William. Security Problems in 802.11-Based Networks. In *Communications of the ACM*, volume 46, pages 31–34, May 2003.

[10] IETF. *RFC2459*. Disponible en: http://www.ietf.org/rfc/rfc2459.txt, 1999.

[11] Jonsson Jakob. On the Security of CTR + CBC-MAC. In *Proceedings of Selected Areas in Cryptography - SAC*, volume 2595, pages 76–93, 2002.

[12] Jouni Malinen, http://hostap.epitest.fi/. *Host AP driver for Intersil Prism2/2.5/3, hostapd, and WPA Supplicant*, 2002.

[13] Jácome-Calderon Germán, Velasco-Medina Jaime, López Hernández Julio. Implementación en Hardware del algoritmo Rijndael [in spanish]. In *X Workshop IBERCHIP*, page 113, 2004.

[14] Kong Jiejun, Das Shirshanka, Tsai Edward, Gerla Mario. ESCORT, A Descentralized and Localized Access Control System for Mobile Wireless Access to Secure Domains. In *Proceedings of the 2003 ACM Workshop on Wireless Security*, pages 51–60, 2003.

[15] Labbé A., Pérez A. AES Implementations on FPGA: Time Flexibility Tradeoff.

[16] Mallick Martyn. *Mobile and Wireless Design Essentials*. WILEY, First edition, 2003.

[17] Menezes Alfred J., Van Oorschot Paul C., Vanstone Scott A. *Handbook of Applied Cryptography*. CRC Press, Fifth edition, 2001.

[18] Moen Vebjϕrn, Raddum Havard and Hole Kjell J. Weaknesses in the Temporal Key Hash of WPA. In *ACM SIGMOBILE Mobile Computing and Communications Review*, volume 8, pages 76–83, April 2004.

[19] Nancy Cam-Winget, Russ Housley, David Wagner, and Jesse Walker. Security Flaws in 802.11 Data Link Protocols. In *Communications of the ACM*, volume 46, pages 35–39, May 2003.

[20] Weaver Nicholas and Wawrzynek John. High Performance, Compact AES implementations in Xilinx FPGAs. Technical report, U.C. Berkeley BRASS group, available at http://www.cs.berkeley.edu/ñnweaver/sfra/rijndael.pdf, 2002.

[21] Nichols Randall K., Lekkas Panos C. *Wireless Security Models, Threats and Solutions*. McGraw-Hill TELECOM, First edition, 2002.

[22] Repairfaq, http://www.repairfaq.org/filipg/LINK/F_crc_v3.html. *A painless guide to CRC error detection*, 1996.

[23] Repairfaq, http://www.repairfaq.org/filipg/LINK/F_crc_v3.html#CRCV_002. *A painless guide to CRC error detection Chap7 A fully worked example* , 1996.

[24] Rodriguez-Henriquez F. , Saqib N. A. and Diaz-Perez A. 4.2 Gbit/s Single-Chip FPGA Implementation of AES Algorithm. In Springer-Verlag, editor, *ELECTRONICS LETTERS*, volume 39, pages 1115–1116, 2003.

[25] Rogaway Phillip, Bellare Mihir and Black John. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *ACM Transactions on Information and System Security (TISSEC)*, volume 6, pages 365–403, 2003.

[26] RSA Laboratories. *Crypto FAQ, version 4.1.* Disponible en: http://www.rsasecurity.com/rsalabs/node.asp?id=2177, 2004.

[27] Saggese G.P., Mazzeo A., Mazzocca N. and Strollo A.G.M. An FPGA-Based Performance Analysis of the Unrolling, Tiling, and Pipelining of the AES Algorithm. In *Field-Programable Logic and Applications*, pages 292–302, 2003.

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

[29] Schneier Bruce. *Applied Cryptography*. WILEY, Second edition, 1996.

[30] Scott Fluhrer, Itsik Mantin and Adi Shamir. Weaknesses in the Key Scheduling Algorithm of the RC4. In *Eighth Annual Workshop on Selected Areas in Cryptography*, August 2001.

[31] Segredoas Alejandro, Zabala Enrique y Bello Gustavo. Diseño de un procesador criptográfico Rijndael en FPGA. In *X Workshop IBERCHIP*, page 64, 2004.

[32] Standaert François-Xavier. *Secure and Efficient Use of Reconfigurable Hardware Devices in Symmetric Cryptography*. PhD thesis, Université catholique de Louvain, June 2004.

[33] Standaert François-Xavier, Rouvroy Gael, Quisquart Jean-Jacques and Legat Jean-Didier. Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs. In *Cryptographic Hardware and Embedded Systems-CHES 2003*, pages 334–350, 2003.

[34] Standaert François, Yahya Eslam and Wagner Charles. Efficient Modular-Pipelined AES Implementation in Counter Mode on ALTERA FPGA. In *Field-Programable Logic and Applications*, pages 282–291, 2003.

[35] Struick Rene. Comments NIST Draft Pub 800-38C. Technical report, Certicom, Disponible en: http://csrc.nist.gov/CryptoToolkit/modes/comments/800-38_Series-Drafts/CCM/Struick_CCM_comments%20.pdf, October 2003.

[36] Stubblefield Adam, Ioannidis John, and Rubin Aviel D. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. Technical report, ATT Labs TD-4ZCPZZ, Disponible en: http://www.cs.rice.edu/~astubble/wep., August 2001.

[37] Stubblefield Adam, Ioannidis John, and Rubin Aviel D. A key Recovery Attack on the 802.11b Wired Equivalent Privacy Protocol (WEP). In *ACM transactions on information and System Security (TISSEC)*, volume 7, pages 319–332, May 2004.

[38] Sánchez Santiago Mizael. Implementación en Hardware-Software del algoritmo criptográfico DES. Master's thesis, CINVESTAV-IPN, Julio 2003.

[39] Trappe Wade and Washington Lawrence C. *Introduction to Cryptography with Coding Theory*. Prentice Hall, First edition, 2002.

[40] Weatherspoon Sultan. Overview of IEEE 802.11b Security. Technical report, Network Communications Group, Intel Corporation, Disponible en: http://developer.intel.com/technology/itj/q22000/pdf/art_5.pdf.

[41] Whiting Doug, Housley Russ, Ferguson Niels. Counter with CBC-MAC (CCM). In *Submission to NIST*, 2002.

[42] Wollinger Thomas and Paar Christof. How Secure Are FPGAs in Cryptographic Applications? In *Field-Programable Logic and Applications*, pages 91–100, 2003.

[43] Wollinger Thomas, Guajardo Jorge and Paar Christof . Security on FPGAs: State-of-the-art implementations ans attacks. In *ACM Transactions on Embedded Computing Systems (TECS)*, volume 3, pages 534–574, 2004.

[44] Xilinx, http://www.xilinx.com/bvdocs/publications/ds099.pdf. *Spartan-3 FPGA Family: Complete Data Sheet*, January 2005.

Índice alfabético

Todo tiene su tiempo, y todo lo que se quiere debajo del cielo tiene su hora

Eclesiastés 3:1