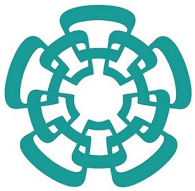

Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Esquemas de Cifrado Autenticable Determinista: Implementaciones y una Nueva Propuesta de Esquema

Tesis que presenta

Alejandro García Luna

para obtener el Grado de

Maestro en Ciencias

en la Especialidad de

Computación

Director de la Tesis

Dr. Debrup Chakraborty

México, D.F. **Julio 2012**

Agradecimientos

Quiero agradecer a mi familia y amigos por su apoyo en esta parte tan importante de mi vida.

También quisiera agradecerle a mi asesor el Dr. Debrup Chakraborty por todo el tiempo y atención que le dedicó a este proyecto, es un ejemplo de esfuerzo, trabajo y disciplina, todavía tengo mucho que aprender de personas como él.

A todos los profesores del departamento de computación, por que he aprendido mucho de ellos.

Al Dr. Francisco Rodríguez, por sus consejos y recomendaciones.

Al Dr. Luis Gerardo de la Fraga, por los comentarios hechos a este trabajo.

A Sofi por estar al pendiente de todos los alumnos.

A la Red temática del TIC del Conacyt (http://www.redtic-conacyt.mx/), en especial a su línea de seguridad por el apoyo que me ofrecieron.

Al CINVESTAV y al CONACYT por permitirme estudiar en esta gran institución que es el CINVESTAV.

Índice general

Abstract

Two most important security services provided by cryptographic algorithms are privacy and authentication. Informally, an encryption scheme is said to provide privacy if no computationally bounded adversary can obtain any information regarding the plaintext from the ciphertext. The scheme is said to provide authentication if no computationally bounded adversary can create a valid ciphertext, i.e., if an adversary changes the ciphertext while it is in transit, such a change can be easily detected. Deterministic authenticated schemes (DAE) are a class of symmetric key encryption scheme which provides security both in terms of privacy and authentication. As the name suggest, they are deterministic in contrast to other authenticated encryption schemes which are either randomized or stateful. DAEs were first proposed as a solution to the key wrap problem, but can be useful for many other applications where both the services of privacy and authentication are simultaneously required. In this thesis we study deterministic authenticated encryption schemes. As a first contribution, we implement two existing DAE schemes namely HBS and BTM in re-configurable hardware. Both BTM and HBS are schemes which uses block-cipher along with a kind of polynomial hash. In our implementations we use the advanced encryption standard (AES) with 128 bit key as the block cipher and realize the polynomial hash using a field multiplier in GF(2^{128}). Our AES design utilizes ten pipelined stages, which gives us very good throughput. The field multiplier also uses a pipelined design of four stages. We also device ways to compute polynomials efficiently using

pipelined multipliers. As our second contribution we propose a new construction of a DAE which uses stream ciphers with initialization vectors. The new scheme is called SCDAE. We prove that SCDAE is a secure DAE. Also we argue about the efficiency of SCDAE using operation counts.

Resumen

Los dos servicios más importantes en seguridad proveídos por la criptografía son la privacidad y la autenticación. Informalmente, un esquema de cifrado se dice que provee privacidad si no existe adversario alguno con límites computacionales que pueda obtener alguna información a partir del texto plano o del texto cifrado. Se dice que el esquema provee autenticación si ningún adversario computacionalmente restringido puede crear un texto cifrado válido, es decir, si un adversario cambia el texto cifrado mientras este transita, dicho cambio puede ser fácilmente detectado. Los esquemas de cifrado autenticado determinista (CAD) son una clase de esquema de cifrado de llave simétrica que provee seguridad tanto en el sentido de privacidad como en el de autenticación. Como el nombre sugiere, Estos deterministas en contraste a otros esquemas de cifrado autenticado que utilizan estados o necesitas de valores aleatorios. Los CAD fueron propuestos como una solución al problema de la envoltura de llave, pero pueden ser útiles para muchas otras aplicaciones donde ambos servicios de privacidad y autenticación son requeridos de manera simultánea. En esta tesis nosotros estudiamos los esquemas de cifrado determinista autenticado. Como primera contribución, nosotros implementamos dos de los esquemas CAD existentes, HBS y BTM, en hardware reconfigurable. Ambos, BTM y HBS, son esquemas que usan un cifrador por bloques junto con una especie de función picadillo polinomial. En nuestras implementaciones usamos el AES con una llave de 128 bits como cifrador de flujo y realizamos una función picadillo polinomial haciendo uso de un multiplicador de campo GF(2^{128}).

Nuestro diseño utiliza una implementación de AES en tubería con diez estados, la cual brinda buen rendimiento. El multiplicador de campo tiene un diseño en tubería de cuatro estados. También desarrollamos formas de computar los polinomios con el multiplicador en tubería de forma eficiente. Como una segunda contribución nosotros proponemos una nueva construcción de esquema CAD el cual hace uso de cifradores de flujo con vectores iniciales. El nuevo esquema es llamado SCDAE. Probamos que SCDAE es seguro en el sentido de un DAE. También discutimos sobre la eficiencia de SCDAE con respecto a sus operandos.

Capítulo 1

Introducción

1.1. Antecedentes

La criptografía moderna es el estudio de técnicas para conseguir seguridad en la información digital, transacciones y computación distribuida [1].

El servicio más antiguo de la criptografía es el de proveer comunicación secreta entre dos entes (privacidad), sin embargo, no es el único servicio que la criptografía puede ofrecer, otro servicio importante y de interés para este trabajo es el servicio de autenticación que consiste en dar la capacidad a estos entes de comprobar que el mensaje no ha sido modificado durante su transmisión.

La manipulación de un mensaje para que este solamente pueda ser interpretado por las personas autorizadas se denomina cifrado, y a su vez, la acción de interpretar correctamente un mensaje cifrado se llama descifrado. Para que el sistema funcione, es necesario que las personas autorizadas para leer el mensaje cuenten con alguna ayuda que permita interpretar el mensaje de manera eficiente y correcta, dicha ayuda recibe el nombre de "llave". Por otra parte, descifrar dicho mensaje sin la llave debe de resultar lo más difícil posible.

Otro de los servicios que proveé la criptografía moderna es la autenticación, este servicio consiste en poder verificar, con cierta seguridad, que un mensaje no haya sido modificado por entes no autorizados. Es común que para poder verificar la auten-

ticidad de un mensaje se agreguen pequeñas cadenas de información al mensaje, a dichas cadenas se les suele llamar MAC (Message Authenticated Code, o código de autenticación de mensaje).

1.1.1. Criptografía moderna: criptografía simétrica y asimétrica

La criptografía moderna está dividida en dos grandes ramas, la criptografía simétrica y la criptografía asimétrica. En la criptografía simétrica, o de llave privada, los entes que desean comunicarse comparten una misma llave, tienen un secreto compartido. El emisor utiliza dicha llave para cifrar ("revolver") el mensaje antes de enviarlo, mientras que el receptor utiliza la misma llave para descifrar ("ordenar") dicho mensaje.

En la criptografía asimétrica o de llave pública existen dos llaves, una pública y una privada. Un usuario de un sistema criptográfico de llave pública debe de crear y tener ambas llaves, las cuales son dependientes una de la otra, la llave privada debe de ser mantenida en secreto mientras que la llave pública puede ser difundida sin problemas. La forma en que funciona el esquema de llave pública para el envío de mensajes cifrados es el siguiente: cuando alguien desea enviar algún mensaje cifrado al usuario dueño de las llaves, el emisor hará uso de la llave pública del usuario para cifrar dicho mensaje, para que cuando este sea recibido por el usuario, pueda ser interpretado con su llave privada. A pesar de que la criptografía asimétrica ofrece varias ventajas con respecto a la criptografía simétrica, la criptografía asimétrica resulta ser muy costosa tanto para cifrar como para descifrar mensajes, por lo que solamente se utiliza para cifrar mensajes pequeños, que bien podrían ser llaves de un esquema de cifrado simétrico.

En la criptografía simétrica se puede observar, a grosso modo, dos tipos básicos de bloques de construcción: los cifradores por bloques y los cifradores de flujo. Los cifradores de bloques son funciones que, dado un texto en claro de una longitud predeterminada y una llave, producen un texto cifrado de la misma longitud que el

texto en claro. Un ejemplo de un cifrador por bloques es AES (Advanced Encryption Standard)[2], el cifrador por bloques recomendado por el NIST (National Institute of Standards and Technology), el cual recibe dos parámetros: un texto en claro de 128 bits de longitud y una llave de 128, 192 o 256 bits de longitud y devuelve un texto cifrado de 128 bits.

Los cifradores de flujo funcionan como generadores pseudoaleatorios de bits, es decir, son funciones que a partir de una cadena de bits pequeña, esto es la llave, generan una cadena bits lo suficientemente grande y aparentemente aleatoria como para cifrar un mensaje. La cadena de bits grande es utilizada para enmascarar, mediante la operación XOR, el mensaje a cifrar. Un cifrador de flujo es, en general, más rápido que un cifrador por bloques, sin embargo, en la práctica, los cifradores por bloques suelen ofrecer mejor seguridad que los cifradores por flujo.

1.1.2. Modos de operación de cifradores de bloque

Como se puede ver, los cifradores por bloques permiten cifrar únicamente mensajes de longitud fija, más aún, tanto el cifrador por bloques como el cifrador de flujo solamente pueden ofrecer el servicio de privacidad. Para un uso práctico de ambos tipos de cifradores es necesario emplear esquemas de cifrado. Los esquemas de cifrado pueden ser vistos como un conjunto de algoritmos que indican como deben de ser cifrados y descifrados los mensajes para conseguir un conjunto de servicios criptográficos.

Los modos de operación son esquemas de cifrado que hacen uso de un cifrador por bloques. El modo de operación más básico es el conocido como *electronic code book* (ECB). El modo de operación ECB funciona de la siguiente forma: se divide el mensaje en claro en pequeños bloques de datos, estos bloques deben de ser del tamaño requerido por el cifrador por bloques, luego cada bloque del texto en claro es cifrado de manera independiente, finalmente se unen todos los bloques cifrados, en el orden en que fueron obtenidos, obteniendo así el mensaje cifrado listo para ser enviado. Sin embargo este modo de operación, aunque sencillo y rápido, es inseguro, ya que es

posible obtener mucha información del mensaje original a partir del mensaje cifrado.

Cifrar mensajes de longitud arbitraria u obtener otros servicios de seguridad mediante el uso de cifradores por bloque no es una tarea trivial.

Existen muchos modos de operación y cada uno se ha diseñado buscando un objetivo en particular. Las categorías más importantes de modos de operación con cifradores por bloque pueden ser separados en los siguientes tres tipos:

- **Solo privacidad:** Aquellos que sólo cifran la información con la intención de que no pueda ser interpretada por personas no autorizadas, es conocida como modos de operación de solo privacidad (privacy only).

  Entre los modos de operación de solo privacidad están el *cipher block chaining* (CBC), *cipher feedback* (CFB), *output feedback* (OFB), *counter mode* (CTR).

- **Cifrado Autenticado:** Los modos de operación de cifrado autenticado, o Authenticated Encryption (AE), son modos de operación que ofrecen, además del servicio de privacidad, el servicio de autenticación de datos. Dichos modos de operación agregan al mensaje cifrado una cadena de bits que sirve para verificar la integridad de los datos enviados, dicha cadena de bits es conocida como MAC (Message Authentication Code). Entre los modos de operación autenticados están IAPM [3], OCB [4], EAX [5], CWC [6], CCM [7] y el GCM [8].

- **Esquemas de Cifrado Entonable:** Los esquemas de cifrado TES o esquemas de cifrado entonables (Tweakable Enciphering Schemes) son esquemas diseñados para cifrar y descifrar información de discos duros, así como para permitir la autenticación de dicha información. En estos esquemas se espera que el mensaje cifrado sea de una longitud idéntica al mensaje en claro, por lo que agregar un MAC al mensaje cifrado es inviable. Para superar este impedimento, los esquemas TES son construidos de tal forma que resulte imposible recuperar la información cifrada si esta ha sufrido un cambio mientras estaba cifrada. Los esquemas TES hacen uso de un dato de entrada extra llamado tono o ajuste (Tweak), se espera que este tono incremente la variabilidad de los mensajes

cifrados obtenidos. Existen distintas propuestas de TES como lo son el EME[9], CMC[10], HCTR [9], HCH[11], HMCH[12], etc. La principal aplicación de los TES es en el cifrado de dispositivos de almacenamiento que estén divididos en bloques, como los discos duros. Si se está utilizando un TES para cifrar discos duros, el tono sería el número de sector de disco, de esta forma dos sectores del mismo disco, con la misma información y cifrados con la misma llave, parecerían contener información distinta para cualquiera que no tuviera la llave.

Los esquemas conocidos como CAD (Cifrado Autenticado Determinista) son modos de operación que ofrecen los servicios de autenticación y privacidad. Para ofrecer autenticación, estos esquemas suelen hacer uso de MACs, por lo que no suelen preservar la longitud del mensaje. Otra característica importante de los CAD es que consideran un conjunto de datos adjuntos al mensaje a cifrar o cabeceras (headers), estos datos no serán cifrados y serán enviados junto con el mensaje cifrado. Finalmente, la característica más importante de los CAD, es el hecho de que no utilizan valores aleatorios ni almacenan algún tipo de información entre usos de los esquemas. Otros esquemas como el modo contador u otros AE, hacen uso de un valor aleatorio para proveer el servicio de privacidad o guardan un "estado" entre cada uso. Los esquemas de cifrado CAD fueron presentados en [13] donde se definen los requerimientos para que un CAD sea considerado seguro y se presenta el modo de operación Synthetic IV (SIV). Después Tetsu Iwata y Kan Yasuda presentan dos modos de operación más, hash block stealing (HBS) [14] y bivariative tag mixing (BTM)[15].

1.1.3. Seguridad demostrable

La seguridad demostrable es el estudio criptográfico de esquemas y primitivas criptográficas, con la finalidad de determinar la seguridad que estos poseen y las condiciones necesarias para que esta sea lo más alta posible.

La seguridad demostrable de un modo de operación es una demostración formal de su seguridad bajo ciertas condiciones. Hasta el momento solo existe un sistema criptográfico probado seguro por completo: el One Time Pad [1]. Sin embargo dicho

sistema es poco eficiente y los requerimientos para su implementación son difíciles de conseguir; por ejemplo, requiere de una llave generada de manera completamente aleatoria de longitud igual a la longitud del mensaje a cifrar, es decir, si el mensaje a cifrar es de 2 giga bytes de información, se requiere que la llave sea dos giga bytes; además, cada vez que se quiera cifrar un mensaje, es necesario generar una llave nueva (nunca antes usada) de manera completamente aleatoria, esta llave debe de ser conocida tanto por el emisor del mensaje como por el receptor de este. Estos requerimientos presentan varios problemas, como son el almacenamiento y protección de llaves tan grandes, la generación de números realmente aleatorios y la forma en que la llave pueda llegar a manos de ambos usuarios sin poner en peligro el sistema criptográfico.

Sin embargo, la seguridad perfecta no es necesaria para la mayoría de los casos, usualmente basta con que el mensaje se mantenga secreto durante un lapso lo suficientemente grande, por ejemplo, es suficiente que un mensaje sea indescifrable por 50, 500 o 50 000 años. En general estamos interesados en esquemas eficientes que provean "seguridad computacional". De manera informal se dice que un esquema es seguro en el sentido de seguridad computacional si para cualquier adversario que utilice una cantidad "razonable" de recursos computacionales, la probabilidad de que dicho adversario tenga éxito en vulnerar la seguridad del esquema es bastante baja.

La seguridad de cualquier modo de operación que usa cifradores por bloques depende fuertemente del cifrador por bloques con el que es implementado. Es trivial ver que si el cifrador por bloques es inseguro, los modos de operación resultantes serán inseguros. Debido a que no se sabe que tan seguros son los cifradores por bloques actuales, al momento de crear un modo de operación se hace uso de cifradores por bloques teóricos, es decir, no se define el modo de operación sobre un cifrador por bloques en específico, sino que se describe el comportamiento esperado por el cifrador por bloques que se utilice para dicho modo de operación. El comportamiento perfecto de un cifrador por bloques es modelado mediante un objeto teórico llamado permutación pseudoaleatoria. Informalmente una permutación pseudoaleatoria es

una permutación con llave, cuyas salidas son indistinguibles de cadenas aleatorias.

En un análisis típico de un modo de operación que hace uso de cifradores por bloques en el paradigma de la seguridad demostrable, primero se trata de definir su seguridad en términos concretos. La seguridad de un esquema es definida de tal modo que este cubre todos los ataques prácticos posibles a dicho esquema. Una vez que la definición es obtenida, se da un argumento reduccionista con el cual se concluye que si cualquier adversario compromete la seguridad del esquema, entonces existe otro adversario que puede romper la seguridad del cifrador por bloques.

1.2. Contribuciones y alcance de la tesis

En esta tesis se realiza un estudio de los esquemas de cifrado autenticado determinista. Dicho estudio está compuesto por dos partes distintas: la primera (capítulo 3 y 4) se enfoca en el problema de la implementación eficiente en hardware reconfigurable de dos de los esquemas actualmente existentes; en la segunda parte proponemos un nuevo esquema CAD que resulta ser bastante distinto a las propuestas actuales.

A pesar de que en los ultimos años se han desarrollado numerosos estudios concernientes a la construcción de esquemas CAD probablemente seguros, no se han reportado resultados de implementaciones eficientes en la literatura. Nosotros presentamos implementaciones eficientes de los esquemas HBS y BTM, y presentamos medidas de desempeño de los esquemas en hardware reconfigurable. Hasta donde conocemos, las implementaciones presentadas en esta tesis son las primeras de su tipo en la literatura abierta.

HBS y BTM (como se describen en el capítulo 2) usan cifradores por bloques y funciones picadillo polinomiales. Asi, para implementar estos esquemas es necesario un cifrador por bloques y un multiplicador de campo finito. Al momento de escribir esta tesis el cifrador por bloques más usado es el AES (siglas en inglés de Advanced Encryption Standard), por lo que nuestras implementaciones hacen uso de un diseño eficiente de este. Nosotros realizamos nuestro AES en 10 etapas en tubería de manera

similar a la descrita en [16]; este AES produce un bloque cifrado cada ciclo de reloj una vez que se ha cumplido la latencia inicial de 10 ciclos que es requerida para llenar la tubería. Adicionalmente, dicho AES ocupa un área razonable y tiene una ruta crítica[1] pequeña. El multiplicador ha sido diseñado para tener una ruta crítica similar a la del AES. Entre las distintas posibilidades para el diseño del multiplicador, seleccionamos el diseño de un multiplicador de cuatro estados en tubería el cual tiene una ruta crítica bastante similar al de nuestro AES.

Como se ha mencionado previamente, el multiplicador es usado para computar ciertos tipos de polinomios. La forma más común para computar un polinomio es utilizando la regla de Horner, donde se necesitan realizar m multiplicaciones para evaluar un polinomio de grado m. Un problema con la regla de Horner es que cada multiplicación es dependiente de las multiplicaciones realizadas previamente, por esto, el uso de un multiplicador en tubería para evaluar polinomios usando dicha técnica parece no ser óptimo. En un multiplicador en tubería, digamos de cuatro estados (como el que usamos en nuestra implementación), lo común es dividir las tareas de una multiplicación en cuatro fases distintas, de esta forma cuatro multiplicaciones pueden ser realizadas en paralelo y como consecuencia, uno puede obtener un resultado cada ciclo de reloj después de la latencia inicial de cuatro ciclos; necesarios para llenar la tubería del multiplicador. Para obtener el mejor desempeño de esta técnica, es necesario que en cada ciclo cuatro multiplicaciones distintas se encuentren disponibles. Si uno usa solo la regla de Horner , entonces esto no es posible. Para aprovechar el multiplicador en tubería, nosotros dividimos el polinomio en 4 partes distintas y computamos cada parte de forma independiente utilizando la regla de Horner. Esto nos permite obtener cuatro multiplicaiones independientes en cada ciclo, y así mantener la tubería llena en todos los ciclos. Después estos cuatro polinomios son combinados para obtener el polinomio deseado. Dicha tecnica de separación ha sido usada previamente en [16]

[1]La ruta crítica es la distancia mayor que debe de recorrer una señal durante un ciclo de reloj en un circuito, entre más largo sea dicho camino más tiempo tarda la señal en transitarlo. La ruta crítica determina la máxima frecuencia que un circuito puede alcanzar. De manera indistinta se hará uso del término ruta crítica para referirse tanto a la distancia máxima, como al tiempo requerido por una señal para recorrerla.

para dividir la evaluación polinomial en tres partes y así optimizar el cómputo de esta por parte de un multiplicador de tres estados en tubería. Hemos modificado la técnica descrita en [16] para que se adapte a nuestro multiplicador.

Las implementaciones que reportamos son para mensajes de longitud fija, y han sido optimizados para funcionar con mensajes de 4kB. Si bien estas implementaciones han sido diseñadas específicamente para esta longitud, el diseño presentado es escalable y se puede adaptar para cualquier longitud fija de mensaje. Además se presenta una implementación del esquema BTM para mensajes y cabeceras de longitud variable. Los resultados reportados son de implementaciones hechas sobre una FPGA Xilinx Virtex 5, y nuestras implementaciones, tanto BTM como HBS, pueden conseguir un rendimiento de alrededor de 15 Gigabits por segundo (Ver el capítulo 4 para resultados más detallados).

La primera vez que se propuso un CAD fue en [13] como una solución al problema de envoltura de llave, después en [17] se declara que no se requiere de todo el mecanismo de un DAE para el propósito de cifrar llaves. Sin embargo existen muchas otras áreas de posible aplicación para los CAD, pero hasta la fecha dichos escenarios no han sido explorados de manera extensa. Nosotros creemos que existen áreas de aplicación donde tener un algoritmo que utilice valores aleatorios o estados (como los esquemas de cifrado autenticado) no pueden se aplicados de manera adecuada debido a restricciones de ancho de banda, etc. Se debe notar que en el caso de un esquema AE (por sus siglas en inglés 'Authenticated Encryption') [18], los algoritmos más usados utilizan estados, en el sentido de que el algoritmo requiere de un valor que no se repita, dicho valor recibe el nombre de nonce, el cual tiene que ser enviado al receptor como parte del texto cifrado. Esto conlleva a una expansión extra del texto cifrado y requiere de más banda ancha. Los altos índices de información cifrada que nuestra implementación alcanza, demuestran que los CAD pueden ser usados en aplicaciones donde se requiere de grandes índices de velocidad en transmisión de datos, pero uno debe de ser cuidadoso en escoger las aplicaciones donde se utilizarán los CAD, ya que las garantías de seguridad provistas por los esquemas CAD son inferiores a las

ofrecidas por los esquemas AE. Una posible área de aplicación puede ser el cifrado de discos. El bien aceptado modelo para cifrado de discos es una clase de esquemas de cifrado llamados esquemas de cifrado entonable (TES por sus siglas en inglés)[10], los cuales son esquemas que preservan la longitud del mensaje al ser cifrado. En [13] se muestra que un TES es un CAD que preserva la longitud, debido a esto podemos concluir que un CAD y TES proveen la misma seguridad, por lo que en términos de seguridad no debe de haber problemas en el uso de un CAD como cifrador de discos duros, el problema consiste en acomodar el texto cifrado en el disco duro dado que el texto cifrado tiene una longitud superior al texto en claro. La respuesta a esto puede estar en el elegir un formato adecuado para cada sector de disco cifrado de tal forma que permita el almacenamiento de la expanción del mensaje. Los índices de cifrado para HBS y BTM que reportamos en este documento son mejores que los índices mostrados por las implementaciones más eficientes de TES[16], lo cual prueba que la eficiencia no será un cuello de botella para dichas aplicaciones[2].

En la segunda parte de la tesis (Capítulo 4) desarrollamos un nuevo esquema CAD. Hemos llamado a este esquema SCDAE el cual usa un cifrador de flujo de una forma novedosa. Todos los CAD existentes son construidos usando un cifrador por bloques, por lo que hasta donde sabemos, SCDAE es la primera construcción de un CAD que hace uso de un cifrador de flujo. SCDAE puede ser construido usando cualquier cifrador de flujo basado en vectores de inicialización (IV). Todos los cifradores de flujo modernos en uso; como el Trivium, Grain, Mickey, HC-128, Rabbit etc [19]; estan basados en vectores de inicialización, por lo que cualquiera de estos puede ser usado para construir un SCDAE. Proveémos una comparación teórica de SCDAE con los esquemas HBS y BTM basados en el número y tipo de operaciones necesarias por parte del esquema. También, proveemos la prueba de seguridad de SCDAE. La suposición fundamental requerida para probar SCDAE es que el cifrador de flujo

[2]Sabemos de la posibilidad de utilizar los esquemas CAD para el cifrado de discos por Chakraborty y Mancillas, actualmente ellos se encuentran explorando los caminos para utilizar los esquemas CAD para el cifrado de discos. El trabajo se encuentra en proceso, y en esta tesis no ahondaremos en esta posibilidad.

es una función pseudoaleatoria con respecto al vector inicial, se ha visto que dicha suposición en los cifradores de flujo modernos es bastante razonable [20]. Nuestro teorema en SCDAE afirma que si es posible realizar un ataque exitoso en contra de SCDAE, entonces es posible realizar un ataque exitoso al cifrador de flujo sobre el que esté construido, la suposición de que el cifrador de flujo es seguro nos lleva a la conclusión de que SCDAE es seguro. La técnica de demostración esta basada en la técnica de secuencia de juegos y es similar a las demostraciones en [14][15][20].

Esta tesis esta compuesta por, además del presente, cinco capítulos más, el contenido de cada capítulo está resumido a continuación:

Capítulo 2: Se presenta a profundidad los esquemas CAD, se hace un análisis de su definición de seguridad y una breve discusión sobre los esquemas existentes. En este capítulo se plantean las principales características que se esperan de un esquema CAD eficiente y seguro.

Capítulo 3: Se definen las características de los bloques de construcción básicos para poder realizar las implementaciones de los esquemas HBS y BTM.

Capítulo 4: Se muestran las implementaciones de los esquemas BTM y HBS, así como los detalles de dichas implementaciones. En este capítulo también se presentan los resultados experimentales obtenidos con nuestras implementaciones de HBS y BTM.

Capítulo 5: Se muestra el esquema SCDAE, las nociones básicas para entenderlo así como su prueba de seguridad.

Capítulo 6: Conclusiones del trabajo realizado y posible trabajo a futuro.

Capítulo 2

Esquemas de Cifrado Autenticable Determinista (CAD)

Los esquemas de Cifrado Autenticable Determinista (CAD) fueron propuestos por Phillip Rogaway y Thomas Eric Shrimpton en el año 2009 en [13] como una posible solución al problema de envoltura de llave. El problema de envoltura de llaves consiste, a grandes rasgos, en cifrar el material de la llave. Las razones para querer hacer esto pueden ser muy variadas, por ejemplo, es posible querer cifrar llaves para su almacenamiento, o tal vez se quiera cifrar una llave de sesión para reducir la cantidad de mensajes cifrados con la llave "principal". Los esquemas de cifrado autenticable determinista proveen dos servicios de seguridad importantes: el servicio de privacidad y el servicio de autenticación. La privacidad es el servicio criptográfico que consiste en impedir que un mensaje X pueda ser interpretado por entes no autorizados, mientras que aquellos entes autorizados deben de ser capaces de interpretar de manera correcta y eficiente dicho mensaje. La autenticación es otro servicio criptográfico el cual consiste, básicamente, en poder comprobar la integridad de los datos cifrados. Cabe destacar que existe otro servicio criptográfico también conocido como autenticación, que consiste en comprobar que alguna información ha sido enviada por algún ente en particular, sin embargo, dado que es posible conseguir la autenticación mediante la integridad de datos se usará el término autenticación para referirse a la integridad.

Dicho problema, el de envoltura de llave, fue propuesto por "The American Standards Committee Working Group X9F1" (El grupo de trabajo X9F1 del comité de estándares americano) en el borrador del estándar conocido como ANS X9.102. Entre los requisitos que este describía se encontraba "la protección de la privacidad y la integridad para datos especializados como es el caso de llaves criptográficas sin el uso de contadores o valores aleatorios". En [13] Rogaway y Shrimpton hacen hincapié en la falta de formalidad matemática del estándar ANS X9.102, por lo que definen un concepto de seguridad para este propósito, así como un nuevo tipo de esquema de cifrado de llave simétrica, los esquemas CAD.

Los CAD reciben su nombre de dos atributos importantes para estos esquemas: Los esquemas CAD permiten la autenticación de los datos y no hacen uso de ningún valor aleatorio o contador, por lo que, a diferencia de otros esquemas, si recibe dos veces la misma entrada, el algoritmo devolverá ambas veces la misma salida, es decir, es un algoritmo determinista.

Si bien los CAD han mostrado ser suficientes para solucionar el problema de envoltura de llaves, también es cierto que estos esquemas pueden ser usados para otras funciones, como por ejemplo el envío de mensajes con información relacionada que no puede ser cifrada. Tal es el caso del envío de paquetes de internet donde las direcciones IP no pueden ser cifradas y deben de viajar de manera clara por la red, de esta forma dichas direcciones funcionarían como cabeceras, mientras que los paquetes de datos si serían cifrados.

Otra aplicación importante de los CAD puede ser el cifrado de discos duros. La restricción del no incremento de la longitud del mensaje de un esquema para cifrar discos puede ser evitada dado que, en la mayoría de las arquitecturas de discos, los sectores de discos tienen más espacio del requerido para almacenar la información.

2.1. Definiciones y notación

El conjunto de todas las cadenas de bits de cualquier longitud será denotados por $\{0,1\}^*$, mientras que el conjunto de todos los vectores de cadenas será denotado por $\{0,1\}^{**}$. El símbolo $\perp$ puede ser leído como *inválido*. Dado un número x, su representación en n bits será denotada por $\underline{x}_n$, por ejemplo $\underline{0}_n$ sería una cadena de n ceros. La concatenación de dos cadenas de bits A y B será denotada por $A||B$. Dadas dos cadenas de bits A y B de la misma longitud, $A\&B$ significa aplicar el operador *and* bit a bit entre ambas cadenas. Dado un conjunto finito de cadenas S, para decir que seleccionamos una cadena x de manera uniformemente aleatoria de S se escribirá $x \xleftarrow{\$} S$.

$\vec{H}$ expresará un elemento de $\{0,1\}^{**}$ y representaría un vector $(H_1, \cdots, H_h)$ donde cada $H_i \in \{0,1\}^*$. Dada una cadena de bits $H \in \{0,1\}^b$, es decir una cadena de b bits, la longitud de dicha cadena se expresará con $|H|$. Dado $H \in \{0,1\}^b$ y $n \in \mathbb{N}$ con $n > 0$ entonces $\eta_n(H) := \lceil b/n \rceil$, es decir, $\eta_n(X)$ es la cantidad de bloques de n bits en que se puede particionar X, el último bloque puede tener menos de n bits.
Dado $\vec{H} \in \{0,1\}^{**}$, denotaremos el numero de componentes de dicho vector con $||\vec{H}||$.
La función $\mathsf{MSB}_\ell(X)$ devolverá los primeros ℓ bits de X, mientras que la función $\mathsf{END}_\ell(X)$ regresará como resultado los últimos ℓ bits de X. Dados dos números naturales n, m la expresión $n \mid m$ quiere decir que m es múltiplo de n, mientras que $n \nmid m$ quiere decir que no existe número natural tal que, al ser multiplicado por n, obtengamos como resultado m. En este documento trataremos el conjunto $\{0,1\}^n$ como el campo GF(2^n). Los elementos de GF(2^n) se pueden ver como polinomios de grado, a lo más, $n-1$, es decir, polinomios de la forma $a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$, donde todos los a_i son elementos de GF(2), o dicho de otra forma, son 0 o 1. Las operaciones de campo consisten en operaciones sobre polinomios, la adición es una suma polinomial ordinaria donde los coeficientes son sumados módulo 2, así que, dados $x_1, x_2 \in \{0,1\}^n$, $x_1 \oplus x_2$ expresará la suma en GF(2^n). La multiplicación es calcula-

da como una multiplicación de polinomios, seguida por una reducción del resultado módulo un polinomio irreducible de grado n. El elemento 10 es un elemento en $\{0,1\}^n$ que será denotado por 2. Para $x \in \mathrm{GF}(2^n)$, por $2x$ queremos decir la multiplicación de x y el elemento especial 2.

Un oráculo $\mathcal{O}$ es un algoritmo que es tratado como caja negra, es decir se desconoce el funcionamiento interno de este, pero es posible interactuar con él, dada una entrada válida nos devuelve la salida correspondiente. Si un algoritmo $\mathcal{A}$ tiene acceso a un oráculo $\mathcal{O}$ es denotado por $\mathcal{A}^{\mathcal{O}}$.

Definiremos al conjunto de todas las funciones con dominio X y codominio Y como $\mathrm{FUNC}(X,Y)$. Si $X = \{0,1\}^x$ y $Y = \{0,1\}^y$ entonces también denotaremos a $\mathrm{FUNC}(X,Y)$ como $\mathrm{FUNC}(x,y)$

Una función pseudoaleatoria (fpa) f es una familia de funciones eficientemente computables $f : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^y$, donde $\mathcal{K}$ es el conjunto de llaves, tales que si $k \xleftarrow{\$} \mathcal{K}$ entonces la probabilidad de distinguir entre $f(K,\cdot)$ y $g \xleftarrow{\$} \mathrm{Func}(\{0,1\}^n, \{0,1\}^y)$ es bastante pequeña.

2.2. Definición de los esquemas CAD

Un esquema CAD es una tupla $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, donde $\mathcal{K}$ es el algoritmo de generación de llaves, el algoritmo usualmente consiste en seleccionar de forma aleatoria un elemento $K \in \mathcal{K}$, por lo que se hará un poco de abuso de notación al utilizar el símbolo $\mathcal{K}$ tanto para denotar la función para elegir dicha llave como para denotar al conjunto de llaves válidas $\mathcal{K} \subset \{0,1\}^*$. El algoritmo de cifrado $\mathcal{E}$ es un algoritmo determinista $\mathcal{E} : \mathcal{K} \times \{0,1\}^{**} \times \{0,1\}^* \to \{0,1\}^*$. El algoritmo de descifrado $\mathcal{D}$ también es un algoritmo determinista tal que $\mathcal{D} : \mathcal{K} \times \{0,1\}^{**} \times \{0,1\}^* \to \{0,1\}^*$. Por simplicidad se utilizará la notación $\mathcal{E}_K(H,X)$ para $\mathcal{E}(K,H,X)$ y $\mathcal{D}_K(H,Y)$ para $\mathcal{D}(K,H,Y)$. Llamaremos al espacio $\mathcal{H} \subset \{0,1\}^{**}$ el espacio de cabeceras. El espacio de mensajes será definido como $\mathcal{X} \subset \{0,1\}^*$ tal que $\mathcal{E}_K(H,X) \in \{0,1\}^*$ si y

sólo si $X \in \mathcal{X}$, $H \in \mathcal{H}$ y $K \in \mathcal{K}$. El espacio de mensajes cifrados será definido como $\mathcal{C} = \{\mathcal{E}_K(H, X) : K \in \mathcal{K}, H \in \mathcal{H}, X \in \mathcal{X}\}$, se requiere que $\mathcal{D}_K(H, Y) = X$ si $\mathcal{E}_K(H, X) = Y$ y $\mathcal{D}_K(H, X) = \perp$ en otro caso. Se da por hecho que para cualquier $K \in \mathcal{K}, H \in \mathcal{H}$ y $X \in \mathcal{X}$, $|\mathcal{E}_K(H, X)| = |X| + e(H, X)$ donde e es una función $e : \{0,1\}^{**} \times \{0,1\}^{*} \to \mathbb{N}$ y e depende únicamente de la cantidad de componentes de H, de la longitud de estos y de la longitud de X, e será llamada la función de expansión de Π [13].

2.2.1. Seguridad en un esquema CAD

Para definir la seguridad de un esquema CAD es necesario definir primero el concepto de adversario.

De manera informal definiremos a un adversario como cualquier algoritmo que intente romper el criptosistema. Mientras que un adversario eficiente será un algoritmo probabilista que se ejecuta en tiempo polinomial.

Para definir la seguridad de un criptosistema se necesitan describir los objetivos y recursos de un adversario. Un adversario puede tener varios objetivos, el más fuerte y difícil de conseguir consiste en recuperar la llave del esquema utilizado, con el conocimiento de la llave el adversario puede descifrar todos los mensajes cifrados que circulan por un canal de comunicación público y también puede reemplazar los mensajes cifrados con mensajes de su elección. Pero el problema de la recuperación de la llave es un objetivo muy difícil y aún sin la recuperación de la llave un adversario puede descifrar o cifrar ciertos mensajes. Un objetivo más fácil de conseguir sería el de la distinción de mensajes, el cual básicamente consiste en distinguir los mensajes cifrados por el esquema atacado de cadenas aleatorias de bits. Para acotar los objetivos y capacidades del adversario se dará una definición seguridad de un CAD.

Definición 1. Dado un esquema $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ con espacio de cabeceras $\mathcal{H}$, espacio de mensajes $\mathcal{X}$, espacio de llaves $\mathcal{K}$ y función de expansión e. La ventaja de un

adversario $\mathcal{A}$ en romper el esquema Π es definida como

$$\mathbf{Ven}_{\Pi}^{CAD}(\mathcal{A}) = |\Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathcal{A}^{\$(\cdot,\cdot),\perp(\cdot,\cdot)} \Rightarrow 1]|.$$

Esto es, la ventaja del adversario $\mathcal{A}$ al atacar el esquema Π como un CAD ($\mathbf{Ven}_{\Pi}^{CAD}(\mathcal{A})$) es igual a la diferencia de las probabilidades de dos eventos: En el primer evento se seleccionará una llave de forma aleatoria del conjunto de llaves ($K \xleftarrow{\$} \mathcal{K}$) y se le concederá a $\mathcal{A}$ acceso a los algoritmos de cifrado ($\mathcal{E}_K$) y descifrado ($\mathcal{D}_K$); en el segundo evento al adversario $\mathcal{A}$ se le concederá acceso a los oráculos aleatorio $\$(\cdot,\cdot)$ e inválido $\perp(\cdot,\cdot)$.

Cuando se realiza una consulta al oráculo aleatorio $\$(\cdot,\cdot)$ con los valores $H \in \mathcal{H}$ y $X \in \mathcal{X}$ este regresa una cadena aleatoria de longitud $|X| + e(H,X)$. Si el adversario realiza alguna consulta fuera del dominio de $\mathcal{E}$ el oráculo regresa el valor $\perp$. El oráculo $\perp(\cdot,\cdot)$ regresa el símbolo $\perp$ ante cualquier consulta sin importar la entrada. Se asume que el adversario no preguntará (H,Y) al segundo oráculo si previamente se ha consultado al primer oráculo con (H,X) y este ha devuelto Y, también se asume que el adversario no realizará consultas al primer oráculo fuera de $H \times X$ y que no repetirá una consulta. Las últimas suposiciones se hacen sin pérdida de generalidad, mientras que la primera suposición es para prevenir victorias triviales por parte del adversario.

El esquema Π se considera seguro si para todo adversario eficiente $\mathcal{A}$ la ventaja $\mathbf{Ven}_{\Pi}^{CAD}(\mathcal{A})$ es pequeña.

Rogaway y Shrimpton mostraron en [13] que probar la seguridad de un esquema CAD Π, es equivalente a probar que dicho esquema es seguro en términos de privacidad determinista y autenticación determinista.

La ventaja en términos de privacidad determinista de un adversario $\mathcal{A}$ sobre un esquema CAD $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, el cual tiene espacio de cabeceras $\mathcal{H}$ y espacio de mensajes $\mathcal{X}$, esta definida como:

$$\mathbf{Ven}_{\Pi}^{detPriv}(\mathcal{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot)} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\$(\cdot,\cdot)} \Rightarrow 1\right] \tag{2.1}$$

Se supone que el adversario $\mathcal{A}$ no repite ninguna consulta. De manera informal el adversario intenta distinguir entre el oráculo de cifrado y el oráculo aleatorio, por lo que el repetir una consulta le daría una victoria trivial.

La ventaja de un adversario $\mathcal{A}$ sobre un esquema CAD $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, con espacio de cabeceras $\mathcal{H}$ y espacio de mensajes $\mathcal{X}$, en términos de un de autenticación determinista esta definida como:

$$\mathbf{Ven}_{\Pi}^{detAuth}(\mathcal{A}) = \Pr\left[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot),\mathcal{D}_K(\cdot,\cdot)} \text{ falsifica}\right] \tag{2.2}$$

Cuando se menciona que $\mathcal{A}$ *falsifica* quiere decir que realiza una consulta a su oráculo derecho (H, Y) y recibe una salida distinta a $\perp$. Se da por hecho, sin pérdida de generalidad, que $\mathcal{A}$ nunca realiza la consulta (H, Y) a su oráculo derecho si previamente $\mathcal{A}$ ha realizado la consulta (H, X) a su oráculo izquierdo y este ha devuelto Y.

2.3. Esquemas CAD existentes

Los esquemas CAD actuales siguen la propuesta del esquema general propuesto por P. Rogaway y T. Shrimpton [13], este esquema general divide el CAD en dos partes: Un esquema simétrico de solo privacidad basado en vectores iniciales Ψ, y una función pseudo aleatoria (fpa) F con la cual se generan los valores iniciales para Ψ. La construcción general se puede observar en la Fig. 2.1.

A continuación se muestran los esquemas CAD que actualmente se han publicado.

2.3.1. SIV: Synthetic IV

Este esquema fue propuesto por P. Rogaway, T. Shrimpton en el mes de agosto del año 2007 [13]. En la Fig. 2.2 se puede ver el funcionamiento general del modo

STDAE.$\mathcal{E}_K(H, X)$	STDAE.$\mathcal{D}_K(H$,MAC$, C)$
1. MAC$\leftarrow F(H, X)$	1. $X \leftarrow \Psi^{-1}$(MAC$, C)$
2. $C \leftarrow \Psi$(MAC$, X)$	2. mac$\leftarrow F(H, X)$
3. **return** MAC, C	3. **if** mac $\neq$MAC **then**
	4. **return** $\perp$
	5. **else**
	6. **return** X
	7. **end if**

Figura 2.1: Algoritmos de cifrado y descifrado de una construcción estándar CAD.

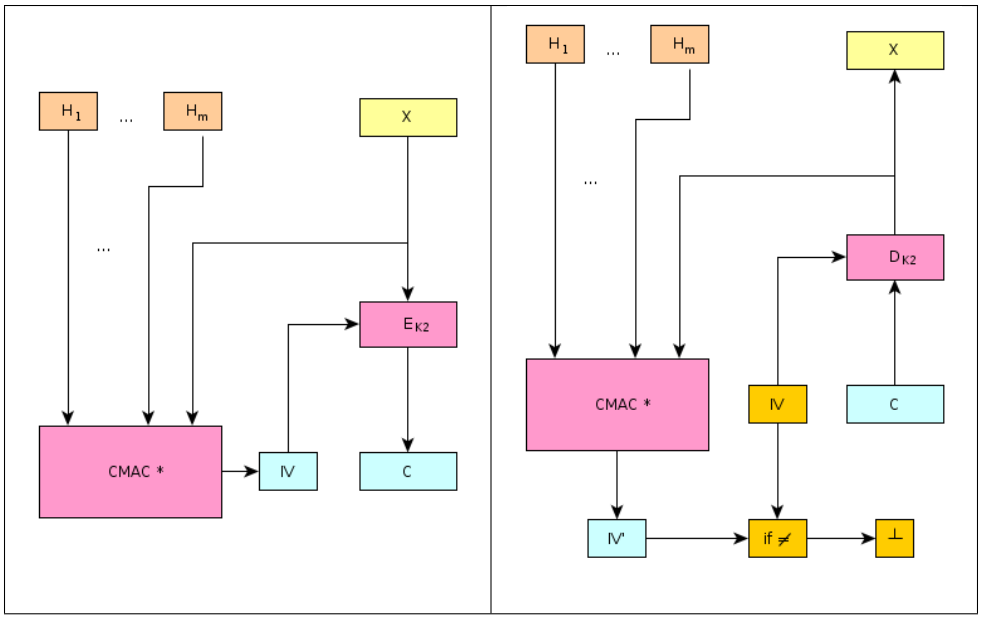

Figura 2.2: Diagramas de cifrado (izquierda) y descifrado (derecha) del esquema SIV.

$SIV.\mathcal{E}_{K1,K2}(H_1, ..., H_h, X)$

1. $IV \leftarrow \mathsf{CMAC}^*_{K1}(H_1, ..., H_h, X)$
2. $C \leftarrow \mathsf{CTR}_{K2}(IV, X)$
3. **return** $Y \leftarrow IV||C$

Figura 2.3: $SIV.\mathcal{E}_{K1,K2}(H_1, ..., H_t, X)$

$SIV.\mathcal{D}_{K1,K2}(H_1, ..., H_h, Y)$

1. **if** $|Y| < n$ **then**
2. **return** $\perp$
3. **end if**
4. $IV \leftarrow \mathsf{MSB}_n(Y)$
5. $C \leftarrow \mathsf{END}_{n+1-|Y|}(Y)$
6. $X \leftarrow \mathsf{CTR}_{K2}(IV, C)$
7. $IV' \leftarrow \mathsf{CMAC}^*_{K1}(H_1, ..., H_h, X)$
8. **if** $IV = IV'$ **then**
9. **return** X
10. **else** X
11. **return** $\perp$
12. **end if**

Figura 2.4: $SIV.\mathcal{D}^{H_1,...,H_t}_{K1,K2}(Y)$

de operación SIV, mientras que su algoritmo de cifrado se puede ver en la Fig. 2.3 y su algoritmo de descifrado se puede ver en la Fig. 2.4. SIV utiliza como esquema de cifrado de solo privacidad a una variación del modo contador y como una fpa utiliza un algoritmo llamado CMAC^*, el cual será descrito a continuación.

CMAC^*

Para poder definir CMAC^* primero debemos definir el algoritmo CMAC.

El algoritmo CMAC genera un mensaje de autenticación de una cadena X. Para construir CMAC se necesita un cifrador por bloques E cuyo tamaño de bloque es n. Los parámetros de entrada de CMAC son: Una llave K y un mensaje X, el cual debe de ser dividido en bloques de tamaño n, es decir, $X = X_{[1]}||X_{[2]}||...||X_{[\ell]}$ donde cada $|X_{[i]}| = n$ para todo $1 \leq i \leq \ell - 1$ y $|X_{[\ell]}| \leq n$. Un valor precalculado importante para CMAC es R_n, para conseguirlo es necesario escoger el primer polinomio p, lexicográficamente hablando, del conjunto de polinomios irreducibles de grado n con la menor cantidad de unos posible, R_n son los coeficientes del polinomio p sin considerar al del elemento x^n. El algoritmo CMAC se muestra en la Fig. 2.5.

$\mathsf{CMAC}_K(X)$	$\mathsf{keygen}(K)$
0. $\ell = \eta_n(X)$	1. $L \leftarrow E_K(\underline{0}_n)$
1. $K1, K2 \leftarrow \mathsf{keygen}(K)$	2. **if** $\mathsf{MSB}_1(L) = 0$ **then**
2. **if** $\lvert X_{[\ell]}\rvert = n$ **then**	3. $\quad K1 \leftarrow L << 1$
3. $\quad X_{[\ell]} \leftarrow K1 \oplus X_{[\ell]}$	4. **else**
4. **else**	5. $\quad K1 = (L << 1) \oplus R_n$
5. $\quad X_{[\ell]} \leftarrow K2 \oplus (X_{[\ell]} \lvert\lvert 10^{\lvert n - \lvert X_{[\ell]}\rvert - 1\rvert})$	6. **end if**
6. **end if**	7. **if** $\mathsf{MSB}_1(K1) = 0$ **then**
7. $C_{[0]} \leftarrow \underline{0}_n$	8. $\quad K2 \leftarrow K1 << 1$
8. **for** $i \leftarrow 1$ **to** ℓ **do**	9. **else**
9. $C_{[i]} \leftarrow E_K(C_{[i-1]}, M_{[i]})$	10. $\quad K2 \leftarrow (K1 << 1) \oplus R_n$
10. **end for**	11. **end if**
11. **return** $C_{[\ell]}$	12. **return** $K1, K2$

Figura 2.5: Algoritmo CMAC

La función CMAC^* es descrita en la Fig. 2.6.

Modo contador de SIV

El modo contador de SIV esta definido como:

$$\mathsf{CTR}_K(IV, X) = X_{[1]} \oplus E_K(ctr) \lvert\lvert X_{[2]} \oplus E_K(\mathsf{INCR}(ctr)) \lvert\lvert ... \lvert\lvert X_h \oplus \mathsf{MSB}_{\lvert X_h \rvert}(E_K(\mathsf{INCR}^{h-1}(ctr)))$$

donde $ctr = IV \& 1^{n-64}01^{31}01^{31}$ y $h = \eta_n(X)$. El algoritmo para calcular CTR se muestra en la Fig. 2.7. La función incremento es definida como $\mathsf{INCR}(S) = S + 1$ mód 2^n y la notación $\mathsf{INCR}^p(S)$ significa aplicar el operador $\mathsf{INCR}(\cdot)$ p veces sobre S.

Algoritmo de descifrado

El algoritmo de descifrado recibe como parámetros un código MAC IV, un mensaje cifrado X y una cabecera $\vec{H}$, este hace uso de los mismos algoritmos que el algoritmo de cifrado, sin embargo, primero utiliza el modo contador con el MAC IV como vector de inicialización para descifrar el mensaje, para después calcular un nuevo vector de inicialización IV' el cual será comparado con el vector IV, en caso de que $IV = IV'$

$\mathsf{CMAC}^*_K(H_1, H_2, \cdots, H_h)$

1. $S \leftarrow \mathsf{CMAC}_K(\underline{0}_n)$
2. **for** $i \leftarrow 1$ **to** $h-1$ **do**
3. $\quad S \leftarrow 2S \oplus \mathsf{CMAC}_K(H_i)$
9. **end for**
4. **if** $|H_h| \geq n$ **then**
5. $\quad$ **return** $\mathsf{CMAC}_K(S \oplus \mathsf{END}_n(H_h))$
6. **else**
7. $\quad$ **return** $\mathsf{CMAC}_K(2S \oplus H_h || 10^{n-|H_h|-1})$
8. **end if**

Figura 2.6: Algoritmo CMAC^*

$\mathsf{CTR}_K(IV, X)$

1. $Ctr \leftarrow IV || 1^{n-64} 01^{31} 01^{31}$
2. $Pad \leftarrow E_K(Ctr) || E_K(Ctr+1) || E_K(Ctr+2)...$
3. **return** $C \leftarrow X \oplus \mathsf{MSB}_{|X|}(Pad)$

Figura 2.7: $\mathsf{CTR}_K(IV, X)$

se devuelve el texto descifrado, en otro caso devuelve $\perp$.

2.3.2. HBS: Hash Block Stealing

El HBS es presentado por Tetsu Iwata y Kan Yasuda en [14]. La principal ventaja que este esquema presentaba, en comparación con el SIV, es el uso de una sola llave de n bits. Sin embargo este esquema está diseñado para trabajar con encabezados de una sola cadena, además de que este esquema requiere de la implementación de la función inversa del cifrador por bloques. Este esquema utiliza una evaluación polinomial como principal componente de su fpa, la cual es usada poder obtener tanto la MAC como el vector inicial del modo contador.

Para poder describir la fpa de HBS es necesario definir primero la función $\mathsf{pad}_{\mathsf{hbs}}$ y el valor c_Y. La función $\mathsf{pad}_{\mathsf{hbs}}$ estará definida de la siguiente forma:

$$\mathsf{pad}_{\mathsf{hbs}}(Y) = \begin{cases} Y & \text{si } |Y| = 0 \bmod n \\ Y||10^r & \text{en otro caso} \end{cases}$$

Donde r cumple con $|Y| + 1 + r = 0 \bmod n$. Mientras que el valor c_Y será definido de la siguiente forma:

$$c_Y = \begin{cases} \underline{2}_n & \text{si } |Y| = 0 \quad \text{mód } \ n \\ \underline{1}_n & \text{en otro caso} \end{cases}$$

Ahora, si $L \in \{0,1\}^n$ y $Z = \mathsf{pad}_{\mathsf{hbs}}(Y)$ donde $Z = Z_{[0]}||Z_{[1]}||...||Z_{z-1}$ y $|Z_{[i]}| = n$ para $0 \leq i \leq z-1$ entonces definiremos $f_L(Y)$ como:

$$f_L(Y) = c_Y L(L^z \oplus L^{z-1} Z_{[0]} \oplus ... \oplus L Z_{[z-2]} \oplus Z_{[z-1]})^2$$

Finalmente definiremos a la función picadillo del esquema HSB como $F_L^{(2)}(H, M) = f_L(H) \oplus f_L(M)$. El algoritmo para calcularlo se puede ver en la Fig. 2.8, donde:

$F_L^{(2)}(H, M)$

1. $z_H \leftarrow L \oplus H_{[0]}$
2. **for** $i = 1$ **to** $\eta_n(H) - 1$ **do**
3. $\quad z_H \leftarrow L \cdot z_H \oplus H_{[i]}$
4. **end for**
5. $z_M \leftarrow L \oplus M_{[0]}$
6. **for** $i = 1$ **to** $\eta_n(M) - 1$**do**
7. $\quad z_M \leftarrow L \cdot z_M \oplus M_{[i]}$
8. **end for**
9. **return** $L \cdot z_H^2 \cdot g_0 \oplus (L \cdot z_M)2 \cdot g_1$

Figura 2.8: $F_L^{(2)}(H, M)$

$$g_0 = \begin{cases} \underline{1}_n & \text{si } n \nmid |H| \\ \underline{2}_n & \text{en otro caso} \end{cases}$$

$$g_1 = \begin{cases} \underline{1}_n & \text{si } n \nmid |M| \\ \underline{2}_n & \text{en otro caso} \end{cases}$$

El modo contador de HBS esta definido como:

$$\mathsf{CTR}_K(S, M) = M_{[1]} \oplus E_K(\mathsf{INCR}^0(S))||...||M_{[m]} \oplus \mathsf{MSB}_{|M_{[m]}|}(E_K(\mathsf{INCR}^{m-1}(S)))$$

donde, $m = \eta_n(M)$ y la función incremento es definida como $\mathsf{INCR}^i(S) = S \oplus \underline{j_i}_n$ donde $j_i = i + 1 \mod 2^n$. El algoritmo para calcular $\mathsf{CTR}_K(S, M)$ se muestra en la Fig. 2.9.

Como se puede ver en la Fig. 2.10 el algoritmo de cifrado de HBS genera la llave L de la función picadillo $F_L^{(2)}$ al cifrar con la llave K la cadena $\underline{0}_n$, después calcula la función $F_L^{(2)}$ con la cabecera y el mensaje como entradas. El resultado S de $F_L^{(2)}(H, M)$ se utiliza como vector de inicialización del modo contador, se cifra el mensaje con dicho modo contador. Finalmente se calcula el valor $T = E_K(S)$ y regresa tanto T como el mensaje cifrado como resultados.

$\mathsf{CTR}_K(S, M)$

1. **for** $i \leftarrow 0$ **to** $\lceil |M|/n \rceil - 1$ **do**
2. $\quad R_i \leftarrow E_K(S \oplus \underline{j_i}_n)$
3. **end for**
4. $R \leftarrow R_0 || R_1 || \cdots || R_{\lceil |M|/n \rceil - 1}$
5. $C \leftarrow M \oplus \mathsf{MSB}_{|M|}(R)$
6. **return** C

Figura 2.9: $\mathsf{CTR}_K(S, M)$

$\text{HBS.Enc}_K(H, M)$

1. $L \leftarrow E_K(0^n)$
2. $S \leftarrow F_L^{(2)}(H, M)$
3. $C \leftarrow \mathsf{CTR}_K(S, M)$
4. $T \leftarrow E_K(S)$
5. **return** (T, C)

Figura 2.10: $\text{HBS.Enc}_K(H, M)$

La Fig. 2.11 es el algoritmo de descifrado de HBS, este recupera los valores S y L utilizados en el algoritmo de cifrado. Para recuperar L se cifra la cadena $\underline{0}_n$ con el cifrador de bloques y la llave compartida. El valor S se obtiene fácilmente al hacer uso de la función inversa E_K^{-1} del cifrador por bloques E. El modo contador al recibir como parámetros H y C regresará una cadena M', la función $F_L^{(2)}(H, M')$ regresará una cadena S' , en caso de que $S = S'$ la función HBS.Dec regresará M' o $\perp$ en otro caso:

2.3.3. BTM: Bivariate Tag Mixing

El algoritmo BTM es presentado por Tetsu Iwata y Kan Yasuda en [15]. Este esquema hace uso de una sola llave, sin embargo, a diferencia de HBS, este esquema no requiere de la función inversa del cifrador de bloques y es capaz de aceptar vectores de cadenas como cabecera [15].

La Fig. 2.12 es el algoritmo descrito por Iwata y Yasuda para obtener el vector de

HBS.Dec$_K(H,(T,C))$

1. $L \leftarrow E_K(0_n)$
2. $S \leftarrow E_K^{-1}(T)$
3. $M \leftarrow \mathsf{CTR}_K(S,C)$
4. $S' \leftarrow F_L^{(2)}(H,M)$
5. **if** $S = S'$ **then**
6. **return** $\perp$
7. **else**
8. **return** M
9. **end if**

Figura 2.11: HBS.Dec$_K(H,(T,C))$

inicialización del modo contador utilizado. Este algoritmo evalúa múltiples polinomios sobre dos variables, de ahí el nombre de *Bivariate*. Se realiza una evaluación polinomial sobre la llave L del mensaje a cifrar y de cada cadena del vector de cabeceras, después, cada uno de estos resultados son utilizados en una segunda evaluación polinomial pero esta vez sobre la llave U. La Fig. 2.13 muestra la forma en que estos polinomios son evaluados. La función f utilizada para BTM es la siguiente:

Sea $X = X_{[0]}||\cdots||X_{[m-1]}$ donde $|X_{[i]}| = n$ para $0 \leq i < m-1$ y $|X_{[m-1]}| \leq n$. Entonces definiremos a f como

$$f_L(X) = \delta(X_{[|X|-1]}) \cdot (L^m \oplus L^{m-1} \cdot X_{[0]} \oplus \cdots \oplus L \cdot X_{[m-2]} \oplus \pi(X_{[m-1]}))$$

donde:

$$\delta(x) = \begin{cases} \underline{1}_n & \text{si } 0 \leq |x| \leq n-1 \\ \underline{2}_n & \text{si } |x| = n \end{cases}$$

$$\pi(x) = \begin{cases} X||1||0^{n-1-|x|} & \text{si } 0 \leq |x| \leq n-1 \\ X & \text{si } |x| = n \end{cases}$$

Sea $U, L \in \{0,1\}^n$ y $\vec{H} = [H_1, H_2, \cdots, H_h]$ y $M \in \{0,1\}^*$. Entonces definiremos

$F_{L,U}(\vec{H}, M)$

1. $S \leftarrow \underline{0}_n$
2. **for** $i = 0$ **to** $||\vec{H}|| - 1$ **do**
3. $\quad f \leftarrow \underline{1}_n$
4. $\quad$ **for** $j = 0$ **to** $\eta_n(H_i) - 2$ **do**
5. $\quad\quad f \leftarrow (f \cdot L) \oplus H_{i[j]}$
6. $\quad$ **end for**
7. $\quad f \leftarrow (f \cdot L) \oplus \pi(H_{i[\eta_n(H_i)-1]})$
8. $\quad f \leftarrow f \cdot \delta(H_{i[\eta_n(H_i)-1]})$
9. $\quad f \leftarrow f \cdot U$
10. $\quad S \leftarrow f \oplus S$
11. **end for**
12. $f \leftarrow \underline{1}_n$
13. **for** $j = 0$ **to** $\eta_n(M) - 2$ **do**
14. $\quad f \leftarrow (f \cdot L) \oplus M_{i[j]})$
15. **end for**
16. $f \leftarrow (f \cdot L) \oplus \pi(M_{[\eta_n(M)-1]})$
17. $f \leftarrow f \cdot \delta(M_{[\eta_n(M)-1]})$
18. $S \leftarrow S \oplus f$
19. **return** $T \leftarrow E_K(S)$

Figura 2.12: $F_{L,U}(\vec{H}, M)$

la función $F_{U,L}$ como:

$$F_{U,L}(\vec{H}, M) = U^h f_L(H_0) \oplus U^{h-1} f_L(H_1) \oplus ... \oplus U f_L(H_{||\vec{H}||-1}) \oplus f_L(M)$$

Dados $U, L, \vec{H}$ y M el procedimiento para computar $F_{U,L}(\vec{H}, M)$ es mostrado en la Fig. 2.12. El procedimiento es mostrado gráficamente en la Fig. 2.13.

$\mathsf{CTR}_K^N(X)$

1. $x \leftarrow |X|/n$
2. **for** $i \leftarrow 0$ **to** $\eta_n(X) - 1$ **do**
3. $\quad R_{[i]} \leftarrow E_K(N \boxplus \underline{i}_n)$
4. **end for**
5. $R \leftarrow R_{[0]} \cdots R_{[\eta_n(X)-1]}$
6. $Y \leftarrow X \oplus \mathsf{MSB}_{|X|}(R)$
7. **return** Y

Figura 2.14: $\mathsf{CTR}_K^N(X)$

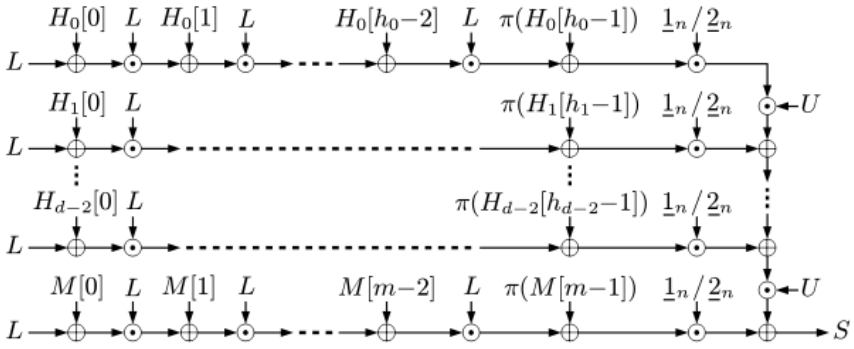

Figura 2.13: Diagrama de la función $F_{U,L}(\vec{H}, M)$ de BTM, esta función es utilizada para calcular el vector de inicialización de BTM. Imagen tomada de [15].

El algoritmo mostrado en la Fig. 2.14 es el modo contador de BTM, el cual está definido de la misma forma que el modo contador de HBS, sin embargo, la función de incremento INCR de BTM es distinta. INCR hace uso del operador tagmixing ($\boxplus$), el cual se define de la siguiente forma:

$$A \boxplus B = (A_H + B_H)||(A_L + B_L)$$

donde $A_H||A_L = A$, $B_H||B_L = B$ y $|A_H| = |A_L| = |B_H| = |B_L|$ el operador $+$ representa una suma entera módulo $n/2$.

El algoritmo de cifrado, el cual se puede ver en la Fig. 2.15, consiste en generar las llaves L y U al cifrar $\underline{0}_n$ y $\underline{1}_n$ respectivamente. Tras esto se evalúa la función

BTM.Enc$_K(\vec{H}, M)$
1. $L \leftarrow E_K(\underline{0}_n)$
2. $U \leftarrow E_K(\underline{1}_n)$
3. $T \leftarrow F_{L,U}(\vec{H}, M)$
4. $C \leftarrow \mathsf{CTR}_K^{T \boxplus U}(M)$
5. **return** (T, C)

Figura 2.15: BTM.Enc$_K(H, M)$

BTM.Dec$_K(\vec{H}, (T, C))$
1. $L \leftarrow E_K(\underline{0}_n)$
2. $U \leftarrow E_K(\underline{1}_n)$
3. $M \leftarrow \mathsf{CTR}_K^{T \boxplus U}(C)$
4. $T' \leftarrow F_{L,U}(\vec{H}, M)$
5. **if** $T \neq T'$ **then**
6. $\quad M \leftarrow \perp$
7. **end if**
8. **return** M

Figura 2.16: BTM.Dec$_K(H, (T, C))$

$F_{U,L}(\vec{H}, M)$ obteniendo como resultado T, el cual es utilizado para generar el vector de inicialización $(U \boxplus T)$ del modo contador, así como para ser usado como MAC. El modo contador, como ya se había mencionado, utiliza como vector de inicialización el resultado de aplicar el tagmixing a U y a T, el modo contador $\mathsf{CTR}_K^{T \boxplus U}(M)$ regresa el mensaje cifrado que será entregado junto con T como resultado.

El algoritmo de descifrado, comienza generando las llaves L y U de la misma forma que se hizo en el algoritmo de cifrado, Una vez con dichas llaves se aplica la función $\mathsf{CTR}_K^{T \boxplus U}(H, C)$ obteniendo como resultado M', se ejecuta $F_{U,L}(\vec{H}, M')$ para obtener el valor T', finalmente se comparan los valores T y T' y se devuelve M' en caso de que sean iguales o $\perp$ en otro caso:

2.4. Sumario

Este capítulo se enfoca en los esquemas CAD, su origen, funcionamiento, definiciones y aplicaciones. También se da una descripción de los esquemas CAD existentes.

La primera parte del capítulo da un vistazo a los esquemas CAD, las razones de su creación y su nombre, también se describen brevemente sus posibles aplicaciones.

En la segunda parte se definen conceptos importantes para el correcto entendimiento de un CAD, como lo son el espacio de mensajes, el espacio de cabeceras, los términos adversario, ventaja y falsificación entre otros.

La tercera parte, con ayuda de las definiciones anteriores, nos permite definir de manera formal un CAD y la ventaja que un adversario tiene contra un esquema de este tipo.

La última parte de este capítulo es la descripción de los esquemas CAD existentes: SIV, HBS y BTM. Se da una descripción de cada uno de estos esquemas, de sus algoritmos de cifrado y descifrado.

Capítulo 3

Bloques Básicos para la Construcción de los Modos de Operación CAD

En este capítulo se mostrará la estructura y diseño de los principales componentes para la implementación de los esquemas CAD: un multiplicador Karatsuba de cuatro estados en tuberia, un cifrador por bloques AES de 10 estados en tubería y un cifrador inverso (descifrador) AES. Estos bloques fueron usados para la implementación de los modos de operación CAD que se muestran en el siguiente capítulo. El diseño de estos bloques está enfocado a la velocidad, por lo que se dejó el consumo de recursos en segundo plano.

Se ha elegido AES como cifrador por bloques debido a que es el cifrador por bloques estándar, además se eligió una longitud de llave de 128 bits debido a que la implementación de éste sería suficiente para mostrar los resultados deseados. Existen muchas implementaciones disponibles en la literatura, como lo son [21], [22], [23], [24]. La implementación de AES está basada en [25] , la principal diferencia entre la implementación de [25] y esta implementación, es que nuestra implementación en lugar de ser secuencial, es una implementación en tubería. En [25], se tiene solamente una ronda de AES por lo que se obtiene un bloque cifrado cada 10 ciclos de reloj, mientras que en nuestra implementación se presenta un esquema de tubería en 10 etapas, donde cada etapa es una ronda de AES. Este esquema se comporta de manera

similar al anterior en los primeros diez ciclos de reloj, pero a partir del onceavo ciclo se obtiene un bloque cifrado cada ciclo de reloj.

Si se observan los algoritmos 2.10 y 2.15 se puede observar que una operación necesaria e importante para la implementación de dichos esquemas es la multiplicación en el campo GF(2^{128}), si bien estas multiplicaciones se podrían hacer de la forma tradicional, se realizan mediante el método de Karatsuba, el cual es un método de multiplicación más eficiente, como se muestra en [26]. Por otro lado Mancillas et al. mostraron en [27] una implementación del método Karatsuba para multiplicaciones de cadenas de 128 bits a un costo computacional subcuadrático con frecuencias cercanas a los 100 Mhz, sin embargo, en el siguiente trabajo se espera obtener frecuencias cercanas a los 300 Mhz mediante el uso de un esquema en tubería de 4 etapas.

3.1. Funcionamiento general de AES

AES es la especificación del cifrador por bloques estándar del NIST (Instituto Nacional de Estándares y Tecnologías) de los Estados Unidos y fue propuesto por Joan Daemen y Vincent Rijmen en [28]. AES recibe dos entradas, un bloque de datos a cifrar de 128 bits y una llave, la cual puede ser de 128, 192 o 256 bits, dependiendo del grado de seguridad que se desea obtener. Como salida AES regresará un bloque cifrado de 128 bits.

Los algoritmos de cifrado y descifrado de AES consisten en una aplicación sucesiva de transformaciones, éstas pueden ser organizadas en rondas, cada una de las cuales hace uso de una llave de ronda que debe de ser creada mediante el algoritmo de generación de llaves. El número de rondas establecido puede ser de 10, 12 o 14; dependiendo de si la llave es de 128, 192 o 256 bits respectivamente.

El resultado de aplicar una transformación en un resultado intermedio se le llamará *estado*. Tanto los estados, como la llave y las llave de ronda pueden visualizarse como matrices de bytes de 4 filas de largo y 4 columnas de alto. La manera en que esto se consigue se muestra en la Fig. 3.1.

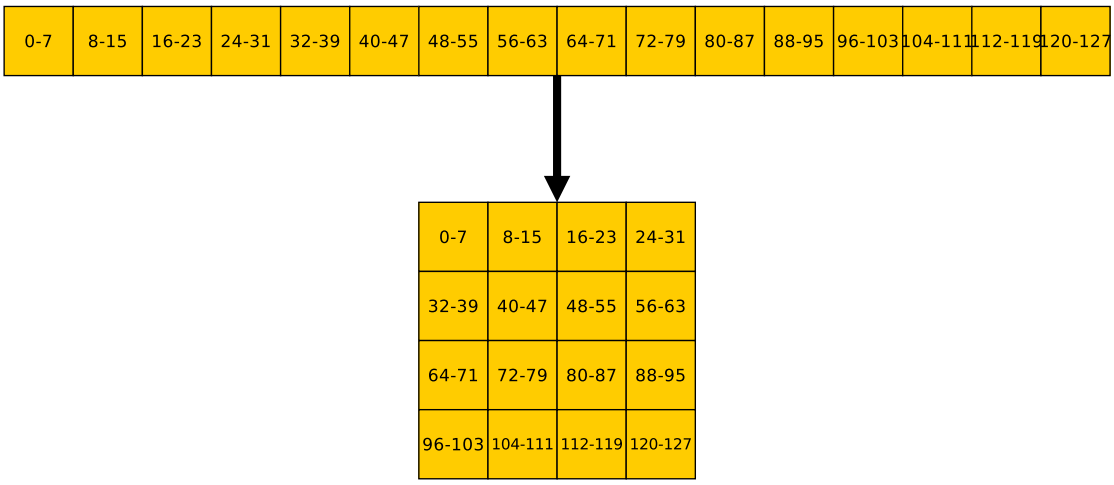

Figura 3.1: Visualización de un vector de 128 bits en un estado. Los números indican la posición de los bits.

3.1.1. Rondas de AES

La transformaciones que contiene cada ronda de AES depende de si se esta cifrando o descifrando y de si se trata de la última ronda o no. Cada ronda esta compuesta por las siguientes transformaciones.

- Sustitución de bytes (byte substitution)
- Corrimiento de filas (shift rows)
- Mezclado de columnas (mix columns)
- Adición de llave de ronda (add round key)

La última ronda es idéntica a las primeras 9 a excepción de que ésta no hace uso del mezclado de columnas.

Sustitución de Bytes: ByteSub

La transformación ByteSub es una sustitución no lineal de bytes, que opera en cada uno de los bytes del estado de manera independiente. Dicha sustitución es realizada mediante el uso de una tabla de sustitución la cual es llamada caja-S. La caja-S es invertible y está construida con la composición de dos transformaciones:

- Primero se toma el inverso multiplicativo en GF(2^8) del byte a transformar.

- Entonces, se aplica la transformación afín (sobre GF(2^8)) definida por

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

La aplicación de esta sustitución de bits sobre todos los bytes del estado, es denotada por:

ByteSub(estado)

La tabla de sustitución inversa es construida por la composición de las siguientes dos transformaciones:

- Se aplica la transformación afín definida por

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

- Se toma el inverso multiplicativo en GF(2^8) del resultado.

La aplicación de esta sustitución de bits sobre todos los bytes del estado, se denota por:

InvByteSub(estado)

Corrimiento de filas

En el corrimiento de filas, cada fila del estado es desplazada una cantidad distinta de lugares. La fila 0 no es desplazada, la fila 1 es desplazada 1 posición (8 bits) hacia la izquierda, la fila 2 y la fila 3 son desplazada 2 (16 bits) y 3 (24 bits) posiciones respectivamente. Dichos desplazamientos son cíclicos, es decir, el bloque 0 de la fila 1 terminará en la posición 4 de la fila 1 al realizar un desplazamiento a la izquierda. Esta transformación se puede ver en la Fig. 3.2

El realizar el corrimiento de filas en un estado dado es denotado por:

ShiftRow(estado)

El corrimiento inverso de filas consiste en realizar estos desplazamientos cíclicos hacia la derecha y es denotado por :

InvShiftRows(estado)

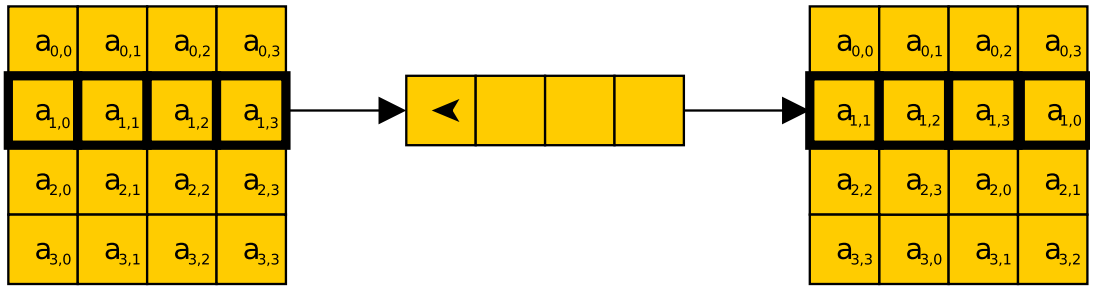

Figura 3.2: Corrimiento de filas.

Mezclado de columnas

En el mezclado de columnas, las columnas del estado son consideradas polinomios dentro de GF(2^8) y multiplicadas modulo $x^4 + 1$ con el polinomio:

$$c(x) = \texttt{0x03} \cdot x^3 + \texttt{0x01} \cdot x^2 + \texttt{0x01} \cdot x + \texttt{0x02}$$

este polinomio es coprimo con $4x+1$ y por lo tanto es invertible. Esto se puede escribir como una multiplicación de matrices. Es decir $b(x) = c(x) \cdot a(x)$ puede ser visto como la multiplicación de matrices:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} \texttt{0x02} & \texttt{0x03} & \texttt{0x01} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x02} & \texttt{0x03} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x01} & \texttt{0x02} & \texttt{0x03} \\ \texttt{0x03} & \texttt{0x01} & \texttt{0x01} & \texttt{0x02} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

La aplicación de esta operación sobre todas las columnas del estado será denotado por: Mixcolumn(estado) La operación inversa de mixcolumn consiste en multiplicar el estado por el polinomio $d(x)$ definido como :

$$d(x) = \texttt{0x0B} \cdot x^3 + \texttt{0x0D} \cdot x^2 + \texttt{0x09} \cdot x + \texttt{0x0E}$$

O visto de otra forma, multiplicar cada columna del estado por la matriz:

$$\begin{bmatrix} 0x0E & 0x0B & 0x0D & 0x09 \\ 0x09 & 0x0E & 0x0B & 0x0D \\ 0x0D & 0x09 & 0x0E & 0x0B \\ 0x0B & 0x0D & 0x09 & 0x0E \end{bmatrix}$$

Cabe destacar que es común que dicha matriz sea descompuesta en una multiplicación de las siguientes dos matrices:

$$\begin{bmatrix} 0x0E & 0x0B & 0x0D & 0x09 \\ 0x09 & 0x0E & 0x0B & 0x0D \\ 0x0D & 0x09 & 0x0E & 0x0B \\ 0x0B & 0x0D & 0x09 & 0x0E \end{bmatrix} = \begin{bmatrix} 0x02 & 0x03 & 0x01 & 0x01 \\ 0x01 & 0x02 & 0x03 & 0x01 \\ 0x01 & 0x01 & 0x02 & 0x03 \\ 0x03 & 0x01 & 0x01 & 0x02 \end{bmatrix} \begin{bmatrix} 0x05 & 0x00 & 0x04 & 0x00 \\ 0x00 & 0x05 & 0x00 & 0x04 \\ 0x04 & 0x00 & 0x05 & 0x00 \\ 0x00 & 0x04 & 0x00 & 0x05 \end{bmatrix}$$

De tal forma que la matriz utilizada durante el cifrado pueda ser utilizada durante el descifrado. El mezclado de columnas inverso aplicado a todas las columnas del estado será denotado por:

InvMixColumns(estado)

Adición de llave de ronda

Este paso consiste en realizar una operación XOR entre el estado actual y la llave de ronda.

En esta operación, una llave de ronda es aplicada al estado por la aplicación bit a bit de la operación XOR. La llave de ronda es derivada de la llave del cifrador mediante la generación de llaves. La llave de ronda tiene la misma longitud que la longitud del bloque. La transformación que consiste en aplicar la operación XOR entre la llave de ronda y el estado es denotada por:

AddRoundKey(estado,llaveDeRonda)

Esta transformación es ilustrada en la Fig. 3.3.

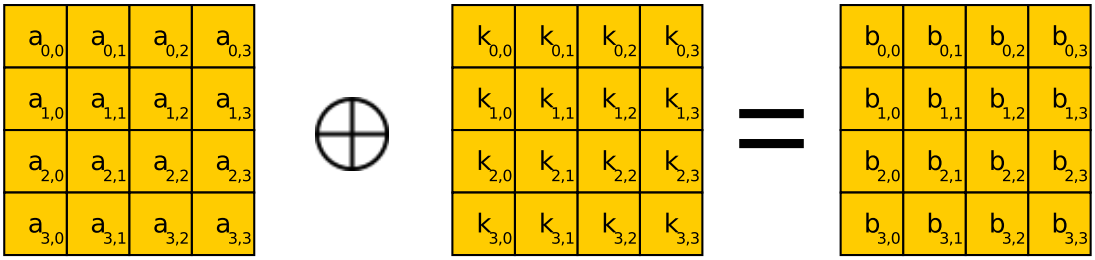

Figura 3.3: Adición de la llave de ronda.

AddRoundKey es su propia inversa.

3.1.2. Generación de llaves de ronda

La generación de llave de ronda consiste de dos componentes: La expansión de la llave y la selección de llave de ronda. La expansión de llave consiste a su vez en la generación de una llave expandida, dicha llave expandida tendrá una longitud igual a la longitud del bloque por la cantidad de rondas más uno (es decir, para bloques de 128 bits y 10 rondas, el tamaño de la llave expandida será de 1408 bits). Una vez que se tiene la llave expandida, si el tamaño de bloque es nb, la llave de la primera ronda serán los primeros nb bits, la llave de la segunda ronda serán los siguientes nb bits y así consecutivamente.

Expansión de la llave

La llave de cifrado original consiste en 128 bits acomodados como una matriz de bytes de 4×4 y nombraremos sus columnas $w[0], w[1], w[2]$ y $w[3]$. A partir de estas columnas es posible generar otras 40 columnas mediante el uso de la siguiente ecuación:

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

donde $T(w[i-1])$ se calcula de la siguiente forma. Sean x_1, x_2, x_3 y x_4 los bytes que componen $x[i]$:

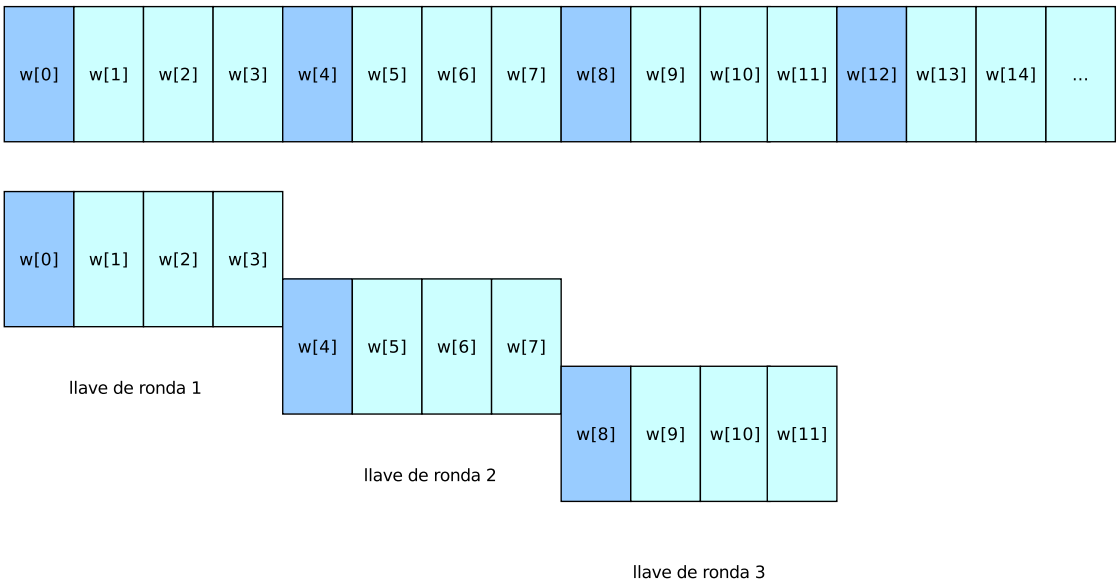

Figura 3.4: Adición de la llave de ronda.

- Se realiza un corrimiento cíclico de bytes, para que el orden de los bytes sea: x_2, x_3, x_4, x_1
- Se remplaza cada byte por el byte correspondiente obtenido de la caja-S: $S(x_2)$, $S(x_3)$, $S(x_4)$, $S(x_1)$.
- Se calcula la constante de ronda $r_{con}(i) = 02^{(i-4)/4}$ en GF(2^8).

Finalmente $T(w[i-1])$ es el vector columna $(S(x_2) \oplus r_{con}(i), S(x_3), S(x_4), S(x_1))$.

La llave de ronda i se obtiene al seleccionar las palabras $W[Nb*i]$ a $W[Nb*(i+1)]$ de la llave extendida. Esto es ilustrado en la Fig. 3.4.

3.2. Implementación de AES

Para implementar HBS y BTM se necesitan los algoritmos de cifrado y descifrado de AES. Tanto HBS como BTM hacen uso del algoritmo de cifrado intensivamente, por lo que se optará por utilizar el algoritmo de cifrado de AES en tubería. Es posible

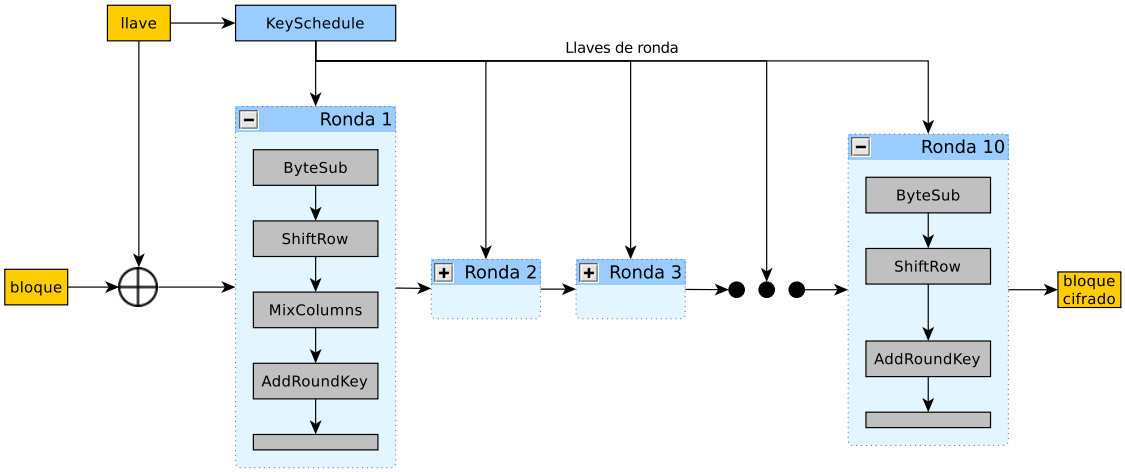

Figura 3.5: AES de llave de 128 bits en pipeline.

implementar únicamente una ronda de AES y utilizarla iterativamente 10 veces, sin embargo este enfoque provocaría que obtuviéramos un bloque cifrado cada 10 ciclos de reloj, por lo que se optó por utilizar 10 instancias de ronda en tubería, de esta forma se obtiene la misma frecuencia, pero se obtiene un mejor desempeño, ya que a partir del décimo ciclo de reloj, se obtendría un bloque cifrado cada ciclo. Dado que el algoritmo de descifrado es utilizado solamente una vez, no se obtendría ningún beneficio de implementarlo con 10 rondas en tubería, por lo que el algoritmo de descifrado tendrá solamente una ronda que será utilizada iterativamente. Las rondas estarán basadas en lo presentado en [29], por lo que se esperan frecuencias cercanas a los 300 MHz, lo cual supera con creces a otras versiones presentadas en otro trabajos como [30].

3.2.1. Implementación de una ronda de AES

Ya se han mencionado previamente los componentes de una ronda de AES, a continuación mencionaremos como se implementaron estos.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Tabla 3.1: Caja S, los valores que aquí se presentan se encuentran en base 16.

Sustitución de bytes: ByteSub

La implementación de la sustitución de bytes consiste en la búsqueda de una tabla (S-BOX). Dicha tabla fue implementada como una memoria ROM y puede verse en la Tabla 3.1

Para sustituir un byte se deben de tomar los primeros 4 bits para determinar la columna y los últimos 4 para determinar la fila.

Corrimiento de filas: ShiftRows

El corrimiento de filas se consigue fácilmente mediante un reordenamiento de las señales.

Mezclado de columnas: MixColumns

Para esta transformación, el vector de bits de la entrada es redireccionado de la siguiente forma:

- La columna uno (c_1) esta compuesta por las palabras compuestas por los bits:
  - del 127 al 120.

  - del 95 al 88.
  - del 63 al 56.
  - del 31 al 24.

- La columna dos (c_2) esta compuesta por los las palabras compuestas por los bits:
  - del 119 al 112.
  - del 87 al 80.
  - del 55 al 48.
  - del 23 al 16.

- La columna tres (c_3) esta compuesta por los las palabras compuestas por los bits:
  - del 111 al 104.
  - del 79 al 72.
  - del 47 al 40.
  - del 15 al 8.

- La columna cuatro (c_4) esta compuesta por los las palabras compuestas por los bits:
  - del 103 al 96.
  - del 71 al 64.
  - del 39 al 32.
  - del 7 al 0.

La transformación MixColumn esta compuesta a su vez de 4 instancias de la función ColMix, cada una de estas instancias recibirá como entrada una de las columnas mencionadas previamente y regresarán como salida la columna correspondiente:

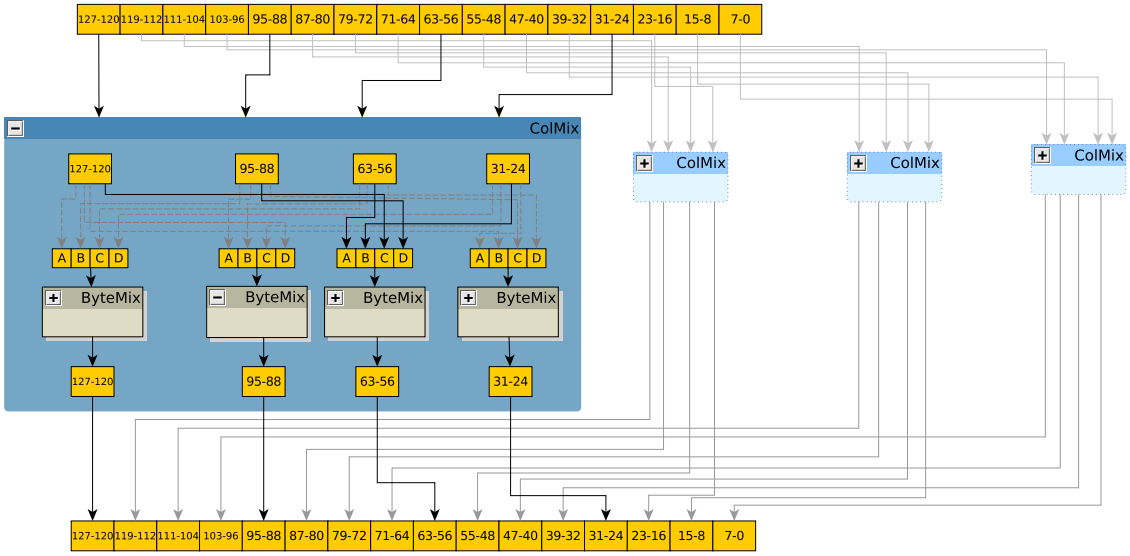

Figura 3.6: Funcionamiento general de Mixcolumn.

$$n_1 = ColMix(c_1)$$

$$n_2 = ColMix(c_2)$$

$$n_3 = ColMix(c_3)$$

$$n_4 = ColMix(c_4)$$

El nuevo vector de bits será construido a partir de las columnas n_1, n_2, n_3 y n_4 de manera similar a como el vector de entrada fue descompuesto en columnas. Este procedimiento se puede ver en la Fig. 3.6.

A su vez, $ColMix$ esta compuesto de 4 instancias del procedimiento llamado $ByteMix$. La entrada de $ColMix$ está compuesta por las 4 palabras de 8 bits U_0, U_1, U_2 y U_3, estas palabras se utilizarán como entradas para cada una de las instancias de de $ByteMix$ de la siguiente forma:

$$S_0 = ByteMix(U_0, U_1, U_2, U_3)$$

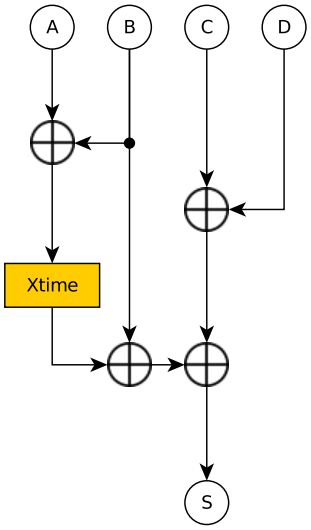

Figura 3.7: Arquitectura de $ByteMix$.

$$S_2 = ByteMix(U_1, U_2, U_3, U_0)$$

$$S_3 = ByteMix(U_2, U_3, U_0, U_1)$$

$$S_4 = ByteMix(U_3, U_0, U_1, U_2)$$

Finalmente la salida de $ColMix$ sera el conjunto de palabras (S_0, S_1, S_2, S_3).

El circuito del procedimiento $ByteMix$ es mostrado en la Fig. 3.7

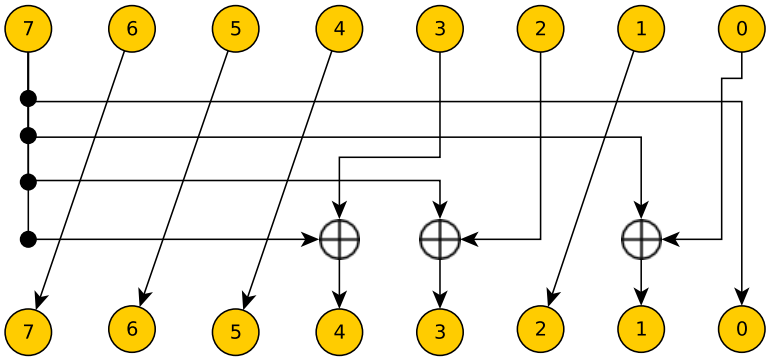

Figura 3.8: Arquitectura de $Xtime$.

3.3. Algoritmo de Karatsuba

El método de Karatsuba permite realizar multiplicaciones de manera más eficiente, en el caso particular de este trabajo de tesis el método de Karatsuba será utilizado para realizar multiplicaciones de polinomios.

La forma más sencilla de explicar el algoritmo de Karatsuba es con un ejemplo: Sean $A = a_1x + a_0$ y $B = b_1x + b_0$ dos polinomios sobre un anillo $\mathbb{R}$. La multiplicación de estos polinomios se realiza usualmente como se muestra en la ecuación (3.1), sin embargo es posible realizar la misma operación de forma más eficiente:

$$\begin{aligned} AB &= (a_1x + a_0)(b_1x + b_0) \\ &= a_1b_1x^2 + [a_0b_1 + a_1b_0]x + a_0b_0 \qquad (3.1) \\ &= a_0b_0(1 + x) + a_1b_1(x^2 + x) + (a_0 - a_1)(b_1 - b_0)x \qquad (3.2) \end{aligned}$$

Como se puede observar en la ecuación (3.2) la cantidad de multiplicaciones realizadas es menor, un total de 3 (no se consideran las multiplicaciones por la raíz x ni sus potencias), en comparación del método original (3.1), que hace un total de 4.

El método Karatsuba-Ofman se puede generalizar. Sea $Y = x^n$, $A = (a_1Y + a_0)$ y $B = (b_1Y + b_0)$ donde cada a_i y b_i son polinomios de hasta grado $n - 1$, entonces la multiplicación de estos polinomios sería

$$AB = a_0b_0[1 + Y] + a_1b_1[Y^2 + Y] + (a_0 - a_1)(b_1 - b_0)Y$$

después cada uno de los elementos a_0b_0,a_1b_1 y $(a_0 - a_1)(b_1 - b_0)$ puede ser calculado de manera recursiva con el método Karatsuba-Ofman.

3.3.1. Reducción modular

Cuando se realiza una multiplicación de dos elementos de n bits, el resultado suele ser un elemento de $2n$ bits, por lo que es necesario realizar una reducción modular.

La reducción modular se realizará base un polinomio irreducible de grado n, en este trabajo se utiliza el polinomio $p(x) = x^{128} + x^7 + x^2 + x + 1$.

El proceso de reducción modular sería el siguiente, dado $A \in \{0,1\}^{2n}$ la reducción de A utilizando el polinomio $p(x)$ sería:

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

donde $A_{(i,j)}$ es la cadena de bits resultante de únicamente considerar desde el bit j hasta el bit i de A. Mientras que $A_{(i)}$ es el bit i de A.

3.4. Implementación del multiplicador Karatsuba y Ofman

Como ya se he mencionado, el tamaño de los bloques es de 128 bits, debido a esto, nuestro multiplicador Karatsuba de 128 bits divide cada dato en dos de 64 bits, los cuales serán divididos en datos de 32 bits, y así sucesivamente hasta obtener multiplicaciones de 4 bits, las cuales serán efectuadas de manera tradicional. La forma

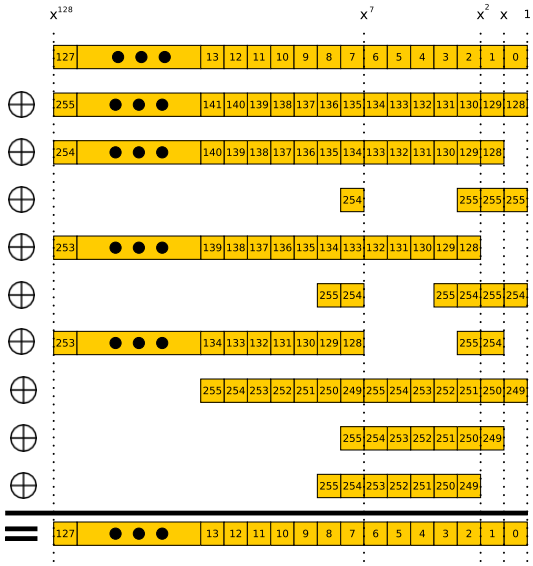

Figura 3.9: Reducción modular.

en que esto se realizó se puede ver en la Fig. 3.10.

Una parte importante del diseño es la ubicación de los latch, estos permiten dividir la multiplicación en operaciones más pequeñas que pueden realizarse más rápidamente, permitiendo a su vez que todo el esquema funcione de manera más eficiente.

3.5. Sumario

En este capítulo se muestra a fondo el diseño de los componentes principales de nuestras implementaciones: El AES de llave de 128 bits con diez estados en tubería y el multiplicador Karatsuba de 128 bits con cuatro estados en tubería.

La primera parte de este capítulo se enfoca en nuestra implementación de AES, se da una explicación breve de la especificación de AES-128, de manera general se describe que dicha especificación puede ser visualizada como la creación de 10 llaves de ronda y una secuencia de 10 pasos o 10 rondas.

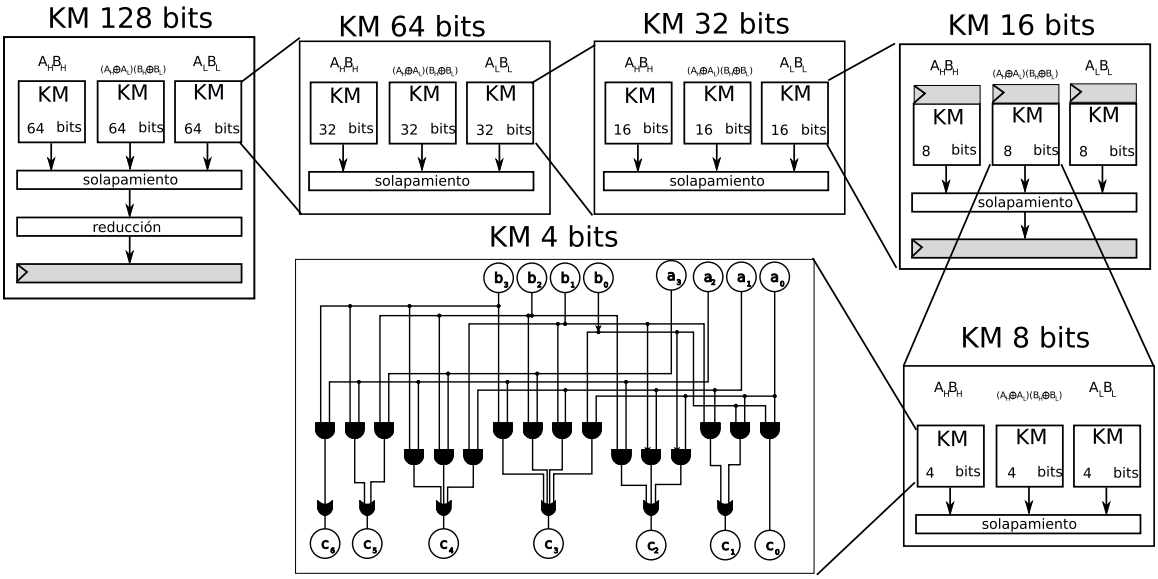

Figura 3.10: Multiplicador Karatsuba de 128 bits.

A continuación se detallan las operaciones que componen cada una de las rondas de cifrado: Sustitución de bytes (ByteSub), corrimiento de filas (ShiftRows), mezclado de columnas (MixColumns) y adición de llave de ronda (AddRoundKey). También se detallan las operaciones necesarias para la función de descifrado de AES: Sustitución de bytes inversa (InvByteSub), corrimiento de filas inversa (InvShiftRows), mezclado de columnas inverso (InvMixColumns).

La segunda parte del capítulo trata sobre la implementación del multiplicador Karatsuba de cuatro estados en tubería. Primero se explica el método Karatsuba, después se describe la reducción modular, la cual es una operación necesaria para que la multiplicación sea cerrada en GF(2^{128}). Finalmente se describe como se implementó el multiplicador Karatsuba para elementos en GF(2^{128}) de tal manera que este cumpliera con los criterios de desempeño que se esperaban.

Capítulo 4

Implementación de los modos BTM y HBS

En este capítulo se muestran la implementación y detalle del diseño de los esquemas HBS y BTM. Para la implementación de estos esquemas se hace uso de los componentes detallados en el capítulo 3.

4.1. Consideraciones de diseño

Para poder realizar los esquemas HBS y BTM se tomaron las siguientes consideraciones.

- La longitud de los mensajes a cifrar será de 256 bloques y cada bloque tendrá una longitud de 128 bits (4096 bytes en total).
- La longitud de las cabeceras será de 1 bloque (128 bits) para HBS y BTM, además, se realizará una versión de BTM con la longitud de cabecera variable.
- Se supondrá que los datos de entrada serán proporcionados por un circuito independiente, el cual se sincronizará con estas implementaciones mediante señales generadas por el circuito de control.
- La validación de los resultados se hizo con vectores de prueba generados con Maple.

- El dispositivo sobre el que se realizarán las implementaciones será un Virtex 5 XC5VLX85.

4.2. Implementación de operaciones básicas

Algunas operaciones básicas son necesarias para la implementación de los modos de operación BTM y HBS.

4.2.1. Xtimes

La operación xtimes consiste en multiplicar un polinomio por la raíz x, dado que en nuestro caso dicha raíz es 2, la operación básicamente es realizar un desplazamiento de bits a la izquierda y aplicar una reducción en caso de ser necesario.

Supóngase que $A \in \{0,1\}^{128}$ es un arreglo de bits, la expresión A_α denota el bit α menos significativo de A, mientras que la expresión $A_{(\alpha,\beta)}$ denota el vector de bits desde el bit menos significativo β hasta el bit menos significativo α.

Debido a nuestro polinomio $p(x) = x^{128} + x^7 + x^2 + x + 1$ la reducción consistirá básicamente en las siguientes asignaciones:

$$\begin{aligned} R_{(0)} &= A_{(127)} \\ R_{(1)} &= A_{(0)} \text{ xor } A_{(127)} \\ R_{(2)} &= A_{(1)} \text{ xor } A_{(127)} \\ R_{(6,3)} &= A_{(5,2)} \\ R_{(7)} &= A_{(6)} \text{ xor } A_{(127)} \\ R_{(127,8)} &= A_{(126,7)} \end{aligned}$$

4.2.2. Elevar al cuadrado

El elevar al cuadrado cuando se visualizan los elementos de $\mathsf{GF}(2^{128})$ como polinomios es fácil y de bajo coste. Dado un polinomio $\sum_{i=0}^{127} x^i \ell_i$ donde ℓ_i es el i-ésimo coeficiente del polinomio $L(x)$, el cuadrado de este polinomio sería:

$$[L(x)]^2 = \sum_{i=0}^{127} x^{2i} \ell_i$$

Una vez calculado el cuadrado del polinomio, es necesario realizar una reducción modular en base a polinomio $p(x) = x^{128} + x^7 + x^2 + x + 1$, esta reducción puede verse en el capítulo 3 en la sección respectiva al algoritmo de Karatsuba(en la Pag. 59).

4.3. Evaluación Polinomial

Ambos esquemas utilizan evaluaciones polinomiales para obtener el vector de inicialización del modo de operación de solo privacidad, dichas evaluaciones realizan una cantidad considerable de multiplicaciones de elementos de GF(2^{128}), por lo que se debe de aprovechar el multiplicador Karatsuba de 4 estados en tubería el mayor tiempo posible. Para conseguirlo, es necesario dividir cada evaluación polinomial de una variabe en 4 evaluaciones distintas, tales que, al ser sumadas obtengan el mismo resultado.

Sea $X = X_{[0]}||X_{[1]}||...||X_{[x-1]}$ una cadena de bits tal que cada $X_i \in$GF(2^n) y sea $L \in$GF(2^n). La evaluación polinomial de X con respecto de L se expresaría de la siguiente forma

$$\begin{aligned} P_L(X) &= L^x \oplus L^{x-1} X_{[0]} \oplus ... \oplus L X_{[x-2]} \oplus X_{[x-1]} \\ &= L^x \oplus \sum_{i=0}^{x-1} L^i X_{[x-i-1]} \\ &= L^x \oplus p1 \oplus p2 \oplus p3 \oplus p4 \end{aligned} \tag{4.1}$$

donde $\oplus$ es la operación de adición definida para el campo GF(2^n) y $p1, p2, p3$ y $p4$ son

$$\begin{aligned} p1 &= \sum_{i=1}^{\frac{x}{4}} (L^4)^{\frac{x}{4}-i} X_{[4i-1]} \\ p2 &= L\sum_{i=1}^{\frac{x}{4}} (L^4)^{\frac{x}{4}-i} X_{[4i-2]} \\ p3 &= L^2\sum_{i=1}^{\frac{x}{4}} (L^4)^{\frac{x}{4}-i} X_{[4i-3]} \\ p4 &= L^3\sum_{i=1}^{\frac{x}{4}} (L^4)^{\frac{x}{4}-i} X_{[4i-4]} \end{aligned} \tag{4.2}$$

De esta forma se puede ver a la evaluación polinomial como la suma de cuatro evaluaciones polinomiales distintas y la suma del elemento L^x.

Cada una de estas evaluaciones polinomiales es una evaluación polinomial con respecto de L^4, por lo que es posible realizar dichas evaluaciones haciendo uso de la regla de Horner.

Una vez que se comience a calcular el polinomio $p4$ el resultado tardará 4 ciclos en obtenerse, por lo que durante el ciclo 2 se podrá comenzar a calcular el polinomio $p3$, cuyo resultado se obtendrá en 4 ciclos, de la misma forma se comenzarán a calcular los polinomios $p2$ y $p1$, para el cuarto ciclo se obtendrá el resultado de la primera multiplicación de $p4$ por lo que se podrá seguir con su evaluación, lo mismo sucederá en el ciclo 5 con el polinomio $p3$ y así consecutivamente cada ciclo hasta haber realizado las cuatro evaluaciones polinomiales necesarias. De esta forma se estarán aprovechando los cuatro estados del multiplicador el mayor tiempo posible.

4.4. Implementación de HBS

La implementación del algoritmo de cifrado de HBS puede ser dividida en cuatro partes, a continuación se muestran dichas partes y su correspondencia con la Fig. 4.1:

HBS.Enc$_K(H, M)$

1. $L \leftarrow E_K(0^n)$
2. $L^2 \leftarrow \mathsf{squaring}(L)$
3. $L^3 \leftarrow L^2 * L$
4. $L^4 \leftarrow \mathsf{squaring}(L^2)$
5. $S \leftarrow F^{(2)}(L, L^2, L^3, L^4, H, M)$
6. $C \leftarrow \mathsf{CTR}_K(S, M)$
7. $T \leftarrow E_K(S)$
8. **return** (T, C)

Figura 4.1: HBS.Enc$_K(H, M)$

HBS.Dec$_K(H, (T, C))$

1. $L \leftarrow E_K(0_n)$
2. $L^2 \leftarrow \mathsf{squaring}(L)$
3. $L^3 \leftarrow L^2 * L$
4. $L^4 \leftarrow \mathsf{squaring}(L^2)$
5. $S \leftarrow E_K^{-1}(T)$
6. $M \leftarrow \mathsf{CTR}_K(S, C)$
7. $S \leftarrow F^{(2)}(L, L^2, L^3, L^4, H, M)$
8. **if** $S = S'$ **then**
9. **return** $\perp$
10. **else**
11. **return** M
12. **end if**

Figura 4.2: HBS.Dec$_K(H, (T, C))$

1. La parte de inicialización donde se generan los valores necesarios para todo el proceso (renglones 1 a 4).

2. La evaluación polinomial de la cabecera y del mensaje (renglón 5).

3. El uso del modo contador para cifrar los mensajes (renglón 6).

4. Entrega del código de autenticación (renglón 7).

En el caso del algoritmo de descifrado, también puede ser dividido en cuatro partes, las cuales se muestran a continuación junto con su correspondencia con el algoritmo 4.2:

1. La parte de inicialización, donde se generan los valores necesarios para todo el proceso (renglones 1 a 5).

2. El uso del modo contador para descifrar los bloques del mensaje (renglón 6).

3. La evaluación polinomial de la cabecera y del mensaje (renglón 7).

4. La validación de los datos (renglones 8 a 12).

Cabe destacar que durante el algoritmo de descifrado, se realiza el modo contador (para descifrar los bloques del mensaje) y la evaluación polinomial al mismo tiempo, la única razón por la que se consideran procesos separados, es para puntualizar que los procesos de control internos de estas etapas (los cuales son descritos más adelante) son los mismos a los utilizados en el cifrado, la diferencia reside en el circuito de control general.

Dado que la longitud del mensaje y de la cabecera son fijos, es posible determinar qué operación debería de estar realizando un circuito en un ciclo de reloj dado, por lo que el control general del circuito está basado en un contador ctr.

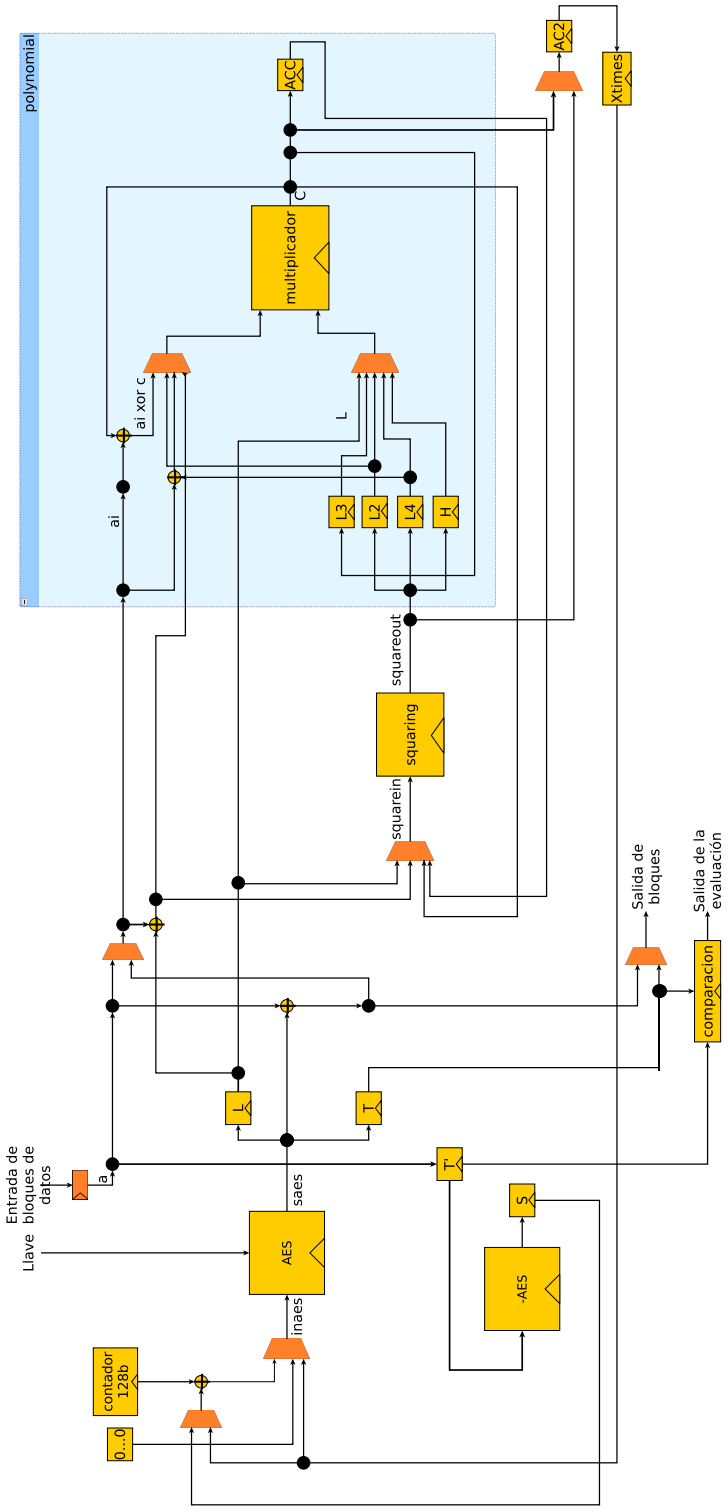

Figura 4.3: Arquitectura desarrollada para HBS.

4.4.1. Inicialización de HBS

La inicialización de los procesos de cifrado y descifrado son similares ya que ambos requieren de el cálculo de los valores L, L^2, L^3 y L^4. La principal diferencia entre la inicialización del proceso de cifrado y de descifrado es la obtención del vector inicial del modo contador en el proceso de descifrado. Este paso se lleva a cabo en la linea 2 del algoritmo 4.2 al descifrar el MAC . El diagrama de estados de esta parte del algoritmo se muestra en la Fig. 4.4, mientras que el significado de cada estado se muestra en la tabla 4.4.1

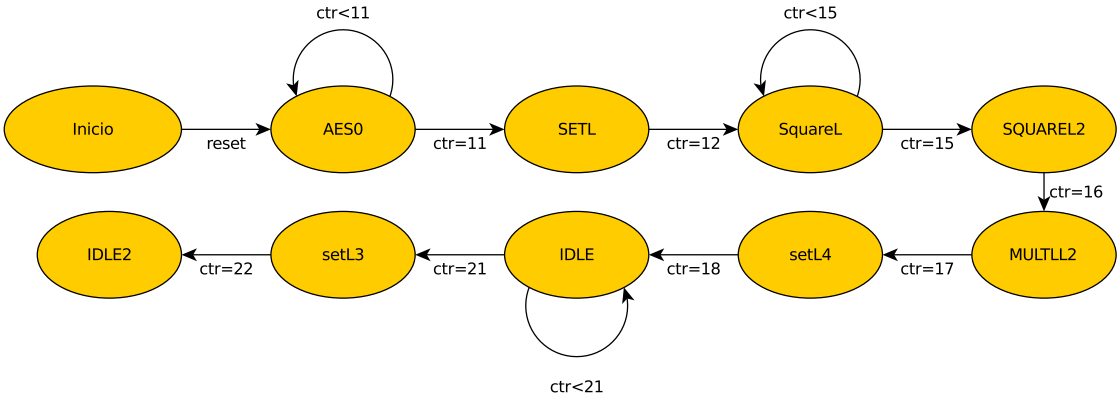

Figura 4.4: Diagrama de estados de la generación de variables de HBS.

4.4.2. Función picadillo de cabeceras de un solo bloque

La función picadillo de cabeceras de un solo bloque se realiza de la siguiente forma: La cabecera H está compuesta únicamente por un bloque H de 128 bits, el mensaje está compuesto por el vector de bloques $X = (X_{[0]}, X_{[1]}, \cdots, X_{[255]})$. La evaluación polinomial de H y X sería:

$$\begin{aligned} F^{(2)}(L, L^2, L^3, L^4, H, X) &= (L \oplus H)^2 \cdot L \cdot 2 \oplus (L^{257} \oplus L^{256} \cdot X_{[0]} \oplus \cdots \oplus L \cdot X_{[255]})^2 \cdot 2 \\ &= ((L \oplus H)^2 \cdot L \oplus (L^{257} \oplus L^{256} \cdot X_{[0]} \oplus \cdots \oplus L \cdot X_{[255]})^2) \cdot 2 \end{aligned}$$

Estado	Significado
AES0	Se asigna el valor $\underline{0}_{128}$ a la señal 'inaes'.
SETL	Se asigna el valor de la señal 'saes' al registro 'Lout'.
SquareL	Se asigna el valor de salida del registro 'L' a 'insquaring' (la señal de entrada de la función para elevar al cuadrado squaring).
SquareL2	Se asigna el valor de la señal 'squareout' (la salida de la función squaring) a la señal 'insquaring'. Se asigna el valor de la señal 'squareout' a el registro 'L2'.
MULTLL2	Se asignan los valores de las señales de salida de los registros 'L2' y 'L' a las señales de entrada del multiplicador Karatsuba.
setL4	Se asigna el valor de la señal 'squareout' el registro 'L4'.
IDLE	Se espera por un resultado válido del multiplicador.
setL3	Se asigna el valor de la señal 'c' al registro 'L3'.
IDLE2	Continúan las demás partes del algoritmo de cifrado.

Tabla 4.1: Significado de cada estado de la Fig. 4.4.

El proceso para realizar esta evaluación polinomial se describe en el algoritmo 4.5

Las señales correspondientes a estas operaciones pueden ser vistas en la Fig. 4.6.

La implementación de HBS considera únicamente mensajes de 4096 bytes, por lo que cada mensaje X está compuesto por 256 bloques ($X = X_{[0]}||X_{[1]}||...||X_{[255]}$) de 128 bits, La evaluación polinomial de este mensaje se puede realizar de manera similar a la mostrada en 4.1. Sin embargo, dado que ésta es multiplicada después por L, podemos visualizarla como la evaluación de un polinomio en L de grado 256 sin

$F^{(2)}(L, L^2, L^3, L^4, H, X)$
1. LxorH $\leftarrow L \oplus H$
2. SLxorH $\leftarrow$ squaring(LxorH)
3. headerHash $\leftarrow$ SLxorH $\cdot\, L$
4. msgHash $\leftarrow$ eval(L, L^2, L^3, L^4, X)
5. msgSqrt $\leftarrow$ squaring(msgHash)
6. sumHash $\leftarrow$ msgSqrt $\oplus$ headerHash
7. hash $\leftarrow$ xtimes(sumHash)

Figura 4.5: $F^{(2)}(L, L^2, L^3, L^4, H, X)$. La correspondencia entre este algoritmo y la arquitectura final se puede observar en Fig. 4.6

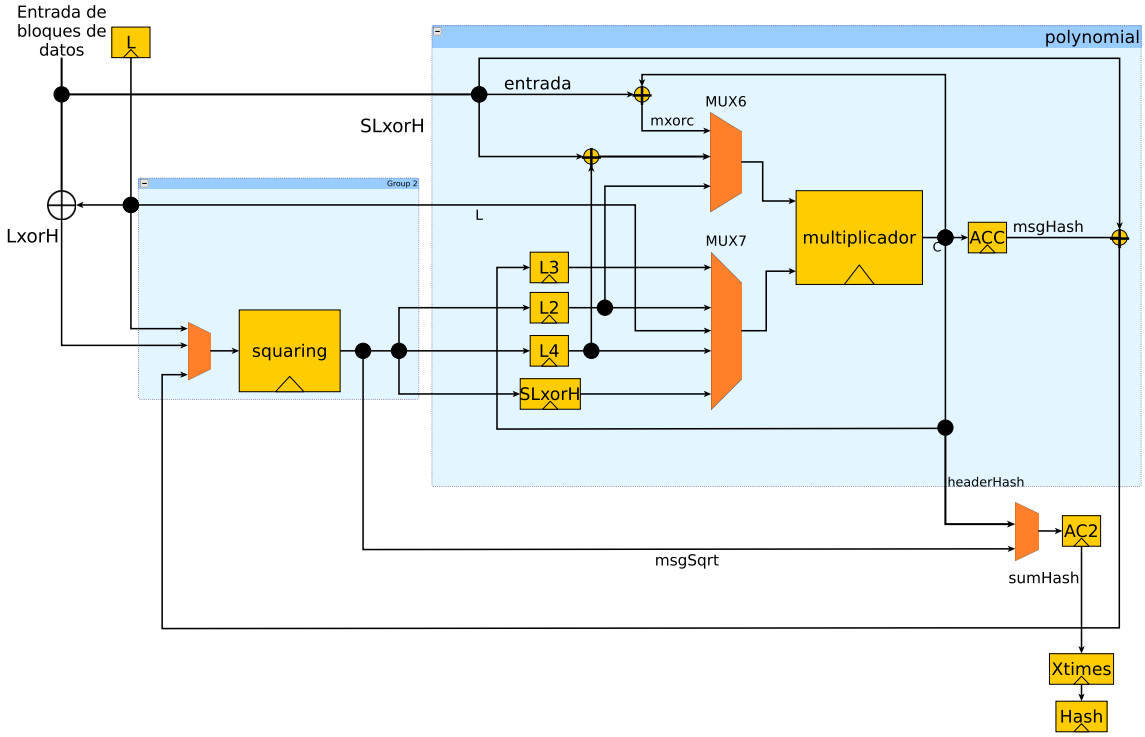

Figura 4.6: Arquitectura necesaria para el hash de HBS

término independiente.

La evaluación polinomial del mensaje puede ser dividida en 4 partes:

Inicial Consiste en los primeros cuatro ciclos de reloj. En los primeros tres ciclos se inicia el cálculo de los valores $L^4X_{[0]}$, $L^4X_{[1]}$ y $L^4X_{[2]}$, obteniendo los respectivos resultados en los ciclos 4, 5 y 6. En el ciclo 4 se calcula el valor $L^4(L^4 \oplus X_{[3]})$.

Media Esta fase comprende desde el ciclo 5 hasta el ciclo 252, donde las señales de entrada del multiplicador se fijan en L^4 y $X_{[i-1]} \oplus c$, donde c es la señal de salida del multiplicador, de esta forma se calcula la mayor parte de la evaluación polinomial.

Final En esta fase se calcula el valor final de cada parte del polinomio. En los ciclos 253, 254 y 255 se obtienen los resultados de las partes en las que se ha dividido el polinomio, cada una de ellas es multiplicada por L^3,L^2 y L respectivamente. Es por esto que las señales de entrada al multiplicador son $X_{[i-1]} \oplus c$ y L^3 en el ciclo 253, L^2 en el ciclo 254 y L en el ciclo 255.

Suma Finalmente se agregan los resultados de las 4 partes del polinomio al acumulador en los ciclos 257, 258, 259 y 260.

Una vez que se ha evaluado el polinomio del mensaje, podemos proceder al paso 5 del algoritmo 4.5. Para esto se eleva al cuadrado el resultado de la evaluación polinomial del mensaje (ciclo 261-263).

El paso 6 del algoritmo 4.5, la suma de los polinomios, se realiza en el ciclo 264.

Finalmente el paso 7 del algoritmo es obtenido en el ciclo 267, con la señal de salida de xtimes.

El diagrama de estados de la función $F^{(2)}$ se muestra en la Fig. 4.7, mientras que el significado de cada estado se pueden ver en las tablas 4.2 4.3.

Estado	Significado
squaringLxorH	Se asigna la señal 'LxorH' a la señal 'squarein'.
setSLxorH	Se asigna la señal 'squareout' al registro 'SLxorH'
multSHL	Se asignan los valores de los registros 'SLxorH' y 'L' como operandos del multiplicador Karatsuba.
Mult4a1	Se asignan el valor del registro 'L4' y de la señal 'a' como operandos del multiplicador Karatsuba.
Mult4a2	Mientras se comienza la multiplicación del cuarto bloque de mensaje ('L4' y 'a' siguen siendo los operandos de entrada), se obtiene el resultado de la multiplicación 'SLxorH' y 'L', el cual es guardado en el registro 'AC2'.
Mult4xA	Se asignan los valores de las señales 'L4xorA' y 'L4' como operadores del multiplicador Karatsuba, de esta forma estamos calculando el valor $(L^4)^2 + L^4 x_{[5]}$.
multCL4	En este estado se realiza la mayor parte de las operaciones necesarias para realizar la evaluación polinomial, cada ciclo de reloj el multiplicador recibe como entrdas las señales 'mxorc' y 'L4'. Cada una de las cuatro pequeñas evaluaciones polinomiales en la que se ha dividido la evaluación polinomial inicial debe de ser multiplicada por L^4, L^3, L^2 o L como se muestra en 4.2, es por esto que este estado además de calcular los pequeños polinomios se utiliza también para multiplicar al primero por el valor L^4.
multCL3	En este estado el multiplicador recibe como entrada las señales 'mxorc' y 'L3'.

Tabla 4.2: Se muestran los nombres y significados de los primeros estados de la Fig. 4.7.

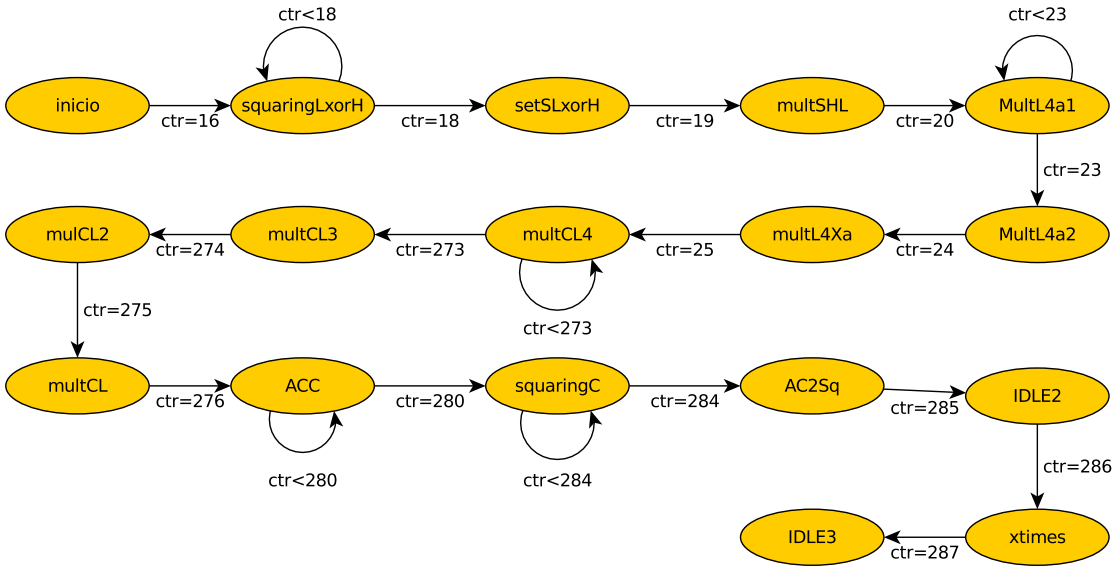

Figura 4.7: Diagrama de estados de la función $F^{(2)}$.

4.4.3. Modo Contador de HBS

El vector inicial del modo contador es el valor obtenido por $F^{(2)}(L, L^2, L^3, L^4, H, X)$, el modo contador de HBS es un modo contador estándar, donde el valor de un contador es sumado al vector inicial, el resultado a su vez es cifrado utilizando el cifrador por bloques (AES). Estos bloques cifrados son utilizados para enmascarar a los bloques del mensaje. En 4.8 se puede ver la arquitectura necesaria para el modo contador de HBS.

4.4.4. Entrega del código de autenticación

Este paso consiste en cifrar el resultado obtenido por $F^{(2)}(L, L^2, L^3, L^4, H, X)$ con el cifrador AES y entregar el valor resultante como MAC .

4.4.5. Validación de datos

Esta parte es exclusiva del proceso de descifrado y consiste en cifrar la cadena obtenida durante la evaluación polinomial, a este valor lo llamaremos MAC' , después

Estado	Significado
multCL2	En este estado el multiplicador recibe como entrada las señales 'mxorc' y 'L2'.
multCL	En este estado el multiplicador recibe como entrada las señales 'mxorc' y 'L'.
ACC	Se realiza la suma de los polinomios para obtener el resultado final de la evaluación polinomial.
squaringC	Se eleva al cuadrado el resultado de la evaluación polinomial y se espera un resultado válido.
AC2Sq	Se inicia la suma en el acumulador AC2 de la evaluación polinomial del mensaje y la cabecera .
IDLE2	Se espera la actualización de señales.
xtimes	Se realiza la operación XTIMES en la sobre el valor obtenido en el acumulador AC2.
IDLE3	Termina la evaluación polinomial y comienza la generación del vector inicial a partir de esta, así como el cifrado mediante el modo contador.

Tabla 4.3: Se muestran los estados de la Fig. 4.7 faltantes en la tabla 4.2.

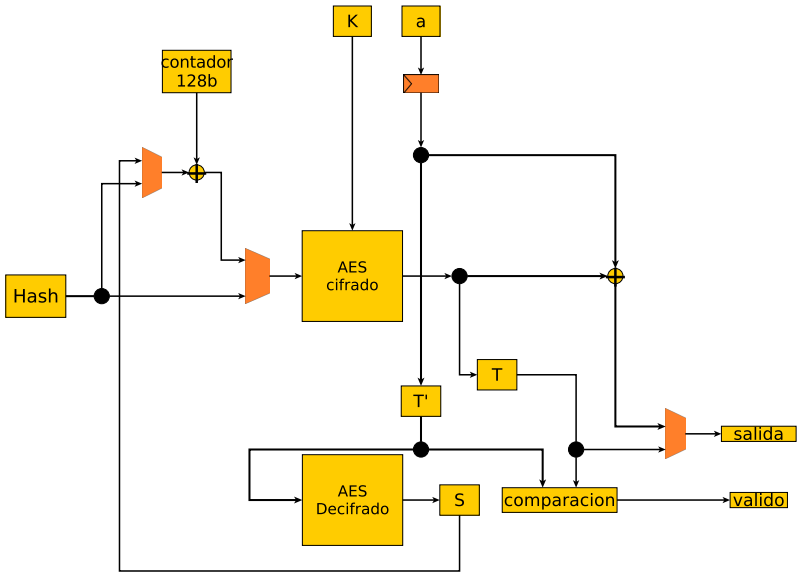

Figura 4.8: Arquitectura necesaria para el modo contador de HBS.

compararemos a MAC' con el parámetro MAC , en caso de que ambos valores sean iguales se devolverá el valor 0, lo cual indica que la pareja de texto cifrado y MAC es válida, en otro caso se devolverá el valor 1.

4.5. Implementación de BTM

Este esquema, como en otros esquemas DAE, se puede dividir en un algoritmo de cifrado de solo privacidad y una función picadillo que genera el vector inicial de dicho algoritmo de cifrado. Para esta implementación en particular se decidió implementar dos funciones picadillo distintas, la primera optimizada para cabeceras compuestas de solamente un bloque, mientras que la segunda implementación es una versión capaz de aceptar cabeceras de una longitud arbitraria.

La arquitectura del esquema BTM se puede ver en la Fig. 4.12, en ella se puede reconocer tanto el cifrador por bloques AES como el multiplicador de Karatsuba, la sincronización de estos elementos se realizará mediante una unidad de control, la cual consistirá de una máquina de estados finitos cuyas transiciones serán dependientes de un contador, un bit de entrada que permite diferenciar entre si se está cifrando o descifrando (*mode*) y un bit que anuncia cuando se ha terminado de realizar la evaluación polinomial (*fhash*). Los procesos tanto de cifrado como de descifrado comienzan con el encendido de la señal *reset*, esta señal debe de permanecer activa por al menos un ciclo de reloj completo. La señal *mode* debe de obtener su valor final mientras *reset* esta activo.

El algoritmo BTM de cifrado puede ser dividido en cuatro procesos distintos, mientras que el algoritmo de descifrado puede ser dividido en cinco partes , a continuación se mencionan estos procesos y su relación con el algoritmo 4.10:

1. La generación de variables: L^4, L^3, L^2, L y U (líneas 1 a 5 en el algoritmo de cifrado, líneas 1 a 6 en el algoritmo de descifrado).

2. La evaluación polinomial en dos variables (línea 6, línea 8 en el algoritmo de descifrado).

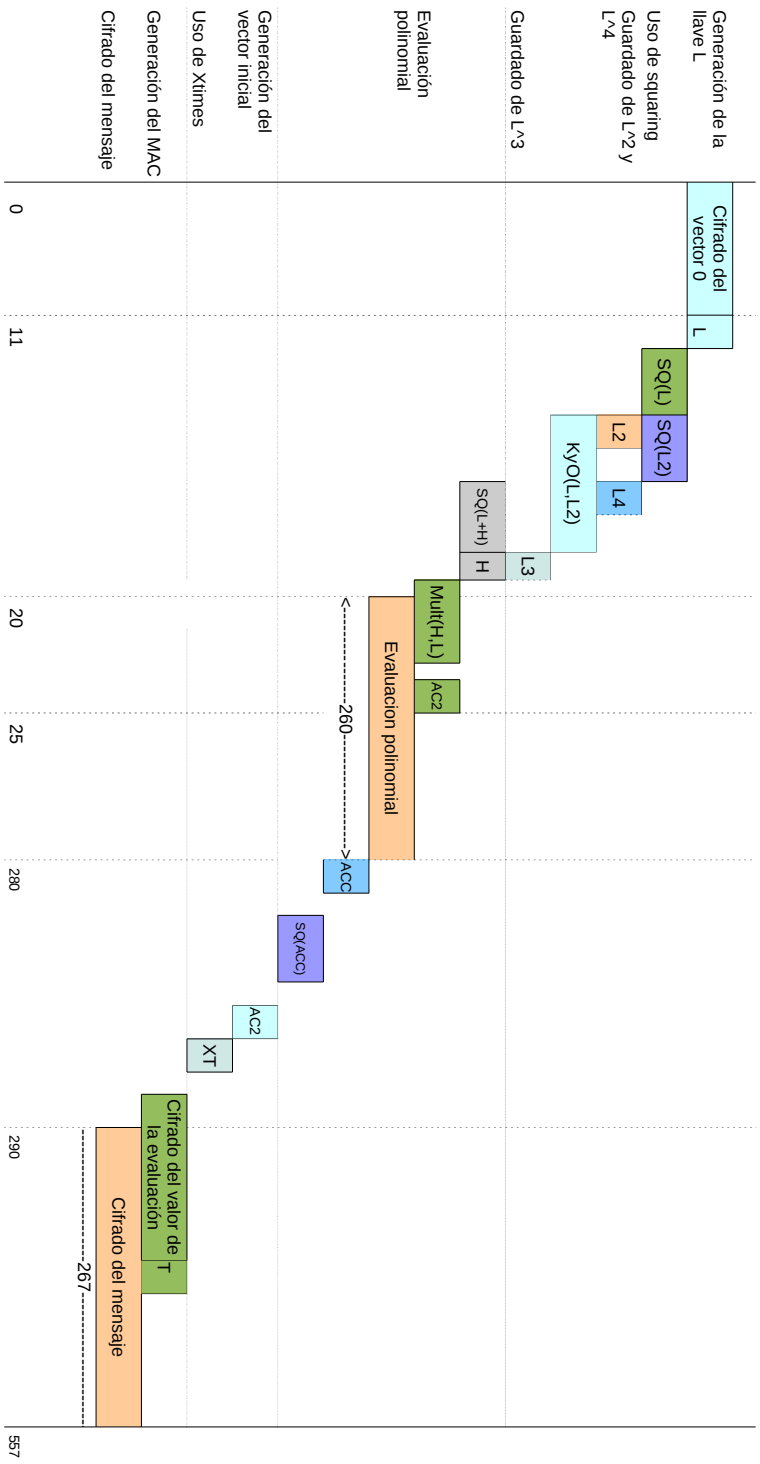

Figura 4.9: Diagrama de tiempos HBS.

$\text{BTM.Enc}_K(H, M)$
1. $L \leftarrow E_K(\underline{0}_n)$
2. $L^2 \leftarrow \mathsf{squaring}(L)$
3. $L^3 \leftarrow L \cdot L^2$
4. $L^4 \leftarrow \mathsf{squaring}(L^2)$
5. $U \leftarrow E_K(\underline{1}_n)$
6. $\mathsf{poly} \leftarrow P_{L,U}(H, M)$
7. $T \leftarrow E_K(\mathsf{poly})$
8. $B \leftarrow T \boxplus U$
9. $C \leftarrow \mathsf{CTR}_K^B(M)$
10. **return** (T, C)

Figura 4.10: $\text{BTM.Enc}_K(H, M)$

$\text{BTM.Dec}_K(H, (T, C))$
1. $L \leftarrow E_K(\underline{0}_n)$
2. $L^2 \leftarrow \mathsf{squaring}(L)$
3. $L^3 \leftarrow L \cdot L^2$
4. $L^4 \leftarrow \mathsf{squaring}(L^2)$
5. $U \leftarrow E_K(\underline{1}_n)$
6. $B \leftarrow T \boxplus U$
7. $M \leftarrow \mathsf{CTR}_K^B(H, C)$
8. $\mathsf{poly} \leftarrow P_{L,U}(H, M)$
9. $T' \leftarrow E_K(\mathsf{poly})$
10. **if** $T \neq T'$ **then**
11. $\quad M \leftarrow \perp$
12. **end if**
13. **return** M

Figura 4.11: $\text{BTM.Dec}_K(H, (T, C))$

3. El procesamiento del resultado de la evaluación polinomial (líneas 7 y 8 en el algoritmo de cifrado, línea 9 en el algoritmo de descifrado).

4. El cifrado de Datos (línea 9 en el algoritmo de cifrado línea 7 en algoritmo de descifrado).

5. Validación de Datos (líneas 10 a 13 en el algoritmo de descifrado)

Se implementaron dos variantes del algoritmo BTM: Una donde se consideraba una cabecera que consistía de un solo bloque y 256 bloques de 128 bits; y otra donde tanto el número de cabeceras como su longitud podían variar, así como la longitud del mensaje. El circuito de control es similar en ambos casos salvo la evaluación polinomial y el cifrado de datos. Es posible utilizar el mismo circuito de control para ambas implementaciones con algunos ajustes. El control para una sola cabecera con longitud fija y con mensaje de longitud fija se realiza mediante un contador *ctr*. El control para mensajes de varias cabeceras de longitud variable y mensajes de longitud variable contará con un modo de detener el incremento de *ctr* cuando éste comience la evaluación polinomial o el cifrado del mensaje, y cuando estos terminen se le asignará a *ctr* el valor deseado.

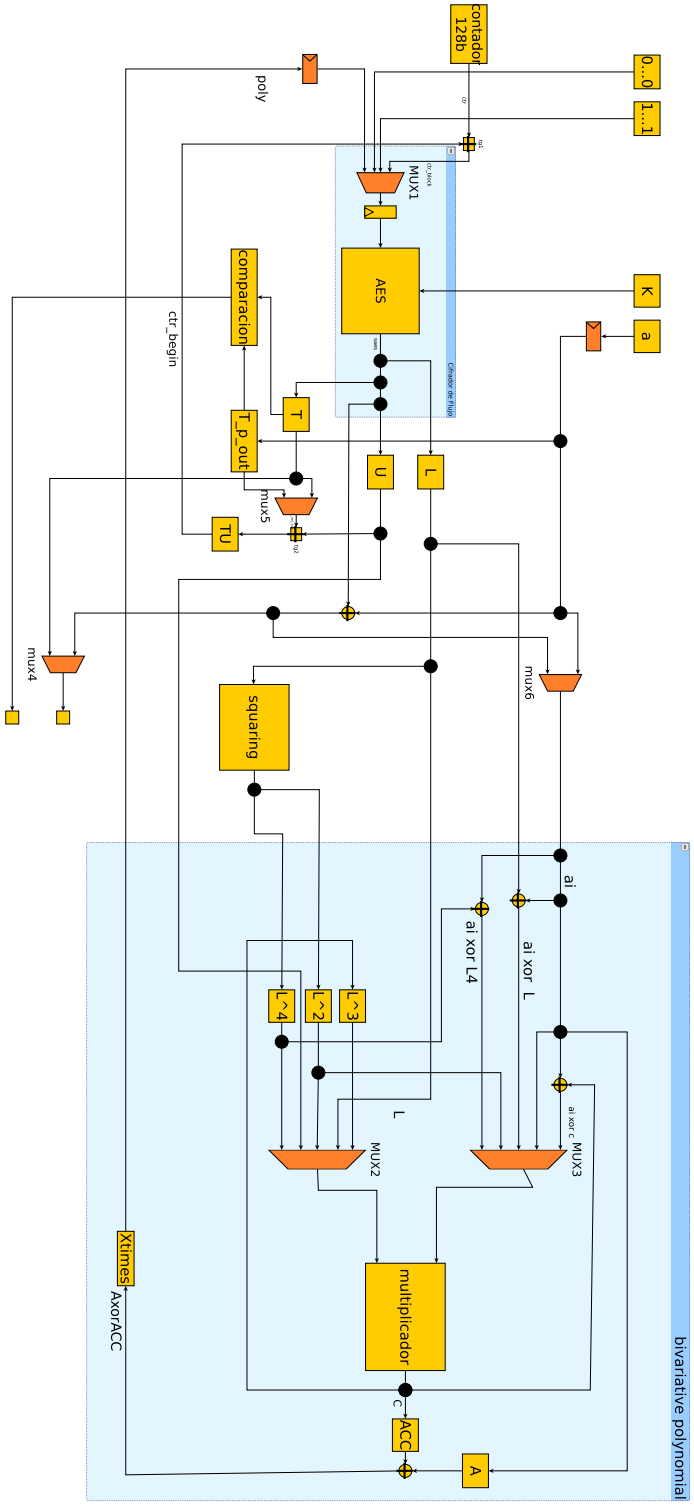

Figura 4.12: Arquitectura desarrollada para BTM.

4.5.1. Proceso de generación de Variables

Este proceso consiste en generar las llaves L y U, así como precalcular los valores L^2, L^3 y L^4. Los valores L^2, L^3 y L^4 serán usados durante la evaluación polinomial tanto de la cabecera como del mensaje.

Como se puede ver en la linea 1 del algoritmo 4.10 la llave L se obtiene de cifrar la cadena de bits $\underline{0}_{128}$, mientras que la llave U se obtiene de cifrar la cadena de bits $\underline{1}_{128}$.

En el proceso de descifrado, también se debe de calcular el valor B a partir de la operación XOR aplicada al MAC y a la llave U.

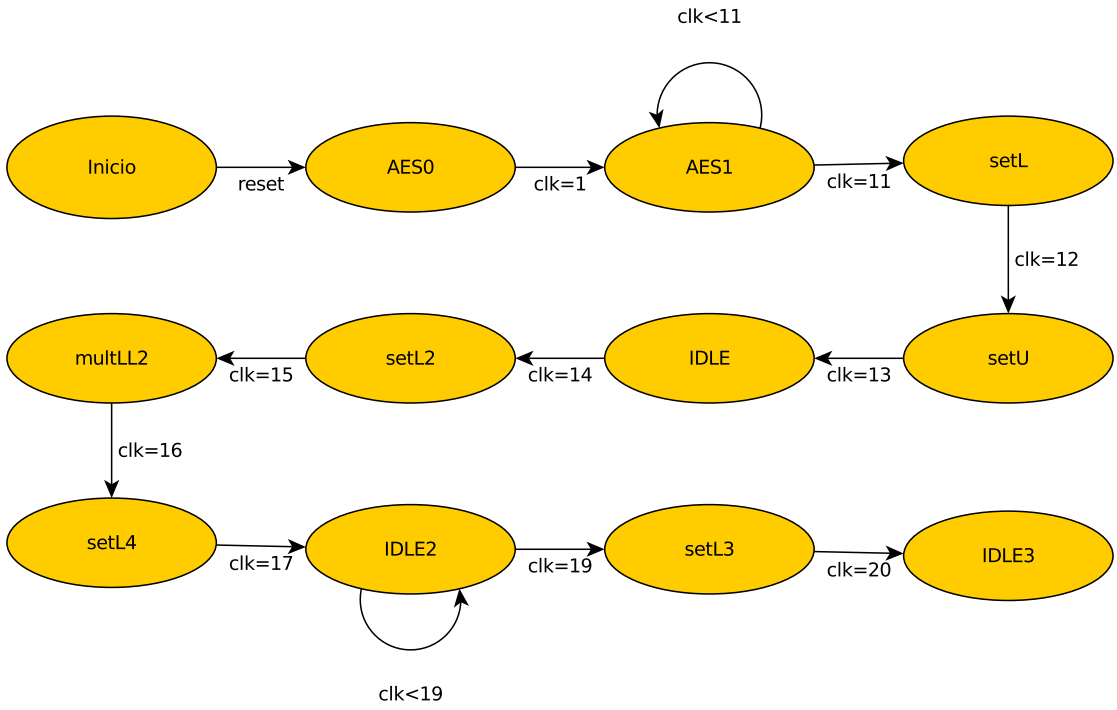

Figura 4.13: Diagrama de estados de la generación de variables de BTM.

Estado	Significado
AES0	Se asigna el valor $\underline{0}_{128}$ como entrada del cifrador AES.
AES1	Se asigna el valor $\underline{1}_{128}$ como entrada del cifrador AES, y se espera por un resultado válido de AES.
setL	Se asigna el resultado de AES (el cifrado del bloque $\underline{0}_{128}$) al registro'L'.
setU	Se asigna el resultado de AES (el cifrado del bloque $\underline{1}_{128}$)al registro'U'. También se introduce el valor del registro 'L' a la función squaring.
IDLE	Se espera por un resultado válido de squaring.
setL2	Se asigna el resultado de squaring al registro 'L2'. Se comienza el cálculo de L^4 al asignar la salida de squaring como su misma entrada.
multLL2	Se asignan a las señales 'L2' y 'L' como entradas del multiplicador, con el fin de obtener L^3.
setL4	Se asigna el resultado de squaring al registro 'L4'.
IDLE2	Se espera por el resultado de multiplicar las señales $L2$ y L.
setL3	Se asigna el valor obtenido por el multiplicador al registro 'L3'
IDLE3	Finaliza la generación de variables.

Tabla 4.4: Definición de estados del diagrama mostrado en la Fig. 4.13

4.5.2. Función picadillo de cabeceras de un solo bloque

La función picadillo de un mensaje X de 256 bloques (4096 bytes) con una cabecera $\vec{H}$ de solo un dato se calcularía de la siguiente forma:

$$\begin{aligned} P_{U,L}(\vec{H}, X) &= (L \oplus h) \cdot U \cdot 2 \oplus (L^{526} \oplus L^{255} \cdot X_{[0]} \oplus \cdots \oplus L \cdot X_{[254]} \oplus X_{[255]}) \cdot 2 \\ &= 2 \cdot ((L \oplus h) \cdot U \oplus L^{526} \oplus L^{255} \cdot X_{[0]} \oplus \cdots \oplus L \cdot X_{[254]} \oplus X_{[255]}) \end{aligned}$$

Para este caso en particular, se puede ver que al bloque de la cabecera se le aplica la operación XOR junto con la llave L y se introduce este valor al multiplicador, cuatro ciclos de reloj después se agrega el resultado al sumador de la función picadillo. Es posible comenzar a realizar la evaluación polinomial de los bloques del mensaje una vez que se han introducido los valores de la cabecera.

$P^{(2)}_{L,L^2,L^3,L^4}(H, X)$
1. $\mathsf{LxorH} \leftarrow L \oplus H$
2. $\mathsf{sum} \leftarrow \mathsf{LxorH} \cdot U$
3. $\mathsf{sum} \leftarrow \mathsf{sum} \oplus L^{256} \oplus L^{255} \cdot X_0 \oplus \cdots \oplus L \cdot X_{254} \oplus X_{255}$
4. $\mathsf{poly} \leftarrow \mathrm{xtimes}(\mathsf{sum})$

Figura 4.14: $P^{(2)}_{L,L^2,L^3,L^4,U}(H, X)$

4.5.3. Evaluación polinomial del mensaje

La implementación de BTM considera únicamente mensajes de 4096 bytes, por lo que cada mensaje X está compuesto por 256 bloques ($X = X[0]||X_{[1]}||...||X_{[255]}$) de 128 bits. La evaluación polinomial de este mensaje se puede realizar de manera similar a la mostrada en 4.1.

La evaluación polinomial del mensaje puede ser dividida en 4 partes:

Inicial Consiste en los primeros cuatro ciclos de reloj, los primeros tres ciclos se calculan los valores $L^4X_{[0]}$, $L^4X_{[1]}$ y $L^4X_{[2]}$, obteniendo los respectivos resultados en los ciclos 4, 5 y 6. En el ciclo 4 se calcula el valor $L^4(L^4 \oplus X_{[3]})$.

Media Esta fase comprende desde el ciclo 5 hasta el ciclo 252, donde las señales de entrada del multiplicador se fijan en L^4 y $X_{[i-1]} \oplus c$, donde c es la señal de salida del multiplicador, de esta forma se calcula la mayor parte de la evaluación polinomial.

Final En esta fase se calcula el valor final de cada parte del polinomio. En los ciclos 253, 254 y 255 se obtienen los resultados de las partes en las que se ha dividido el polinomio, cada una de ellas es multiplicada por L^3,L^2 y L respectivamente. Es por esto que las señales de entrada al multiplicador son $X_{[i-1]} \oplus c$ y L^3 en el ciclo 253, L^2 en el ciclo 254 y L en el ciclo 255.

Suma Se almacena el bloque 256 del mensaje. Se agregan los resultados de las 4 partes

del polinomio al acumulador en los ciclos 257, 258, 259 y 260. Y finalmente se suma el bloque 256 (ciclo 261)

El resultado de la evaluación polinomial es aceptada por Xtimes (ciclo 261) y para el ciclo 262 la evaluación polinomial ha sido realizada.

La maquina de estados de la evaluación polinomial del encabezado de un solo bloque y de un mensaje de 256 bloques de muestra en la Fig. 4.15. La definición de cada estado se da en las tablas 4.5 y 4.6.

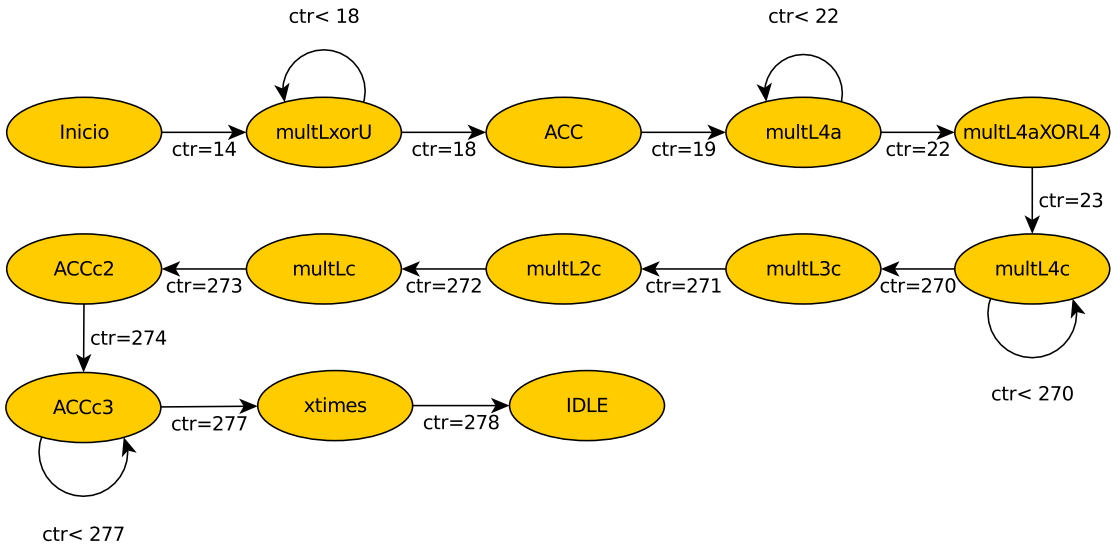

Figura 4.15: Diagrama de estados de la evaluación polinomial de dos variables de BTM, para cabeceras de un solo bloque.

4.5.4. Función picadillo de cabeceras de longitud arbitraria

Existen varios problemas para realizar evaluaciones polinomiales con cabeceras de longitud arbitraria, la posibilidad de que la cabecera tenga únicamente un bloque o 500, un número arbitrario, complica el diseño.

La idea básica para el desarrollo de esta evaluación polinomial es que el circuito que provee la información, sea capaz de saber cuando falten 4 bloques para ser evaluados.

Estado	Significado
mulLxorU	Se dan como entradas del multiplicador KM los valores del registro 'U' y de la señal 'LxorH'. Despúes se espera por un resultado válido del multiplicador. La señal 'LxorH' se obtiene de aplicar la operación XOR a la señal de salida del registro 'L' y un bloque de datos, que se supone es el bloque que conforma la cabecera de datos.
ACC	Se suma el resultado del multiplicador al acumulador 'ACC'.
multL4a	Se asignan como entradas del multiplicador la salida del registro 'L4' y un bloque de datos, con esto comienza la evaluación polinomial del mensaje.
multL4aL4	Se asignan como entradas del multiplicador la salida del registro 'L4' y 'aXORL4'. 'aXORL4' se obtiene de aplicar la operación XOR entre el bloque de mensaje entrante y la señal del registro 'L4'. Este paso es necesario para obtener el valor $L^2 56$.
multL4c	En este estado se asignan como entradas del multiplicador KM el valor del registro 'L4' y la señal 'aXORc'. La señal 'aXORc' se obtiene de aplicar la operación XOR entre la salida actual del multiplicador y un bloque de datos de entrada. Durante este estado se realiza la mayor parte de la evaluación polinomial del mensaje.
multL3c	Al igual que en el estado anterior, se asigna como entrada del multiplicador la señal 'aXORc', pero el otro operando del multiplicador es el valor del registro 'L3'.
multL2c	Al igual que en el estado anterior, se asigna como entrada del multiplicador la señal 'aXORc', pero el otro operando del multiplicador es el valor del registro 'L2'.

Tabla 4.5: Definición de los estados del diagrama 4.15

Estado	Significado
multLc	Al igual que en el estado anterior, se asigna como entrada del multiplicador la señal 'aXORc', pero el otro operando del multiplicador es el valor del registro 'L'.
ACCc2	Se suman en el acumulador 'ACC' el resultado obtenido por el multiplicador. Se almacena el bloque de datos entrante en el registro 'A'.
ACCc3	Se suman en el acumulador 'ACC' el resultado obtenido por el multiplicador.
xtimes	Se asignan como entrada de la función xtimes la señal 'AxorACC', esta señal se obtiene de aplicar la operación XOR entre la señal 'A' y la salida del acumulador 'ACC'.

Tabla 4.6: Definición de los estados del diagrama 4.15

Las señales necesarias para poder controlar la evaluación polinomial de cabeceras de longitud arbitraria son las siguientes:

- *fin* : Es un vector de 3 bits, con el cual se indica uno de 3 estados a : Esta entrando el ultimo bloque: '001'; esta entra entrando el penúltimo bloque: '010'; esta entrando el ante penúltimo bloque: '011', faltan 4 bloques para terminar: '100', faltan al menos 5 bloques del vector: '000'.

- reset : Esta es una señal de entrada que indica que todo el esquema comenzará a ser usado, por lo que los acumuladores son puestos en 0

- *npoly* : Esta es una señal de entrada y es usada para indicar que se introducirá otra cabecera.

- inrdy : Esta es una señal de salida que indica que se debe de seleccionar el valor adecuado de *fin* , y que en dos ciclos de reloj despuś se debe de comenzar a introducir los bloques de datos.

- rdy : Esta señal de salida indica cuando finalmente se puede recoger el resultado actual de la evaluación polinomial y es usada para informar que se puede iniciar a evaluar otro polinomio o bien se puede proseguir con el cifrado del mensaje.

- headrdy : Esta señal interna le indica al circuito que puede continuar con el cifrado del mensaje.

Se muestra en la Fig. 4.16 la máquina de estados que se utiliza para elaborar el picadillo de las cabeceras. Las señales fase1, fase2 y fase3 son contadores que permiten identificar en que parte del proceso se encuentra la evaluación polinomial. El significado de cada estado es explicado en las tablas 4.5.4 y 4.5.4.

Este mismo circuito realiza la evaluación polinomial del mensaje, ya que para términos prácticos, al momento de realizar la evaluación polinomial, el mensaje es igual a una cabecera, con la diferencia de que este no debe de ser multiplicado por la llave U.

Estado	Significado
cero	En este estado se establece como entrada del multiplicador $\underline{0}_{128}$ y la salida del registro $L4$ (aunque se puede utilizar cualquier otro valor), y se espera que el valor de la señal fase1 sea igual a '010', de esta forma el multiplicador devolverá al menos tres $\underline{0}_{128}$ como primeros resultados de la evaluación, polinomial, esto se hizo para facilitar la evaluación polinomial de cadenas que tengan tres o menos bloques.
sel	En este estado se activa la señal inrdy.
masD4	En este estado se asignan como entradas del multiplicador los valores $L4$ y $\underline{1}_{128}$, con la intención de generar el único valor del polinomio cuyo coeficiente, no es dado por un bloque de datos, en caso de una evaluación polinomial de una cabecera de 256 bloques, el valor sería L^{256}.
solo4	En este estado se asignan como entradas del multiplicador los valores $L4$ y $\underline{1}_{128}$, se le asigna el valor '0001' a la señal fase2.
solo3	En este estado se asignan como entradas del multiplicador los valores $L3$ y $\underline{1}_{128}$, se le asigna el valor '0010' a la señal fase2.
solo2	En este estado se asignan como entradas del multiplicador los valores $L2$ y $\underline{1}_{128}$, se le asigna el valor '0011' a la señal fase2.
solo1	En este estado se asignan como entradas del multiplicador los valores L y $\underline{1}_{128}$, se le asigna el valor '0100' a la señal fase2.
last5	Se le asigna el valor '0001' a la señal fase2.
last4	Se asigna la señal de salida del registro $L4$ y la señal a (señal por donde se introducen los bloques de datos) al multiplicador.
last3	Se asigna la señal de salida del registro $L3$ y la señal a al multiplicador.

Tabla 4.7: Definición de los estados del diagrama 4.16

Estado	Significado
last2	Se asigna la señal de salida del registro $L2$ y la señal a al multiplicador.
last1	Se asigna la señal de salida del registro L y la señal a al multiplicador.
acc	Se suma en el acumulador ACC la señal de salida del multiplicador.
wait	Se activa la señal rdy.
multU	Se desactiva la señal rdy, se asignan como entradas del multiplicador la señal de salida del registro U y la la salida del multiplicador.
accNull	Se reinicia el acumulador ACC.
wait2	Se espera por el resultado de la operación asignada en el estado multU.
accU	Se almacena en el acumulador ACC la salida del multiplicador, se asigna el valor '000' a fase1.

Tabla 4.8: Definición de los estados del diagrama 4.16

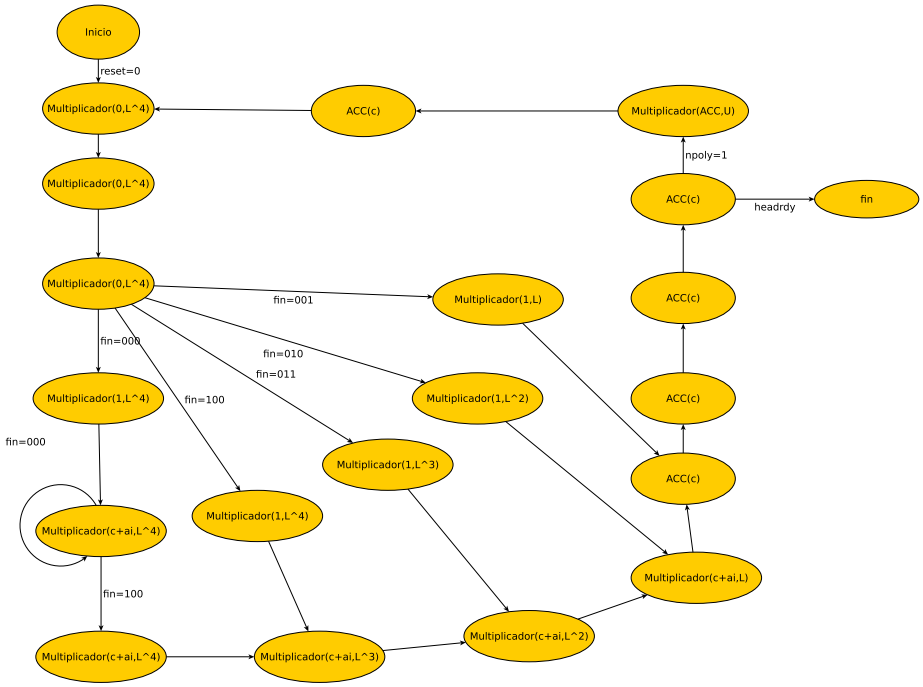

Figura 4.16: Maquina de estados de la función picadillo para cabeceras de longitud variable.

4.5.5. El procesamiento del resultado de la evaluación polinomial

El resultado de la evaluación polinomial, es dado como parámetro a la función de cifrado de AES y se obtiene poly como resultado, dicho valor es devuelto como MAC .

Si se esta cifrando texto, se obtiene el valor B de aplicar la operación XOR a los valores poly y U.

4.5.6. Modo contador de BTM

El modo contador utiliza 5 partes de la arquitectura:

- El cifrador por bloques AES.
- El contador de 128 bits.

- Dos operaciones Tag Mixing.
- El multiplexor 1
- El multiplexor 2

Una vez que se ha obtenido el valor de T, el vector inicial del modo contador se calcula al realizar la operación tag mixing entre el valor T y la llave U. El contador de 128 bits recibirá una señal de reinicio *resetContador* , y comenzará su cuenta desde el valor 0, a dichos valores se les aplicará la función *Tag Mixing* junto con el vector inicial recién obtenido, cada ciclo de reloj, el contador se incrementará en una unidad. Al resultado de la operación *Tag Mixing* entre el valor del contador y el vector inicial lo llamaremos *valorContador* .

El *valorContador* será utilizado como bloque de entrada para AES, y el bloque resultante será utilizado para enmascarar el valor el bloque del texto a cifrar:

$$C_{[n]} = \mathrm{AES}_K(T \boxplus U \boxplus n) \oplus X_{[n]}$$

4.5.7. Validación de Datos

Esta parte es exclusiva del modo de descifrado y consiste básicamente en comparar el valor T' y el parámetro T en caso de ser iguales se regresa el valor 0, que significa que la pareja texto cifrado y MAC es válida. En otro caso se devuelve el valor 1, lo que da a entender que la pareja texto cifrado y MAC es inválida.

4.6. Medidas de Rendimiento en FPGA

Al momento de medir la eficiencia y calidad de un diseño para FPGA se utilizan dos criterios: El Área utilizada y el tiempo de procesamiento. Las definiciones de estos son las siguientes: Área: Se define como la cantidad de recursos ocupados por un diseño y usualmente se mide en slices utilizados. Algunos FPGAs tienen otros recursos como BRAMs, multiplicadores, etc. Si se ocupa alguno de estos recursos es

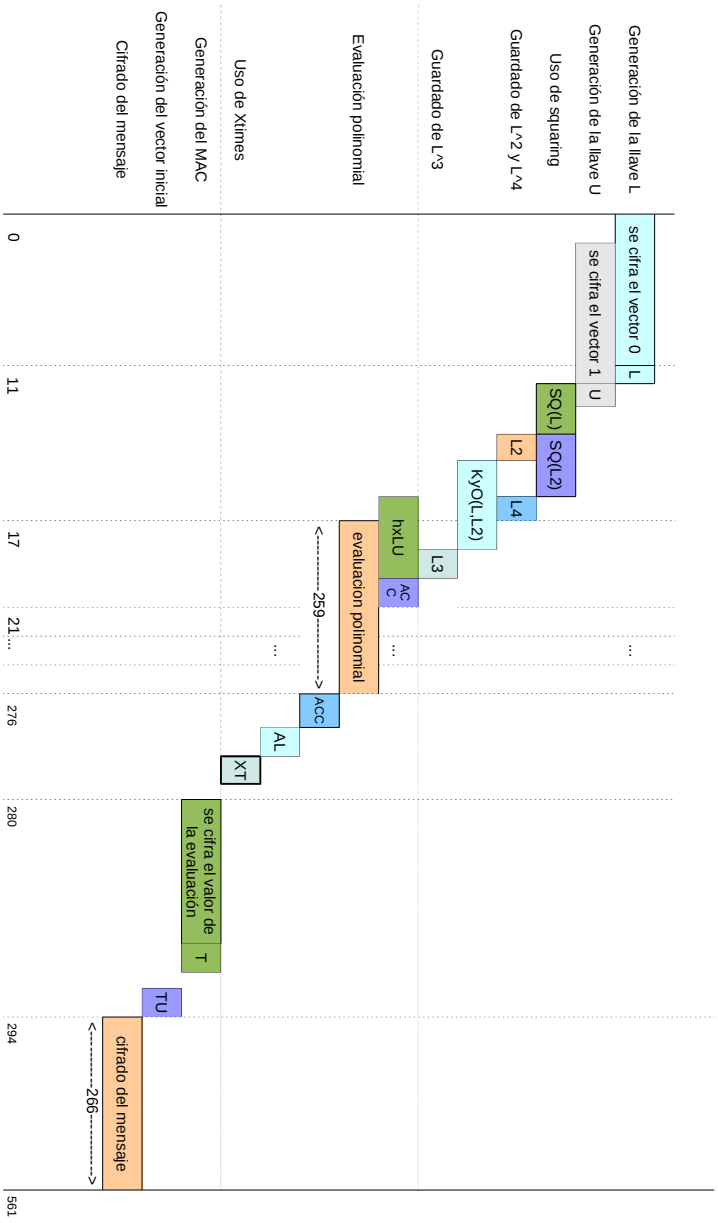

Figura 4.17: Diagrama de tiempos BTM de cabeceras de un solo bloque.

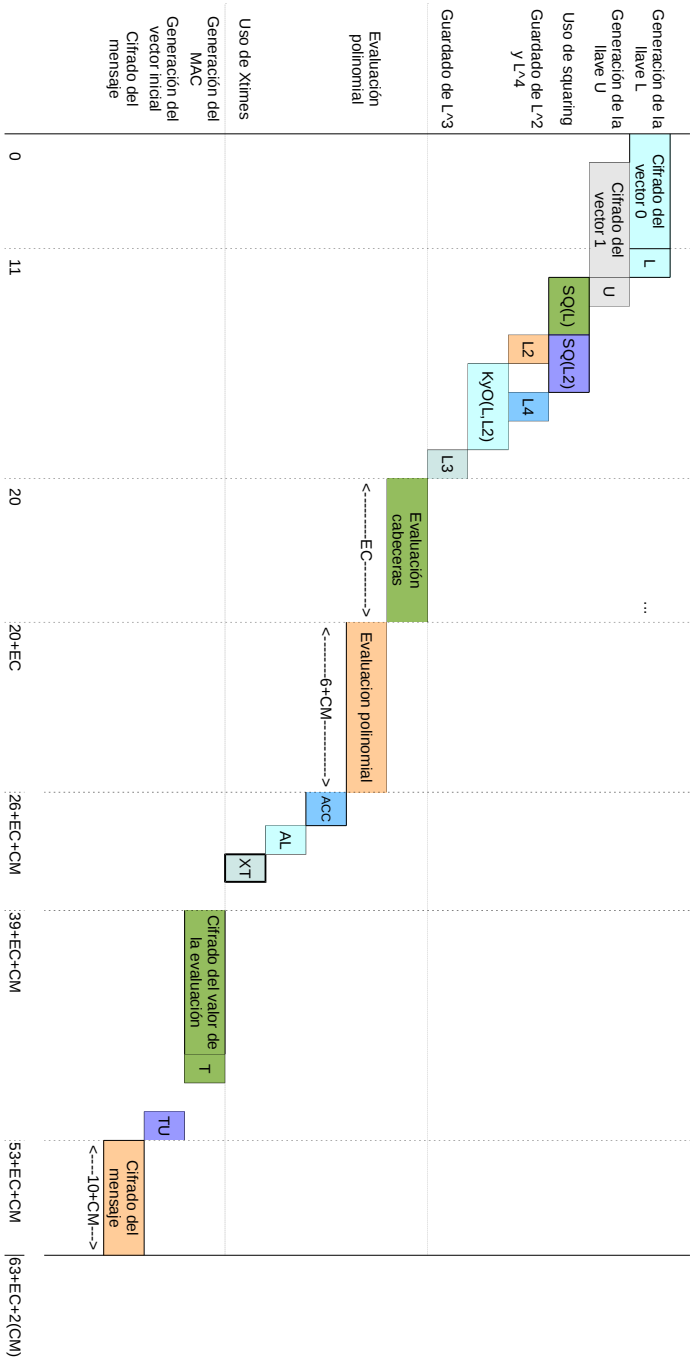

Figura 4.18: Diagrama de tiempos BTM de cabeceras de múltiples cadenas y múltiples bloques. El tamaño de EC es directamente proporcional al tamaño y cantidad de las cabeceras. EC=$6 \cdot ||\vec{H}|| + \sum_{i=1}^{||\vec{H}||} \eta(H_i)$. CM es la cantidad de bloques que contiene el mensaje.

Componente	Area (SLICES)	Ruta Crítica (nS)
Ronda AES cifrado	688	2.951
Ronda de AES descifrado	1876	3.419

Tabla 4.9: Resultados de la implementación de una ronda de AES de cifrado y de una ronda de AES de descifrado.

importante reportarlos ya que estos no ocupan slices, de esta manera se justifica el ahorro de área de un diseño [30].

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

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

El segundo criterio es la latencia, la cual es el tiempo que tarda un diseño en entregar el primer bloque válido en la salida. A continuación con las métricas explicadas se evaluarán los diseños implementados en esta tesis.

4.7. Resultados

En esta sección presentamos los resultados de nuestras implementaciones. Todas las implementaciones fueron hechas en un dispositivo Virtex 5 XC5VLX85. Los resultados reportados son obtenidos de la simulación "post-place & route" ofrecido por la herramienta Xilinx-ISE versión 10.1.

En la tabla 4.7 se muestra el desempeño de las rondas de AES, como se puede ver, la ronda de descifrado es más lenta y ocupa una mayor área, esto es debido a que es más compleja, esta ronda puede funcionar como una ronda normal de descifrado de AES o como una última ronda de descifrado de AES.

En la tabla 4.7 se puede observar el desempeño de los bloques básicos que se

Componente	Area (SLICES)	Frecuencia (MHz)	ciclos de reloj	Tasa de cifrado (GbS)
AES en pipeline	3357	291.630	11	37.32
AES descifrado	1876	246.562	21	31.19
Multiplicador	2560	292.320	4	37.41

Tabla 4.10: Resultados de las implementaciones de los bloques básicos.

Esquema	Area (SLICES)	FRECUENCIA (MHz)	Ciclos de reloj	tasa de cifrado (Gbs)
BTM	6421	291.715	567	16.865
HBS	4928	246.430	563	14.34

Tabla 4.11: Resultados de las implementaciones de los esquemas BTM y HBS.

utilizaron para implementar los modos de operación. Como era de esperarse el área que ocupa el AES de descifrado es menor que el área que ocupa el AES de cifrado, esto es por que por cada ronda que se necesita el AES de cifrado se implementó un circuito diferente, mientras que para el AES de descifrado, se reutiliza la misma ronda.

Finalmente en la última tabla es posible observar el desempeño de las arquitecturas por completo, haciendo uso de los componentes descritos anteriormente. Como se puede observar la ruta crítica es determinada por el cifrador en BTM y por el AES de descifrado en HBS, también se puede observar que, en lo que a latencia refiere, el desempeño es mejor por parte de HBS debido a que se necesitan precalcular menos elementos y el modo contador es más sencillo, sin embargo, la ganancia en desempeño no justifica el aumento en área ni la reducción de frecuencia.

En ambos casos, el desempeño es mejor al momento de descifrar, ya que para cifrar es necesario leer dos veces la información a cifrar: la primera vez es para obtener el vector inicial del modo contador y la segunda para cifrar dicha información, mientras que al descifrar es posible obtener el mensaje descifrado y comenzar a calcular el MAC al mismo tiempo.

4.8. Sumario

Este capítulo se centra en la implementación de los esquemas HBS y BTM, se dan las consideraciones de diseño, se describe como se implementaron las operaciones Xtimes y elevar al cuadrado. También se muestra como se dividen las evaluaciones polinomiales para poder utilizar el multiplicador Karatsuba de forma óptima. Finalmente se dan detalles de como se conectaron los componentes descritos en este capítulo y en el capítulo anterior, así como las máquinas de estados que permiten controlar las implementaciones.

Por último se muestran los resultados obtenidos en nuestras implementaciones, así como algunas métricas de desempeño.

Capítulo 5

SCDAE: Un esquema CAD basado en cifradores de flujo

En este capítulo se muestra un nuevo esquema CAD que le llamamos SCDAE, el cual, a diferencia de los otros esquemas CAD existentes no hará uso de un cifrador por bloques, en su lugar se hará uso de un cifrador por flujo. Los motivos para realizarlo son, principalmente, la velocidad que suelen alcanzar los cifradores de flujo y el poco uso de recursos que estos pueden llegar a hacer cuando son implementados en hardware. La idea básica de SCDAE consiste en reemplazar el esquema de cifrado de solo privacidad que usualmente se usa en los esquemas CAD con un cifrador de flujo con vector inicial, aunque es posible utilizar una mezcla de cifradores de flujo y cifradores por bloques para realizar un esquema CAD, esto resultaría en desperdicio de recursos, por lo que, dado que sustituiremos el cifrador por bloques de la función de sólo cifrado, por el cifrador de flujo con vector inicial.

5.1. SCDAE

SCDAE hace uso de 3 llaves con una longitud de n bits $L, U, K \in \{0,1\}^n$, de aquí en adelante toda n que se mencione hará referencia al mismo número natural. Dos llaves (L, U) sirven para generar el vector inicial al ser utilizados en una evalua-

SCDAE.$\mathcal{E}_{K,U,L}(\vec{H}, M)$	SCDAE.$\mathcal{D}_{K,U,L}(\vec{H}, C, T)$
1. $\gamma \leftarrow P_{U,L}(\vec{H}, M)$ 2. $T \leftarrow \mathsf{MSB}_n(SC_k(\gamma))$ 3. $\tau \leftarrow T \oplus U$ 4. $R \leftarrow \mathsf{MSB}_{\|M\|}(SC_k(\tau,))$ 5. $C \leftarrow M \oplus R$ 6. **return** (T, C)	1. $\tau \leftarrow T \oplus U$ 2. $R \leftarrow \mathsf{MSB}_{\|C\|}(SC_k(\tau))$ 3. $M \leftarrow C \oplus R$ 4. $\gamma \leftarrow P_{U,L}(\vec{H}, M)$ 5. $T' \leftarrow \mathsf{MSB}_n(SC_k(\gamma))$ 6. **if** $T = T'$**then** 7. **return** (M) 8. **else** 9. **return** $\perp$ 10. **end if**
$P_{U,L}(\vec{H}, M)$	$f_L(M)$
1. $p \leftarrow \underline{0}_n$ 2. **para todo** $H_i \in \vec{H}$ **hacer:** 3. $p \leftarrow U \cdot (p \oplus f_L(H_i))$ 4. **fin para** 5. $p \leftarrow p \oplus f_L(M)$ 6. **return** p	1. $p \leftarrow L$ 2. $M = m_1 \| \| ... \| \| m_{\eta_n(m)}$ 3. **for** $i = 1$ **to** $\eta_n(m) - 1$ 4. $p \leftarrow L(p \oplus m_i)$ 5. **fin para** 6. $p \leftarrow p \oplus m_{\eta_n(m)}$ 7. **return** (p)

Figura 5.1: En esta figura se muestran las Funciones de cifrado (arriba a la izquierda) y descifrado (arriba a la derecha) de SCDAE, así como también la función $P_{U,L}(\cdot,\cdot)$ (abajo a la izquierda) y la función $f_L(\cdot)$ (abajo a la derecha).

ción polinomial de dos variables, particularmente U también es utilizada dentro del esquema una vez más en la función de cifrado, esto con la intención de poder facilitar la demostración de la seguridad de SCDAE. La tercera llave K puede verse como la llave del cifrador de flujo, ya que es utilizada sólo como llave de este. La construcción hace uso de un cifrador de flujo $SC : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^\ell$. SC toma una llave de n bits y un vector de inicialización de tamaño n y produce una cadena de longitud ℓ. Se asume que ℓ es lo suficientemente larga. También se supone que SC cumple con que para cualquier $K \in \mathcal{K}$, SC_K es una función pseudo aleatoria.

Las funciones de cifrado y descifrado se describen en la Fig. 5.1. Ambas funciones, la de cifrado y descifrado, usan dos funciones f y P las cuales son descritas en la misma figura.

SCDAE puede dividirse en dos grandes bloques, como los otros esquemas CAD existentes, el primero es la creación del vector inicial del algoritmo de cifrado de solo privacidad, junto con el MAC, y el segundo es la función de cifrado en sí.

La creación del vector inicial se realizará a su vez mediante dos funciones, una evaluación polinomial en dos variables y el cifrador de flujo con vector inicial. Primero se realizará la evaluación polinomial de dos variables de las cabeceras y del mensaje en claro. El resultado será utilizado como vector inicial del cifrador de flujo, de esta forma se obtiene una cadena s de longitud ℓ, pero el código de autenticación debe de ser de n bits, por lo que solo se tomarán los primeros n bits mediante la función MSB_n. De esta forma se obtendrá el código de autenticación $T \leftarrow \mathsf{MSB}_n(SC_k(\gamma))$. El vector inicial τ de la función de cifrado se obtiene al realizar la operación XOR entre T y la llave U.

La función de cifrado es simple, primero se genera una cadena t de bits de longitud ℓ al utilizar el cifrador de flujo $SC_k(\tau)$, después se obtiene la cadena R mediante $\mathsf{MSB}_n(t)$. Finalmente se obtiene el mensaje cifrado C al realizar la operación xor entre el mensaje M y la cadena R.

5.2. Complejidad computacional

Por simplicidad se comenzará el análisis de complejidad computacional del algoritmo a partir de sus partes básicas para culminar con la complejidad total del algoritmo de cifrado.

La evaluación polinomial de una variable $f_L(M)$ realiza $\eta_n(M)-1$ multiplicaciones y $\eta_n(M)$ operaciones XOR de n bits.

La evaluación polinomial $P_{U,L}(\vec{H}, M)$ realiza $||\vec{H}|| + 1$ evaluaciones polinomiales de una variable, $||\vec{H}||$ multiplicaciones y $||\vec{H}|| - 1$ operaciones XOR. Si suponemos que $\eta_n(H_i) \leq m$ para todo $1 \leq i \leq h$, además que $\eta_n(M) \leq m$, es decir, la cantidad máxima de bloques que tiene una cadena de la cabecera o el mensaje es m, entonces las multiplicaciones realizadas serían $m \cdot (||\vec{H}|| + 1)$, mientras que la cantidad de operaciones XOR de n bits sería $m \cdot (|\vec{H}|)$.

El algoritmo de cifrado hace una evaluación polinomial de dos variables, lo cual nos da una cantidad $m \cdot (||\vec{H}|| + 1)$ multiplicaciones y $m \cdot (||\vec{H}||)$ operaciones XOR.

Para la creación del MAC se inicializa el cifrador de flujo, denotaremos el tiempo que tarda el cifrador de flujo en comenzar a proporcionar bits pseudo aleatorios por I_{SC}, también se debe de considerar el tiempo que necesita el cifrador de flujo para generar los bits que le pedimos, este tiempo será directamente dependiente del tamaño de bloque y de la longitud del mensaje a cifrar, en este caso n, y lo denotaremos por la función $T_{SC}(n)$.

Después hay que realizar una operación XOR entre el MAC y la llave U para obtener el vector inicial del cifrador de flujo.

El cifrado de la información requiere que se inicialice nuevamente el cifrador de flujo, para después generar una cadena de bits de longitud M, por lo que requeriría $T_{SC}(|M|)$. Finalmente se requiere de una operación XOR de longitud $|M|$, por lo que se considerará como m operaciones XOR de longitud n. Finalmente, si denotamos el tiempo de una multiplicación como T_{mul} y de una operación XOR con operandos de n bits como T_{XOR} entonces podemos decir que el tiempo que tarda SCDAE en cifrar

un mensaje es

$$m \cdot (||\vec{H}|| + 1) \cdot T_{mul} + m \cdot (||\vec{H}|| + 1) \cdot T_{XOR} + 2 \cdot I_{SC} + T_{SC}(|M|) + T_{SC}(n).$$

5.2.1. Comparación con BTM y HBS

Primero analizaremos al esquema HBS. Este esquema recibe como entrada una cadena H de bits como encabezado (todos los demás esquemas CAD reciben un vector de cadenas de bits como encabeado) y una cadena M de bits como mensaje. A continuación se analizarán las partes del esquema CAD una a una. Dado que el esquema HBS hace uso de un cifrador por bloques es necesario definir una variable que exprese el tiempo que dicho cifrador tarda en cifrar un bloque de información, dicha variable será T_π.

Ahora, lo primero que realiza el esquema HBS es generar la llave L, esto llevará T_π tiempo.

Después se realizan dos evaluaciones polinomiales sobre una variable, suponiendo que $\eta_n(H) \leq m$ como $\eta_n(M) \leq m$, entonces estas evaluaciones polinomiales usarán $2m + 1$ operaciones XOR, $2m$ multiplicaciones y a lo más 3 multiplicaciones por 2.

A continuación se realiza el cifrado del mensaje mediante el uso del modo contador de HBS, este modo contador realiza $2m$ operaciones XOR, $m - 1$ incrementos de enteros de n bits y m llamados al cifrador por bloques. El tiempo que tarda un incremento de un entero de n bits será denominado T_{inc}.

Finalmente el cálculo del MAC se realiza con una llamada al cifrador por bloques lo que toma un tiempo T_π.

Sumando todos estos elementos se puede ver que el tiempo que tarda HBS en cifrar un mensaje es:

$$2 \cdot m \cdot T_{mul} + (4 \cdot m + 1) \cdot T_{XOR} + (|M| - 1) \cdot T_{\text{inc}} + (|M| + 2) \cdot T_\pi.$$

Debido a que HBS sólo utiliza encabezados de una sola cadena fijaremos $||\vec{H}|| = 1$

en SCDAE, de esta forma el tiempo que tarda SCDAE sería

$$2 \cdot m \cdot T_{mul} + 2 \cdot m \cdot T_{XOR} + 2 \cdot I_{SC} + T_{SC}(|M|) + T_{SC}(n).$$

Como se puede ver, ambos esquemas hacen uso de la misma cantidad de multiplicaciones, y aunque SCDAE realiza una cantidad menor de operaciones XOR, es común que el tiempo de esta operación sea considerada como despreciable. La primera diferencia considerable entre HBS y SCDAE es el incremento de enteros de n bits en HBS, el tiempo $T_{\text{inc}}(n)$ está directamente relacionado con el tamaño de n. La otra diferencia entre estos esquemas radica en el tiempo utilizado por sus respectivas primitivas de cifrado en cifrar un mensaje de longitud M.

Bajo la suposición de que los cifradores de flujo cifran datos a mayor velocidad que los cifradores por bloques, se puede concluir que SCDAE sería más rápido que HBS.

Se analizará a BTM de la misma forma que se han analizado los esquemas anteriores, primero se analizará cada uno de los elementos del algoritmo de cifrado, para finalmente sumar el tiempo requerido por todos ellos.

La generación de llaves U, L se realiza con el cifrador por bloques, por lo que esta requiere de un tiempo $2 \cdot T_{\pi}$.

La evaluación polinomial de una variable $f_L(M)$ realiza $\eta_n(M)-1$ multiplicaciones y $\eta_n(M)$ operaciones XOR de n bits.

La evaluación polinomial $F_{U,L}(\vec{H}, M)$ realiza $||\vec{H}|| + 1$ evaluaciones polinomiales de una variable, $||\vec{H}||$ multiplicaciones y $||\vec{H}|| - 1$ operaciones XOR. Si suponemos que $\eta_n(H_i) \leq m$ para todo $1 \leq i \leq h$, además que $\eta_n(M) \leq m$, es decir, la cantidad máxima de bloques que tiene una cadena de la cabecera o el mensaje es m, entonces las multiplicaciones realizadas serían $m \cdot (||\vec{H}|| + 1)$, mientras que la cantidad de operaciones XOR de n bits sería $m \cdot (|\vec{H}|)$.

Para generar el vector inicial del modo contador es necesario realizar una operación Tagmixing ($\boxplus$) lo cual es equivalente a dos sumas de enteros de $\frac{n}{2}$ bits.

El modo contador realiza $|M|$ llamadas al cifrador por bloques, así como $|M|$

llamadas a Tagmixing ($\boxplus$), o dicho de otra forma $2 \cdot |M|$ sumas de enteros de $\frac{n}{2}$ bits, así como $|M|$ operaciones XOR.

Si definimos el tiempo de una suma entera de $\frac{n}{2}$ bits como T_{int} entonces el tiempo máximo que requiere BTM en cifrar un mensaje sería:

$$m \cdot (||\vec{H}|| + 1) \cdot T_{mul} + m \cdot (||\vec{H}|| + 1) \cdot T_{XOR} + (2 + |M|) \cdot T_{\pi} + (2 \cdot |M| + 2) \cdot T_{\text{int}}$$

comparado con el tiempo que podría tardar SCDAE:

$$m \cdot (||\vec{H}|| + 1) \cdot T_{mul} + m \cdot (||\vec{H}|| + 1) \cdot T_{XOR} + 2 \cdot I_{SC} + T_{SC}(|M|) + T_{SC}(n).$$

En este caso es más evidente cuales son las diferencias entre ambos esquemas DAE, dado que el tiempo necesario para realizar sus multiplicaciones y sus operaciones XOR son las mismas. Se puede ver que BTM requiere un tiempo extra $(2 \cdot |M| + 2) \cdot T_{\text{int}}$ para cifrar un mensaje $|M|$. La siguiente diferencia, nuevamente, es dependiente de el tiempo que tarde cada una de las primitivas de cifrado en procesar un mensaje de longitud $|M|$.

Nuevamente bajo la suposición de que un cifrador de flujo es más veloz que un cifrador por bloque, se puede concluir que el esquema SCDAE sería más eficiente en términos de velocidad.

5.3. Seguridad del esquema SCDAE

5.3.1. Definiciones y preliminares

Sea F una familia de funciones con domino $\mathcal{X}$ y rango $\mathcal{Y}$ tal que $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ donde $\mathcal{K}$ es el espacio de llaves, sea $\mathcal{A}$ un adversario eficiente, y sea

$$\mathbf{Adv}_F^{\text{prf}}(\mathcal{A}) = \Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{F(K,\cdot)} \Rightarrow 1] - \Pr[\rho \xleftarrow{\$} \text{Func}(\mathcal{X}, \mathcal{Y}) : \mathcal{A}^{\rho(\cdot)} \Rightarrow 1].$$

la ventaja de $\mathcal{A}$ al atacar a F como prf. Entonces F es una prf si $\mathbf{Adv}_F^{\text{prf}}(\mathcal{A})$ es pequeña.

Un objetivo útil cuando se atacan funciones pseudoaleatorias es la *falsificación* (forgery). Para definir la falsificación de una función pseudoaleatoria $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, a un adversario eficiente $\mathcal{A}$ se le concede acceso de oráculo a $F_K(\cdot)$, donde $K \xleftarrow{\$} \mathcal{K}$. Supóngase que $\mathcal{A}$ realiza q consultas X_i, $1 \leq i \leq q$, al oráculo y obtiene $Y_i = F_K(X_i)$, $1 \leq i \leq q$. Al terminar $\mathcal{A}$ intentará realizar una falsificación al producir un par (X, t) tal que $\mathcal{A}$ nunca obtuvo t como resultado de cualquiera de sus consultas previas X_i $(1 \leq i \leq q)$. Se dice que $\mathcal{A}$ tuvo éxito si en efecto $F_K(X) = t$. Es bien conocido que :

$$\Pr[\mathcal{A}^{F_K(\cdot)}\text{forges}] \leq \mathbf{Adv}_F^{\text{prf}}(\mathcal{A}) + \frac{1}{|\mathcal{Y}|}. \tag{5.1}$$

(por ejemplo véase en [31])

Consideramos que nuestro cifrador de flujo $SC : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^\ell$, el cual toma una llave de n bits y un vector de inicialización de n bits, es una función pseudo aleatoria, es decir, para un adversario $\mathcal{A}$ nosotros definimos la ventaja de $\mathcal{A}$ para distinguir SC de una función aleatoria como

$$\mathbf{Adv}_{SC}^{\text{prf}}(\mathcal{A}) = \Pr[K \xleftarrow{\$} \{0,1\}^n : \mathcal{A}^{SC_K(\cdot)} \Rightarrow 1] - \Pr[\varphi \xleftarrow{\$} \text{Func}(n, \ell) : \mathcal{A}^{\varphi(\cdot)} \Rightarrow 1].$$

Y para todo adversario eficiente $\mathcal{A}$, asumimos que $\mathbf{Adv}_{SC}^{\text{prf}}(\mathcal{A})$ es pequeña.

Por $SCDAE_{[f]}$ se querrá decir $SCDAE$ instanciado con un función específica $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^\ell$.

Como se ha mencionado previamente en el capitulo 2, un CAD es considerado seguro si es seguro en términos de privacidad y autenticación. Véase el capitulo 2 por

la definición formal de la seguridad de un CAD.

5.3.2. Cota de seguridad SCDAE

Los siguientes teoremas sugieren que SCDAE es un CAD seguro.

Teorema 1. *Sea* $\delta \xleftarrow{\$} Func(n, \ell)$ *y sea* $\mathcal{A}$ *un adversario eficiente que ataca a SCDAE*$[\delta]$ *en términos de privacidad, entonces:*

$$\mathbf{Adv}^{priv}_{SCDAE[\delta]}(\mathcal{A}) = \frac{2.5q\sigma}{2^{n+1}}$$

donde q *es la cantidad de consultas que* A *realiza a los oráculos y* σ *es la complejidad de dichas consultas. Es decir,* σ *es el total de bloques de* n *bits consultados por el adversario.*

Teorema 2. *Para todo adversario* $\mathcal{A}$ *eficiente de SCDAE en el sentido de autenticación en esquemas CAD se cumple que:*

$$\mathbf{Adv}^{DAE-auth}_{SCDAE_{[\delta]}}(\mathcal{A}) \leq \frac{(q-1)\sigma}{2^n} + 1/2^n.$$

donde q *es el número de consultas realizadas y* σ *es la complejidad de dichas consultas.*

5.3.3. Demostraciones

Para poder realizar la prueba de los teoremas de seguridad es necesario analizar más a fondo la función $P_{U,L}$ como es descrita en la Fig. 5.1.

Una evaluación polinomial en una variable estará definida como una función $f : \{0,1\}^n \times \{0,1\}^{**} \to \{0,1\}^n$ tal que: Dados $L \in \{0,1\}^n$ y $M \in \{0,1\}^{mn}$:

$$f(L, M) = L^m \oplus L^{m-1}M_0 \oplus \cdots \oplus LM_{m-2} \oplus M_{m-1}$$

donde tanto la adición ($\oplus$) como la multiplicación son hechas en el campo GF(2^n)

Una evaluación polinomial en dos variables estará definida como una función $P : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^{**} \rightarrow \{0,1\}^n$ tal que: Dados $U \in \{0,1\}^n$, $L \in \{0,1\}^n$ y $\vec{H} = [H_0, H_1, .., H_{h-1}]$, donde cada $H_i \in \{0,1\}^{nm}$:

$$P(U, L, \vec{H}) = U^h \oplus U^{h-1} f(L, H_0) \oplus \cdots \oplus U f(L, H_{h-2}) \oplus f(L, H_{h-1}).$$

Supongamos que tenemos a $\vec{H} \in \{0,1\}^{**}$, es decir un vector de cadenas de bits, y supongamos que H_α es la cadena de bits de mayor longitud de $\vec{H}$. Entonces definiremos la función $\mu(\vec{H})$ como:

$$\mu(\vec{H}) = \eta(H_\alpha).$$

Lema 1. *Sea* $P : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^{**} \rightarrow \{0,1\}^n$ *una evaluación polinomial de dos variables,* $U, L \in \{0,1\}^n$ *y* $\vec{H}^{(i)}, \vec{H}^{(j)}, \in \{0,1\}^{**}$ *tal que* $\vec{H}^{(i)} = (H_0^{(i)}, H_1^{(i)}, \cdots, H_{h_{(i)}-1}^{(i)})$, $\vec{H}^{(j)} = (H_0^{(j)}, H_1^{(j)}, \cdots, H_{h_{(j)}-1}^{(j)})$ *y* $\vec{H}^{(i)} \neq \vec{H}^{(j)}$. *Y sea CollP el evento:* $F(U, L, \vec{H}^{(i)}) \oplus F(U, L, \vec{H}^{(j)}) = 0$. *Entonces:*

$$\Pr[CollP] \leq \frac{\max(h_{(i)}, h_{(j)}) - 1 + \max(\eta(\vec{H}^{(i)}), \eta(\vec{H}^{(j)}))}{2^n}.$$

Demostración: Sea $\vec{H}^{(i)} \in \{0,1\}^{**}$ entonces $\vec{H}^{(i)} = (H_0^{(i)} + H_1^{(i)} + \cdots + H_{h_{(i)}-1}^{(i)})$ donde $H_\kappa^{(i)} = H_{\kappa,0}^{(i)} || H_{\kappa,1}^{(i)} || \cdots || H_{\kappa,m_{(i),\kappa}-1}^{(i)}$, $0 \leq \kappa \leq h_i - 1$, $|H_{\kappa,\ell}^{(i)}| = n$ bits para $\ell < m_{(i),\kappa} - 1$ y $|H_{\kappa,\ell}^{(i)}| \leq n$ bits para $\ell = m_{(i),\kappa} - 1$.

Definamos a

$$f_L(H_\kappa^{(i)}) = L^{m_{(i),\kappa}} \oplus \sum_{\ell=0}^{m_{(i),\kappa}-1} L^\ell H_{\kappa,\ell}^{(i)}$$

y a

$$P(U, L, \vec{H}^{(i)}) = \sum_{\kappa=0}^{h_{(i)}-1} U^{h_{(i)}-\kappa-1} f_L(H_\kappa^{(i)}).$$

Ahora definamos al polinomio $G(U, L, \vec{H}^{(i)}, \vec{H}^{(j)})$:

$$G(U, L, \vec{H}^{(i)}, \vec{H}^{(j)}) = P(U, L, \vec{H}^{(i)}) \oplus P(U, L, \vec{H}^{(j)})$$

donde G es un polinomio en U de grado a lo más $\max(h_{(i)}, h_{(j)}) - 1$ y donde el coeficiente V_ℓ de U^ℓ sería:

$$V_\ell = f_L(H_{h_{(i)}-1-\ell}^{(i)}) \oplus f_L(H_{h_{(j)}-1-\ell}^{(j)}).$$

De esta forma se puede ver que

$$\Pr[G(U, L, \vec{H}^{(i)}, \vec{H}^{(j)}) = 0] = \Pr[P(U, L, \vec{H}^{(i)}) \oplus P(U, L, \vec{H}^{(j)}) = 0].$$

Ahora,

$$\begin{aligned} \Pr[G(U, L, \vec{H}^{(i)}, \vec{H}^{(j)}) = 0] &= \sum_{\phi \in \{0,1\}^n} \Pr[G(U, L, \vec{H}^{(i)}, \vec{H}^{(j)}) = 0 | L = \phi] \cdot \Pr[L = \phi] \\ &= \frac{1}{2^n} \sum_{\phi \in \{0,1\}^n} \Pr[G(U, L, \vec{H}^{(i)}, \vec{H}^{(j)}) = 0 | L = \phi]. \end{aligned} \quad (5.2)$$

Sea $Z \subseteq \{0,1\}^n$ tal que si $z \in Z$ entonces:

$$f_z(H_{h_{(i)}-1-\ell}^{(i)}) \oplus f_z(H_{h_{(j)}-1-\ell}^{(j)}) = 0$$

para toda ℓ tal que $0 \leq \ell \leq \max\{h_{(i)}, h_{(j)}\} - 1$.

Podemos ver que:

$$\begin{aligned}\Pr[G(U,L,\vec{H}^{(i)},\vec{H}^{(j)})=0] &= \frac{1}{2^n}[\sum_{z\in Z}\Pr[G(U,L,\vec{H}^{(i)},\vec{H}^{(j)})=0|L=z] \\ &+ \sum_{z\notin Z}\Pr[G(U,L,\vec{H}^{(i)},\vec{H}^{(j)})=0|L=z]] \qquad (5.3)\end{aligned}$$

si

$$A=\sum_{z\in Z}\Pr[G(U,L,\vec{H}^{(i)},\vec{H}^{(j)})=0|L=z] \qquad (5.4)$$

y

$$B=\sum_{z\notin Z}\Pr[G(U,L,\vec{H}^{(i)},\vec{H}^{(j)})=0|L=z] \qquad (5.5)$$

entonces de (5.3, (5.4) y (5.5) tenemos

$$\Pr[G(U,L,\vec{H}^{(i)},\vec{H}^{(j)})=0]=\frac{1}{2^n}A+\frac{1}{2^n}B \qquad (5.6)$$

Si $z \in Z$ entonces $G(U, L, H^{(i)}, H^{(j)})$ es un polinomio en U con todos sus coeficientes igual a 0, por lo que

$$\begin{aligned}A &= \sum_{z\in Z}\Pr[G(U,L,\vec{H}^{(i)},\vec{H}^{(j)})=0|L=z] \\ &= \sum_{z\in Z}1 \\ &= |Z|\end{aligned}$$

$|Z|$ es, a lo más, la cantidad de raíces del polinomio de mayor grado en todos los coeficientes de U, es decir máx$\{\mu(H^{(i)}), \mu(H^{(j)})\}$. Por lo que:

$$A \leq \text{máx}\{\mu(H^{(i)}),\mu(H^{(j)})\} \qquad (5.7)$$

La probabilidad de B la calcularemos de la siguiente forma: Dado que $z \notin Z$

entonces $G(U, L, \vec{H}^{(i)}, \vec{H}^{(j)})$ es un polinomio en U con al menos un coeficiente distinto de 0, por lo que

$$\Pr[G(U, L, \vec{H}^{(i)}, \vec{H}^{(j)}) = 0 | z \notin Z] = \frac{\text{máx}\{h_i, h_j\} - 1}{2^n}$$

finalmente

$$\begin{aligned} B &= \sum_{z \notin Z} \frac{\text{máx}\{h_i, h_j\} - 1}{2^n} \\ &= (2^n - \text{máx}\{\mu(H^{(i)}), \mu(H^{(j)})\}) \frac{\text{máx}\{h_i, h_j\} - 1}{2^n} \\ &\leq \text{máx}\{h_i, h_j\} - 1 \end{aligned} \tag{5.8}$$

Sustituyendo (5.7) y (5.8) en (5.6) obtenemos que:

$$\Pr[G(U, L, \vec{H}^{(i)}, \vec{H}^{(j)}) = 0] \leq \frac{\text{máx}\{h_i, h_j\} - 1 + \text{máx}\{\mu(\vec{H}^{(i)}), \mu(\vec{H}^{(j)})\}}{2^n}$$

Quedando demostrado el lema 1. □

Lema 2. *Sea* $P : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^{**} \to \{0,1\}^n$ *una evaluación polinomial de dos variables ,* $U, L \in \{0,1\}^n$*, sean* $\vec{H}^{(1)}, \vec{H}^{(2)}, \cdots, \vec{H}^{(q)} \in \{0,1\}^{**}$ q *vectores de cadenas tal que* $\vec{H}^{(i)} \neq \vec{H}^{(j)}$ *para todo* $1 \leq i < j \leq q$ *y sea* E *el evento; existe un par* $\vec{H}^{(i)}$, $\vec{H}^{(j)}$ *tal que* $P(U, L, \vec{H}^{(i)}) \oplus P(U, L, \vec{H}^{(j)}) = 0$*. Entonces:*

$$\Pr[E] \leq \frac{(q-1)\sigma}{2^n}$$

donde σ *es la complejidad de las consultas.*

Demostración: Sean Sea $P : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^{**} \to \{0,1\}^n$ una evaluación polinomial de dos variables, $U, L \in \{0,1\}^n$, sean $\vec{H}^{(1)}, \vec{H}^{(2)}, \cdots, \vec{H}^{(q)} \in \{0,1\}^{**}$ q vectores de cadenas, tal que $\vec{H}^{(i)} \neq \vec{H}^{(j)}$ para todo $1 \leq i < j \leq q$.

Definamos a

$$f_L(H_\kappa^{(i)}) = L^{m_{(i),\kappa}} \oplus \sum_{\ell=0}^{m_{(i),\kappa}-1} L^\ell H_{\kappa,\ell}^{(i)}$$

y a

$$P(U, L, \vec{H}^{(i)}) = \sum_{\kappa=0}^{h_{(i)}-1} U^{h_{(i)}-\kappa-1} f_L(H_\kappa^{(i)}).$$

Además definamos la variable d como:

$$d = \text{máx}\{||\vec{H}^{(1)}||, ||\vec{H}^{(2)}||, \cdots, ||\vec{H}^{(q)}||\} - 1.$$

Mientras que la variable c será definida como:

$$c = \text{máx}\{\mu(\vec{H}^{(1)}), \mu(\vec{H}^{(2)}), \cdots, \mu(\vec{H}^{(q)})\}.$$

Y finalmente definiremos la variable λ como:

$$\lambda = \mu(\vec{H}^{(1)}) + \mu(\vec{H}^{(2)}) + \cdots + \mu(\vec{H}^{(q)}).$$

Por el lema 1 tenemos que:

$$\begin{aligned} \Pr[P(U, L, \vec{H}^{(i)}) \oplus P(P, L, \vec{H}^{(j)}) = 0] &\leq \frac{\text{máx}\{h_i, h_j\} - 1 + \text{máx}\{\mu(\vec{H}^{(i)}), \mu(\vec{H}^{(j)})\}}{2^n} \\ &\leq \frac{d + c}{2^n} \end{aligned} \tag{5.9}$$

para todas las $1 \leq i < j \leq q$. Por lo que:

$$\begin{aligned} \Pr[E] &\leq \binom{q}{2} \frac{d + c}{2^n} \\ &\leq \binom{q}{2} \frac{\lambda}{2^n} \\ &= \frac{(q-1)q}{2} \cdot \frac{\lambda}{2^n} \\ &\leq \frac{(q-1)q\lambda}{2^n} \\ &\leq \frac{(q-1)\sigma}{2^n} \end{aligned}$$

Figura 5.2: SCDAE$_\delta$: $\mathcal{E}_{K,U,L}(\cdot,\cdot)$

SCDAE$_{[\delta]}$.$\mathcal{E}_{K,U,L}(H, M)$

1. $\gamma \leftarrow P_{U,L}([\vec{H}, M])$
2. $T \leftarrow \mathsf{MSB}_n(\delta(\gamma))$
3. $\tau \leftarrow T \oplus U$
4. $R \leftarrow \mathsf{MSB}_{|M|}(\delta(\tau))$
5. $C \leftarrow M \oplus R$
6. **return** (T, C)

quedando demostrado el lema 2. □

5.4. Demostración del Teorema 1

Primero, para la prueba de seguridad de SCDAE se remplazará el cifrador de flujo SC con la función $\delta \xleftarrow{\$} \text{Func}(n, \ell)$ en 5.1, con ℓ lo suficientemente grande, lo que nos dará como resultado el algoritmo mostrado en la Fig. 5.2.

Para demostrar el Teorema 1 se seguirá una técnica basada en juegos. La ventaja de $\mathcal{A}$ contra SCDAE$_\delta$ en términos de privacidad es:

$$\mathbf{Adv}^{priv}_{SCDAE_\delta}(\mathcal{A}) = \Pr\left[\delta \xleftarrow{\$} \text{Func}(n, m) : \mathcal{A}^{\mathcal{E}_{K,U,L}(.,.,.)} \Rightarrow 1\right] - \Pr[\mathcal{A}^{\$(.,.,.)} \Rightarrow 1].$$

Para la demostración necesitamos hacer uso de una secuencia de juegos. Comenzaremos con el juego 1 el cual es mostrado en la Fig. 5.3. El juego 1 emula el algoritmo presentado en Fig. 5.2. Entonces tenemos:

$$\Pr[\mathcal{A}^{juego1} \Rightarrow 1] = \Pr[\delta \xleftarrow{\$} \text{Func}(n, \ell) : \mathcal{A}^{\delta(\cdot,\cdot)} \Rightarrow 1]. \tag{5.10}$$

A continuación se realiza un pequeño cambio en el juego 1 al eliminar las partes enmarcadas por rectángulos en Fig. 5.2 para obtener el juego 2. En el juego 2, se debe de notar que el adversario siempre obtiene cadenas aleatorias, por lo que tendríamos:

Figura 5.3: Juego 1 y 2.

$get\delta(x)$
1. $Y \xleftarrow{\$} \{0,1\}^{\ell}$ 2. **if** $x \in D$ **then** $BAD \leftarrow true$ 3. $\boxed{Y \leftarrow \delta(x)}$ 4. **endif** 5. $D \leftarrow D \cup \{x\}$ 6. $\delta(x) \leftarrow Y$ 7. **return**Y
Inicialización
1. $D \leftarrow \{\}$ 2. $BAD \leftarrow false$
$\mathcal{E}_{K,U,L}([\vec{H}, M])$
1. $\gamma^s \leftarrow P_{U,L}([\vec{H}^s, M^s])$ 2. $T^s \leftarrow \mathsf{MSB}_n(get\delta(\gamma^s))$ 3. $\tau^s \leftarrow T^s \oplus U$ 4. $R^s \leftarrow \mathsf{MSB}_{\lvert M^s \rvert}(get\delta(\tau^s))$ 5. $C^s \leftarrow M^s \oplus R^s$ 6. **return** (T^s, C^s)

Figura 5.4: Juego 3

En consulta s (con parámetros $\vec{H}^s$ y M^s) del adversario se realiza lo siguiente
1. $T^s \xleftarrow{\$} \{0,1\}^n$ 3. $C^s \xleftarrow{\$} \{0,1\}^{\lvert M^s \rvert}$ 6. **return** (T^s, C^s)
Fase 2
1. $\gamma^s \leftarrow P_{U,L}([\vec{H}^s, M^s])$ 2. $D \leftarrow D \cup \{\gamma^s\}$ 3. $D \leftarrow \cup\{T^s \oplus U\}$
Finalización. Si existe una colisión en D entonces: $BAD \leftarrow TRUE$

$$\Pr[\mathcal{A}^{juego2} \Rightarrow 1] = Pr[\mathcal{A}^{\$(\cdot,\cdot)} \Rightarrow 1]. \tag{5.11}$$

Si realizamos unos pequeños cambios sintácticos al juego 2 obtendremos el juego 3 (mostrado en la Fig. 5.4). En el juego 3 entregaremos cadenas aleatorias al adversario en cada consulta. Después, en la fase 2, pondremos la variable apropiada en el multiconjunto D. Finalmente buscaremos una colisión en D y si esta existe se otorgará el valor de verdadero a la bandera BAD. Debido a esto:

$$\Pr[\mathcal{A}^{juego2} \Rightarrow 1] = \Pr[\mathcal{A}^{juego3} \Rightarrow 1] \tag{5.12}$$

además

$$\Pr[\mathcal{A}^{juego2} \text{ active } BAD] = \Pr[\mathcal{A}^{juego3} \text{ active } BAD]. \tag{5.13}$$

De (5.10) y (5.11) obtenemos:

$$\Pr[\delta \xleftarrow{\$} \text{Func}(n,\ell) : \mathcal{A}^{\mathcal{E}_\delta(\cdot,\cdot)} \Rightarrow 1] - \Pr[A^{\delta(\cdot,\cdot} \Rightarrow 1] = \Pr[\mathcal{A}^{juego1} \Rightarrow 1] - \Pr[\mathcal{A}^{juego2} \Rightarrow 1].$$

Por lo que:

$$\mathbf{Adv}^{DAE-priv}_{SCDAE[\delta]} = \Pr[\mathcal{A}^{juego1} \Rightarrow 1] - \Pr[\mathcal{A}^{juego2} \Rightarrow 1]. \tag{5.14}$$

Ahora, juego 1 y juego 2 se ejecutan de la misma manera hasta que la bandera BAD es activada. Por esto tenemos que:

$$\Pr[A^{juego1} \Rightarrow 1] - Pr[\mathcal{A}^{juego2} \Rightarrow 1] \leq \Pr[A^{juego2} \text{ active } BAD].$$

De (5.14) y de (5.13) obtenemos:

$$\mathbf{Adv}^{DAE-priv}_{SCDAE[\delta]}(\mathcal{A}) \leq \Pr[\mathcal{A}^{juego3} \text{ active } BAD].$$

Ahora se analizará la probabilidad de colisión en el multiconjunto D del juego 3. Los elementos de D son:

$$D = \{\gamma^{(s)} : 1 \leq s \leq q\} \cup \{T^{(s)} \oplus U : 1 \leq s \leq q\}.$$

Sea CollD el evento: Existe una colisión en D. Los posibles casos de colisión serían:

1. Cuando en dos consultas distintas, y por lo tanto con dos entradas distintas, la evaluación polinomial regresa el mismo resultado: $\frac{\sigma(q-1)}{2^n}$

2. Cuando en alguna consulta s el valor obtenido en la linea 3 ($\tau^{(s)} = T^{(s)} \oplus U$) es igual al resultado de de la evaluación polinomial en alguna consulta: $\frac{q^2}{2^n}$.

3. Cuando en alguna consulta s el valor obtenido en la linea 3 es igual al valor obtenido en la misma linea 3, pero en una consulta distinta: $\binom{q}{2}\frac{1}{2^n}$

$$\begin{aligned}
\Pr[\text{CollD}] &= \frac{\sigma(q-1)}{2^n} + \frac{q^2}{2^n} + \binom{q}{2}\frac{1}{2^n} \\
&= \frac{2\sigma(q-1)}{2^{n+1}} + \frac{2(q^2)}{2^{n+1}} + \frac{q^2-q}{2^{n+1}} \\
&= \frac{2\sigma q - 2\sigma + 3q^2 - q}{2^{n+1}} \\
&\leq \frac{2q\sigma + 3q^2}{2^{n+1}} \\
&\leq \frac{2q\sigma + 3q\sigma}{2^{n+1}} \\
&\leq \frac{2.5q\sigma}{2^n}.
\end{aligned}$$

Quedando demostrado el Teorema 1. □

5.5. Demostración del Teorema 2

Para facilitar la demostración, es necesario definir y probar el siguiente lema:

Lema 3. *Dado* $\delta \xleftarrow{\$} Func(n,\ell)$ *,* $U, L \xleftarrow{\$} \{0,1\}^n$*,* $X \in \{0,1\}^{n\cdot m}$*,*$\vec{H} \in \{0,1\}^{**}$ *definiremos* $G_{\delta,U,L}(\vec{H}, X) = \mathsf{MSB}_n(\delta(P_{U,L}(\vec{H}, X)))$*. Para cualquier adversario* $\mathcal{B}$ *que realice* q *consultas, definiremos las probabilidades* P_1, P_2 *como:*

$$P_1 = \Pr\left[U, L \xleftarrow{\$} \{0,1\}^n, \delta \xleftarrow{\$} Func(n,\ell) : \mathcal{B}^{g_{\delta,U,L}(\cdot)} \Rightarrow 1\right]$$

$$P_2 = \Pr\left[\mathcal{B}^{\$(\cdot)} \Rightarrow 1\right]$$

entonces:

$$P_1 - P_2 \leq \frac{(q-1)\sigma}{2^n}.$$

Demostración: Esta demostración también se realizará haciendo uso de juegos. El juego 4 mostrado en la Fig. 5.5 representa el comportamiento de $G_{\delta,U,L}$ tal como se ha descrito anteriormente. Por lo que:

$$P_1 = \Pr[\mathcal{B}^{juego4} \Rightarrow 1].$$

El juego 5 se obtiene al eliminar la parte enmarcada por un rectángulo de $get\delta$ en el juego 4. El juego 5 emula el comportamiento del oráculo aleatorio, por lo que:

$$P_2 = \Pr[\mathcal{B}^{juego5} \Rightarrow 1].$$

De aquí podemos obtener que:

$$P_1 - P_2 = \Pr[\mathcal{B}^{juego4} \Rightarrow 1] - Pr[\mathcal{B}^{juego5} \Rightarrow 1].$$

El juego 4 y el juego 5 son idénticos hasta que la bandera BAD es activada, por lo que:

$$\Pr[\mathcal{B}^{juego4} \Rightarrow 1] - \Pr[\mathcal{B}^{juego5} \Rightarrow 1] \leq \Pr[\mathcal{B}^{juego4} \text{ active } BAD].$$

Figura 5.5: Juego 4 y 5.

función $get\delta(x)$
1. $Y \xleftarrow{\$} \{0,1\}^{\ell}$ 2. **if** $x \in D$ **then** $BAD \leftarrow true$ 3. $\boxed{Y \leftarrow \delta(x)}$ 4. **endif** 5. $D \leftarrow D \cup \{x\}$ 6. $\delta(x) \leftarrow Y$ 7. **return** Y
Inicialización
1. $D \leftarrow \{\}$ 2. $BAD \leftarrow false$
$G_{\delta,U,L}([\vec{H}, M])$
1. $S \leftarrow P_{U,L}([\vec{H}, M])$ 2. $H \leftarrow get\delta(S)$ 3. $T \leftarrow \mathsf{MSB}_n(H)$ 4. **return** (T)

Ahora, para que la bandera BAD sea activada es necesario que $get\delta$ reciba 2 veces la misma entrada en consultas distintas, para que esto pase, debe de haber una colisión en la evaluación polinomial de dos variables. Así, podemos deducir usando el lema 2 que:

$$\Pr[\mathcal{B}^{juego4} \text{ active } BAD] = \frac{(q-1)\sigma}{2^n} \tag{5.15}$$

comprobando así el lema 3. □

Lema 4. *Dados* $\delta \xleftarrow{\$} Func(n,\ell)$, $U, L \xleftarrow{\$} \{0,1\}^n$, *definimos a* $G_{\delta,U,L}(\vec{H}, X) = \mathsf{MSB}_n(\delta(P_{U,L}(\vec{H}, X)))$. *Sea* $\mathcal{A}$ *es un adversario que intenta realizar una falsificación de* $G_{\delta,U,L}(\cdot)$ *entonces:*

$$\Pr[\mathcal{A}^{G_{\delta,U,L}(\cdot)} forges] \leq \frac{(q-1)\sigma}{2^n} + \frac{1}{2^n}. \tag{5.16}$$

Demostración: La probabilidad de que $\mathcal{A}$ realice una falsificación de $G_{\delta,U,L}$ esta dada por:

$$\Pr[\mathcal{A}^{G_{\delta,U,L}} \text{ forges}] = \mathbf{Adv}^{\text{prf}}_{G_{\delta,U,L}}(\mathcal{A}) + \frac{1}{2^n}.$$

Y a su vez, si definimos $\zeta \xleftarrow{\$} \text{Func}(\mathcal{H} \times \mathcal{M}, \{0,1\}^\ell)$, esto quiere decir: se escoge una función de manera uniformemente aleatoria del conjunto de funciones que tienen por dominio el espacio de cabeceras producto cartesiano el espacio de mensajes y por codominio las cadenas de bits de longitud ℓ, y se asigna a ζ. La ventaja de un adversario $\mathcal{A}$ al atacar a $G_{\delta,U,L}$ en el sentido de una prf es:

$$\mathbf{Adv}^{\text{prf}}_{G_{\delta,U,L}}(\mathcal{A}) = \Pr[\mathcal{A}^{G_{\delta,L,U}(\cdot)} \Rightarrow 1] - \Pr[\zeta \xleftarrow{\$} \text{Func}(z,\ell) : \mathcal{A}^{\zeta} \Rightarrow 1]. \tag{5.17}$$

Del lema 3 tenemos que:

$$\Pr[\mathcal{A}^{G_{\delta,L,U}(\cdot)} \Rightarrow 1] - \Pr[\mathcal{A}^{\$(\cdot)} \Rightarrow 1] \leq \frac{(q-1)\sigma}{2^n}. \tag{5.18}$$

Si definimos P_3 como:

$$P_3 = \Pr[\mathcal{A}^{\zeta} \Rightarrow 1]. \tag{5.19}$$

Ahora, dado que $\mathcal{A}$ no repite consultas se cumple que:

$$\Pr[\mathcal{A}^{\zeta} \Rightarrow 1] = \Pr[\mathcal{A}^{\$(\cdot)} \Rightarrow 1]. \tag{5.20}$$

Con (5.21) y el lema 3 se obtiene el resultado:

$$\mathbf{Adv}^{\text{prf}}_{G_{\delta,U,L}}(\mathcal{A}) = \frac{(q-1)\sigma}{2^n} \tag{5.21}$$

Finalmente con (5.1) y (5.21) obtenemos:

$$\Pr[\mathcal{A}^{G_{\delta,U,L}(\cdot)} \text{ forges }] \leq \frac{(q-1)\sigma}{2^n} + \frac{1}{2^n}. \tag{5.22}$$

Quedando demostrado el lema 4. □

Demostración del Teorema 2: Para probar el Teorema 2 supondremos la existencia de un adversario $\mathcal{A}$ que ataca la autenticación de $SCDAE$. También consideraremos al adversario $\mathcal{B}$ del lema 3, el cual tiene acceso al oráculo $G_{\delta,U,L}$. A dicho adversario $\mathcal{B}$ se le dará acceso a otro oráculo $R_{\delta,U}(\cdot)$ el cual, cuando recibe de entrada (T), ofrece como resultado $\delta(T \oplus U)$. Con estos dos oráculos $\mathcal{B}$ sería capaz de ejecutar a $\mathcal{A}$ y responder sus consultas de manera adecuada. Cuando $\mathcal{A}$ realice una consulta del tipo $SCDAE.\mathcal{E}_K(\vec{H}, M)$ $\mathcal{B}$ responderá con el par (T,C), donde $T = G_{\delta,U,L}(\vec{H}, M)$ y $C = M \oplus \mathsf{MSB}_{|C|}(R_{\delta,U}(T))$. En caso de que $\mathcal{A}$ realice un consulta del tipo $SCDAE.\mathcal{D}(\vec{H}, T, C)$ $\mathcal{B}$ devolverá la cadena M o el símbolo inválido $\perp$. $\mathcal{B}$ calculará $M = C \oplus \mathsf{MSB}_{|C|}(R_{\delta,U}(T))$, después calculará $T' = G_{\delta,U,L}(\vec{H}, M)$ en caso de que $T = T'$ $\mathcal{B}$ regresa M, en caso de que $T \neq T'$ regresará $\perp$.

Cuando $\mathcal{A}$ consiga realizar una falsificación (T, C), $\mathcal{B}$ realizará $M = C \oplus \mathsf{MSB}_{|C|}(R_{\delta,U}(T))$ y regresará (M, T) como su falsificación. $\mathcal{B}$ siempre es exitoso si $\mathcal{A}$ es exitoso por lo que:

$$\mathbf{Adv}^{DAE-priv}_{SCDAE[\delta]}(\mathcal{A}) = \Pr[A^{SCDAE.\mathcal{E}_{\delta,U,L}(\cdot,\cdot)}\text{forges}] \leq \Pr[B^{G_{\delta,U,L(\cdot)},R_{\delta,U}(\cdot)}\text{forges}]. \quad (5.23)$$

Ahora reemplacemos el oráculo $R_{\delta,U}(\cdot)$ por el oráculo $\$(\cdot)$ el cual regresa un elemento aleatorio de $\{0,1\}^\ell$ cuando se realiza una consulta (T). Dado que $\mathcal{B}$ no tiene permitido repetir una consulta a alguno de sus oráculos, es imposible para $\mathcal{B}$ distinguir entre un oráculo y otro, por lo que:

$$\Pr[B^{G_{\delta,U,L}(\cdot),R_U(\cdot)}\text{forges}] - \Pr[B^{G_{\delta,U,L}(\cdot),\$(\cdot)}\text{forges}] = 0. \quad (5.24)$$

Dado que B bien podría generar sus propias cadenas aleatorias, se puede ver que:

$$\Pr[B^{G_{\delta,U,L}(\cdot),\$(\cdot)}\text{forges}] = \Pr[B^{G_{\delta,U,L}(\cdot)}\text{forges}]. \quad (5.25)$$

A partir de (5.22), (5.24) y (5.25) obtenemos que:

$$\Pr[B^{G_{\delta,U,L}(\cdot),\$(\cdot)}\text{forges}] \leq \frac{(q-1)\sigma}{2^n} + \frac{1}{2^n}. \tag{5.26}$$

Finalmente con (5.23) y (5.26) obtenemos:

$$\mathbf{Adv}^{DAE-auth}_{SCDAE[\delta]}(\mathcal{A}) \leq \frac{(q-1)\sigma}{2^n} + \frac{1}{2^n}.$$

Quedando demostrado el Teorema 2. □

5.6. Sumario

En este capítulo se muestra un nuevo esquema CAD llamado SCADE, dicho esquema tiene la particularidad de no hacer uso de cifradores por bloques, en su lugar hace uso de cifradores de flujo.

El capítulo comienza con una descripción del esquema SCDAE, para después mostrar la complejidad computacional de este en base a la cantidad de operaciones utilizadas y se realiza una comparación con los esquemas HBS y BTM.

Después se analiza la cota de seguridad de SCDAE y se realizan las demostraciones necesarias, destacando los siguientes resultados:

1. Sea $\delta \xleftarrow{\$} Func(n, \ell)$ y sea $\mathcal{A}$ un adversario eficiente que ataca a SCDAE[δ] en términos de privacidad, entonces:

   $$\mathbf{Adv}^{priv}_{SCDAE[\delta]}(\mathcal{A}) = \frac{2.5q\sigma}{2^{n+1}}$$

   donde q es la cantidad de consultas que A realiza a los oráculos y σ es la complejidad de dichas consultas. Es decir, σ es el total de bloques de n bits consultados por el adversario.

2. Para todo adversario $\mathcal{A}$ eficiente de SCDAE en el sentido de autenticación en

esquemas CAD se cumple que:

$$\mathbf{Adv}^{DAE-auth}_{SCDAE_{[\delta]}}(\mathcal{A}) \leq \frac{(q-1)\sigma}{2^n} + 1/2^n.$$

donde q es el número de consultas realizadas y σ es la complejidad de dichas consultas.

Capítulo 6

Conclusiones y Trabajo a Futuro

6.1. Resultados obtenidos

- Se implementó un multiplicador Karatsuba de cuatro etapas en tubería con un buen desempeño. La frecuencia alcanzada (292 MHz). supera a la obtenida por el cifrador por bloques, por lo que podemos considerarlo un multiplicador eficiente.

- Se obtuvieron implementaciones eficientes de los modos de operación BTM y HBS. Es importante decir que estas son las primeras implementaciones reportadas en la literatura.

- Se obtuvieron formas de paralelizar los esquemas CAD para hacerlos más eficientes.

- Se observaron detalles que ayudan a la construcción de CAD eficientes para su implementación en hardware.

- Se ideó un nuevo esquema CAD y su demostración de seguridad.

6.2. Conclusiones

En esta tesis se hace un estudio de los esquemas CAD y se profundiza en dos temas importantes referentes a estos: La implementación eficiente de los esquemas CAD existentes y la propuesta de un nuevo esquema.

En lo que respecta a las implementaciones eficientes de los esquemas CAD, esta tesis presenta tres implementaciones de los esquemas HBS y BTM. A pesar de que las entradas y requerimientos de los esquemas HBS y BTM son distintas, estos contienen los mismos componentes principales: ambos tienen un cifrador por bloques y una función picadillo polinomial. Estos componentes determinan que, para la implementación de estos esquemas, es necesaria la implementación de algún cifrador por bloques y de un multiplicador de campo GF(2^{128}). Dado que AES es el cifrador por bloques mas utilizado en la actualidad, se realizó una implementación eficiente de AES-128 (la especificación del cifrador AES que hace uso de llaves de 128 bits) para usarla en ambos esquemas. Esta implementación de AES-128 tiene un diseño de tubería en diez etapas y es gracias a este diseño que nuestra implementación puede ofrecer, después de los primeros diez ciclos de reloj necesarios para llenar la tubería, un bloque cifrado cada ciclo de reloj con una frecuencia de 291.630 MHz. Para computar la función picadillo polinomial, así como para calcular algunas otras variables internas, se implementó un multiplicador de campo GF(2^{128}). Se requería que dicho multiplicador tuviera una frecuencia similar a la de nuestra implementación de AES-128, por lo que se optó por un diseño de multiplicador en tubería de cuatro etapas, con el cual obtuvimos una frecuencia de 292.320 MHz.

Los principales componentes una función picadillo polinomial son evaluaciones polinomiales. Dichas evaluaciones se pueden computar con m multiplicaciones para un polinomio de grado m si se hace uso de la regla de Horner; pero dado que cada multiplicación en la regla de Horner depende directamente de la multiplicación previa, dicha técnica no se puede realizar directamente en nuestro multiplicador sin una pérdida en el rendimiento. Nuestro multiplicador de 4 estados en tubería divide cada

multiplicación en cuatro etapas distintas e independientes entre si, de tal forma que en cierto punto se puede estar realizando hasta cuatro multiplicaciones diferentes, cada una en una etapa distinta del multiplicador. Para aprovechar el multiplicador la mayoría de los polinomios son vistos como la suma de 4 polinomios distintos de menor tamaño, de tal forma que cada polinomio pequeño puede ser evaluado con la regla de Horner de manera independiente, para finalmente sumar los cuatro polinomios y obtener así el resultado de la evaluación.

Ya con estos componentes implementados se procedió con la implementación de los esquemas antes mencionados, HBS y BTM. Este documento presenta 3 implementaciones distintas, dos de estas, una de HBS y una de BTM, son para mensajes y cabeceras de longitud fija, mientras que la otra es una implementación de BTM capaz de trabajar con múltiples cabeceras de longitud arbitraria, así como mensajes de longitud variable. Las implementaciones para mensajes de cabeceras de longitud fija contemplan mensajes de 4 KB y cabeceras de 128 bits, estas implementaciones están optimizadas para estos tamaños de mensajes y cabeceras, pero son fácilmente escalables para mensajes de cualquier longitud fija. Todas las implementaciones se han realizado teniendo como dispositivo objetivo un Xilinx Virtex 5 XC5VLX85 y los resultados presentados son obtenidos a partir del proceso de "place & route" de la herramienta Xilinx webpack 10.1.

En la segunda parte de esta tesis mostramos a SCDAE, una nueva propuesta de esquema CAD. SCDAE es una propuesta novedosa por el uso de cifradores de flujo con vector inicial, que además presenta un costo computacional aceptable y una cota de seguridad buena. La seguridad de SCDAE está basada en la suposición de que el cifrador de flujo es indistinguible de una función pseudo aleatoria para cualquier adversario computacionalmente óptimo. Esta suposición es utilizada en otros trabajos con cifradores de flujo, como es el caso de [20]. Nuestra demostración afirma que cualquier ataque exitoso a nuestro esquema funcionaría como, o requeriría de, un ataque exitoso al cifrador de flujo sobre el que este construido, por lo que se puede afirmar que si el cifrador de flujo es seguro entonces nuestro esquemas también lo es.

6.3. Trabajo a Futuro

En el trabajo a futuro queda:

- La búsqueda de aplicaciones para los esquemas CAD.
- Realizar implementaciones enfocadas a otros dispositivos, ya sea para mejorar el desempeño en velocidad o en uso de recursos.
- Realizar implementaciones más generales de los esquemas CAD actuales.
- Implementar SCDAE. Es posible que dicha implementación tenga como objetivo hacer el menor uso de recursos posible, ya que existen cifradores de flujo con vector inicial que requieren de pocos recursos, como es el caso de Grain [32] y Mickey [33] por ejemplo.

Bibliografía

[1] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

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

[3] Johan Håstad. The security of the IAPM and IACBC modes. *J. Cryptology*, 20(2):153–163, 2007.

[4] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In Michael K. Reiter and Pierangela Samarati, editors, *ACM Conference on Computer and Communications Security*, pages 196–205. ACM, 2001.

[5] Mihir Bellare, Phillip Rogaway, and D. Wagner. EAX: A conventional authenticated-encryption mode. *IACR Cryptology ePrint Archive*, 2003:69, 2003.

[6] Tadayoshi Kohno, John Viega, and Doug Whiting. CWC: A high-performance conventional authenticated encryption mode. *IACR Cryptology ePrint Archive*, 2003:106, 2003.

[7] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). RFC 3610 (Informational), September 2003.

[8] David A. McGrew and John Viega. The security and performance of the galois/counter mode of operation (full version). *IACR Cryptology ePrint Archive*, 2004:193, 2004.

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

[10] Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 482–499. Springer, 2003.

[11] Debrup Chakraborty and Palash Sarkar. HCH: A new tweakable enciphering scheme using the hash-counter-hash approach. *IEEE Transactions on Information Theory*, 54(4):1683–1699, 2008.

[12] Palash Sarkar. Efficient tweakable enciphering schemes from (block-wise) universal hash functions. *IEEE Transactions on Information Theory*, 55(10):4749–4760, 2009.

[13] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.

[14] Tetsu Iwata and Kan Yasuda. HBS: A single-key mode of operation for deterministic authenticated encryption. In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 394–415. Springer, 2009. 2009.

[15] Tetsu Iwata and Kan Yasuda. BTM: A single-key, inverse-cipher-free mode for deterministic authenticated encryption. In Jr. et al. [34], pages 313–330. 2009.

[16] Debrup Chakraborty, Cuauhtemoc Mancillas-López, Francisco Rodríguez-Henríquez, and Palash Sarkar. Efficient hardware implementations of BRW

polynomials and tweakable enciphering schemes. *IEEE Transactions on Computers (por aparecer) disponible también como: IACR Cryptology ePrint Archive*, 2011:161, 2011.

[17] Rosario Gennaro and Shai Halevi. More on key wrapping. In Jr. et al. [34], pages 53–70. 2009.

[18] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[19] estream: The ecrypt stream cipher project. http://www.ecrypt.eu.org/stream/.

[20] Palash Sarkar. Tweakable enciphering schemes from stream ciphers with iv. *IACR Cryptology ePrint Archive*, 2009:321, 2009.

[21] Pawel Chodowiec and Kris Gaj. Very compact FPGA implementation of the AES algorithm. In Walter et al. [35], pages 319–333. 2003.

[22] Tim Good and Mohammed Benaissa. AES on FPGA from the fastest to the smallest. In Josyula R. Rao and Berk Sunar, editors, *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 427–440. Springer, 2005.

[23] Giacinto Paolo Saggese, Antonino Mazzeo, Nicola Mazzocca, and Antonio G. M. Strollo. An FPGA-based performance analysis of the unrolling, tiling, and pipelining of the AES algorithm. In Peter Y. K. Cheung, George A. Constantinides, and José T. de Sousa, editors, *FPL*, volume 2778 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 2003.

[24] François-Xavier Standaert, Gaël Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. Efficient implementation of rijndael encryption in reconfigurable hardware: Improvements and design tradeoffs. In Walter et al. [35], pages 334–350. 2003.

[25] Serge Vaudenay, editor. *Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings*, volume 5023 of *Lecture Notes in Computer Science*. Springer, 2008.

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

[27] Cuauhtemoc Mancillas-López, Debrup Chakraborty, and Francisco Rodríguez-Henríquez. Reconfigurable hardware implementations of tweakable enciphering schemes. *IEEE Trans. Computers*, 59(11):1547–1561, 2010.

[28] Joan Daemen and Vincent Rijmen. The block cipher rijndael. In Jean-Jacques Quisquater and Bruce Schneier, editors, *CARDIS*, volume 1820 of *Lecture Notes in Computer Science*, pages 277–284. Springer, 1998.

[29] Philippe Bulens, François-Xavier Standaert, Jean-Jacques Quisquater, Pascal Pellegrin, and Gaël Rouvroy. Implementation of the AES-128 on virtex-5 FPGAs. In Vaudenay [25], pages 16–26. 2008.

[30] Cuauhtemoc Mancillas López. Implementación eficiente en hardware reconfigurable de esquemas de cifrado entonados. Master's thesis, Centro de Iinvestigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2007.

[31] Palash Sarkar. Pseudo-random functions and parallelizable modes of operations of a block cipher. *IEEE Transactions on Information Theory*, 56(8):4025–4037, 2010.

[32] Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *IJWMC*, 2(1):86–93, 2007.

[33] Steve Babbage and Matthew Dodd. The MICKEY stream ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 191–209. Springer, 2008.

[34] Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors. *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*. Springer, 2009.

[35] Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*. Springer, 2003.