Trabajo de Fin de Máster
Máster Universitario en Microelectrónica:
Diseño y Aplicaciones de Sistemas
Micro/Nanométrico

Diseño Microelectrónico de Cifradores AEAD con Introducción de Técnicas de Reducción del Consumo de Potencia

Autor: Carlos Fernández García
Tutor: Carlos Jesús Jiménez Fernández
Tutor: Francisco Eugenio Potestad Ordóñez
Fecha: 29 de noviembre de 2022

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

Tras la revolución digital, y desde el comienzo de la era de la información, existe la necesidad de proteger las comunicaciones dado que casi todas las transmisiones contienen información sensible, como por ejemplo información personal o datos bancarios. Para ello se han empleado diferentes estándares de cifrado, como puede ser el DES (Data Encryption Standard) o el AES (Advanced Encryption Standard) el cual es el algoritmo de cifrado más empleado en la actualidad.

El diseño de estos algoritmos tiene como objetivo su uso en plataformas de propósito general como pueden ser ordenadores personales o servidores. Sin embargo, este tipo de algoritmos no proporcionan de forma intrínseca mecanismos que permitan garantizar la autenticidad de la comunicación. Esta característica puede ser añadida mediante un mecanismo externo al algoritmo de cifrado.

El avance tecnológico y el creciente aumento del uso de dispositivos que disponen de una cantidad de recursos limitado, ha hecho crecer la necesidad de nuevos algoritmos de cifrado que consuman menos recursos (*lightweight*), así como de dotar a dichos algoritmos de métodos intrínsecos de autenticación de los mensajes manteniendo un rendimiento adecuado en este tipo de dispositivos con pocos recursos. Así, en los últimos años se han organizado varios proyectos o concursos cuyo objetivo es el de seleccionar o estandarizar algoritmos de cifrado que incluyan un mecanismo de autenticación y que además tengan un consumo de recursos reducido tanto en implementaciones Hardware como Software. Los proyectos más importantes y que serán mencionados a lo largo de este trabajo son: la competición CAESAR (2013-2019) y el proceso de estandarización organizado por el Instituto Nacional de Estándares y Tecnología de los Estados Unidos (NIST), el cual comenzó en el año 2019 y se encuentra actualmente en su fase final, a la que han llegado 10 candidatos.

Estos nuevos algoritmos presentan además una novedad adicional frente a los cifradores tradicionales, como es la inclusión de unas entradas adicionales a las entradas de la clave y del dato a cifrar o descifrar y una salida adicional, la firma (tag), que se utiliza para comprobar la autenticidad del mensaje. Las entradas adicionales son la entrada de "nonce" y de "dato asociado" (Associated Data, AD). La entrada de nonce, aunque no tiene que ser secreta, tiene que ser diferente cada vez que se inicia un proceso de cifrado/descifrado con la misma clave. Sirve para garantizar que datos iguales para cifrar con una misma clave produzcan datos cifrados diferentes. La entrada de dato asociado sirve para incluir en la firma partes del mensaje que, aunque se envíen sin cifrar (como la dirección de destino), requieran formar parte de la firma generada. Es por esto que a este tipo de cifradores se les suele llamar también cifradores AEAD (Authenticated Encryption with Associated Data).

La novedad en el desarrollo de estos algoritmos y la importancia que previsiblemente van a tomar en un futuro inmediato hace crecer la necesidad de tener implementaciones hardware y software eficientes. Sin embargo, el hecho de que todavía el proyecto del NIST no haya definido el algoritmo (o los algoritmos) seleccionados como estándar añade un cierto riesgo a este proceso. Pero la realización de diseños de este tipo de cifradores, aunque finalmente no resulten seleccionados en el proyecto, nos permite tener conocimientos sobre su diseño y, sobre todo sobre su funcionamiento para poder realizar de forma más rápida y eficiente diseños de otros algoritmos.

Con todo esto, el objetivo principal de este trabajo es el diseño hardware eficiente de cifradores AEAD finalistas del proyecto de criptografía lightweight del NIST. En concreto se han seleccionado dos de ellos. El primero es el cifrador Ascon y el segundo es el cifrador TinyJAMBU. El primero de ellos ofrece unos niveles de seguridad por encima de lo mínimo exigido por el proyecto, mientras que el segundo puede considerarse como uno de los que menos recursos van a requerir en su implementación hardware.

Para alcanzar este objetivo se han realizado una serie de tareas expuestas a continuación.

En primer lugar, se ha llevado a cabo el diseño hardware de los algoritmos de cifrado AEAD

Ascon y TinyJAMBU. Las descripciones de estos diseños se han realizado utilizando el lenguaje VHDL y cumpliendo las restricciones impuestas por las principales herramientas de síntesis.

Los diseños han sido verificados utilizando los patrones de test proporcionados junto a las propuestas de cada algoritmo. De esta forma se comprueba que se realicen de forma correcta las operaciones tanto de cifrado como de descifrado mediante la comparación de la salida del diseño y los resultados esperados en los patrones propuestos.

Para el cifrador TinyJAMBU se ha realizado una optimización consistente en la aplicación de una técnica de reducción del consumo de potencia. Se han realizado medidas de simulación para comprobar si se produce una reducción efectiva de este consumo.

También se han realizado implementaciones hardware, tanto en FPGA como en ASIC para analizar los recursos consumidos para todos los diseños realizados.

A lo largo de este documento se expone, junto con una introducción de la criptografía y de los métodos de cifrado y autenticación más empleados hasta la fecha, el principio de funcionamiento y diseño microelectrónico de dos algoritmos de cifrado *lightweigth* con autenticación *Lightweight*, Ascon y TinyJambu, que han llegado hasta la fase final del proceso de estandarización mencionado anteriormente. Además, se proponen varias alternativas de bajo consumo del algoritmo TinyJAMBU junto con su diseño microelectrónico.

En el Capítulo 2 se presenta una breve introducción a la criptografía y su uso a lo largo de la historia. Además, también se hace referencia a los dos tipos de criptografía empleadas en la actualidad haciendo hincapié en la criptografía simétrica, ya que los cifradores diseñados en el marco de este trabajo pertenecen a este tipo de métodos de cifrado. Dentro de la criptografía simétrica se expone el concepto de modo de operación junto con algunos ejemplos de los más utilizados además de las diferentes formas de autenticar una comunicación cifrada. Para finalizar, se exponen los principales procesos de selección y estandarización de algoritmos de cifrado con autenticación así como las causas de su necesidad y sus diferencias frente a los estándares de cifrado hasta la actualidad.

En el Capítulo 3 se describe la especificación del cifrador Ascon y el diseño hardware realizado, con explicación de las entradas y salidas, modos de operación y verificaciones realizadas.

En el Capítulo 4 se hace la misma descripción, pero en este caso del cifrador TinyJAMBU. Se presenta el diseño hardware realizado, la explicación de las entradas y salidas, modos de operación y verificaciones realizadas.

En el Capítulo 5 se presentan propuestas de implementaciones hardware del cifrador TinyJAMBU aplicando una técnica de reducción de consumo de potencia. Esta técnica que consiste en la paralelización de la operación de los desplazamientos en el registro de estados, se aplica de dos formas diferentes y con dos modos de funcionamiento diferentes, dando lugar a cuatro diseños diferentes.

En el Capítulo 6 se presentan los resultados de implementación, tanto en tecnologías FPGA como en tecnologías ASIC de los diseños realizados. Se aportan datos de recursos consumidos, frecuencias máximas de operación estimadas y de consumo de potencia.

Finalmente se extraen las principales conclusiones del trabajo.

CAPÍTULO 2

Marco Teórico

2.1. Introducción a la Criptografía

Desde el comienzo de la civilización el ser humano ha tenido la necesidad de mantener en secreto información sensible y evitar que esta cayese en manos de terceros que pudieran actuar en contra de sus intereses. Partiendo de esta necesidad, a lo largo de la historia se han desarrollado numerosos métodos para garantizar una comunicación y almacenamiento seguro de la información. Se diferencian dos métodos principales para realizar esta tarea, la esteganografía, la cual consiste en ocultar información o mensajes dentro de otros objetos conocidos como portadores, y la criptografía, que consiste en la alteración del mensaje o información con el objetivo de hacerlos irreconocibles para una parte no autorizada de la comunicación que desconozca la forma en la que se ha llevado a cabo dicha alteración.

Algunos ejemplos de sistemas criptográficos empleados a lo largo de la historia son: la escítala lacedemonia (Figura 2.1) empleada por los espartanos para enviar mensajes entre campamentos militares, el cifrado de Julio Cesar que consistía en sustituir cada letra de un mensaje por aquella que le sigue tres posiciones por delante en el alfabeto, la maquina Enigma (Figura 2.2) utilizada en la segunda guerra mundial y llegando hasta uno de los estándares de cifrado de la actualidad como es el cifrador simétrico *AES*.

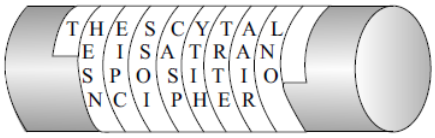


Figura 2.1: Ejemplo de la escítala lacedemonia [1].

Figura 2.2: Fotografía de una máquina Enigma [1].

El principal objetivo de la criptografía es el de proporcionar una transmisión de información segura, entendiendo como segura toda comunicación que garantice la confidencialidad, autenticidad e integridad de la transmisión. La confidencialidad consiste en que el mensaje debe de permanecer secreto excepto para las partes autorizadas de una transmisión que conocen el método de cifrado, la autenticidad hace referencia a la garantía de que un mensaje recibido debe haber sido enviado por una de las partes autorizadas de la comunicación, por último la integridad es

la garantía de que no se ha producido ninguna alteración ni modificación del mensaje durante su transmisión [2].

Destacar que dentro de la criptografía se distingue entre criptografía simétrica, o de clave privada, y criptografía asimétrica, o de clave pública. La criptografía simétrica emplea la misma clave para realizar las operaciones de cifrado y descifrado, mientras que la asimétrica emplea dos claves, una privada que sólo conoce una parte autorizada en la transmisión y que se emplea para descifrar, y una pública derivada de la privada que se envía sin cifrar por el canal de comunicación y es la que se utiliza para cifrar el texto plano.

2.2. Criptografía Asimétrica

La criptografía asimétrica es una rama bastante joven de la criptografía en comparación con la simétrica. Aparece el año 1976 en el cual Whitfield Diffie, Martin Hellman y Ralph Merkle introdujeron los principios básicos de la criptografía de clave pública.

Esta rama surge como la necesidad de paliar algunas de las desventajas de la criptografía de clave privada, en especial el problema derivado de la distribución de las claves y el número de éstas, ya que en principio y considerando el canal de comunicación un canal inseguro se debe de encontrar un método para que las partes autorizadas compartan la clave que se van a utilizar durante la transmisión y además esto se debe de hacer por cada par emisor-receptor.

Así surge la idea de utilizar una clave privada propia de cada parte autorizada de la comunicación y una clave publica derivada de ésta que se utilizará para cifrar los datos. El procedimiento, que se observa en la Figura 2.3, para realizar una transmisión cifrada requiere una primera fase en la que el receptor del mensaje envía al emisor la clave pública con la que éste cifrará los datos que posteriormente serán transmitidos, finalmente el receptor utiliza la clave privada para descifrar este mensaje.

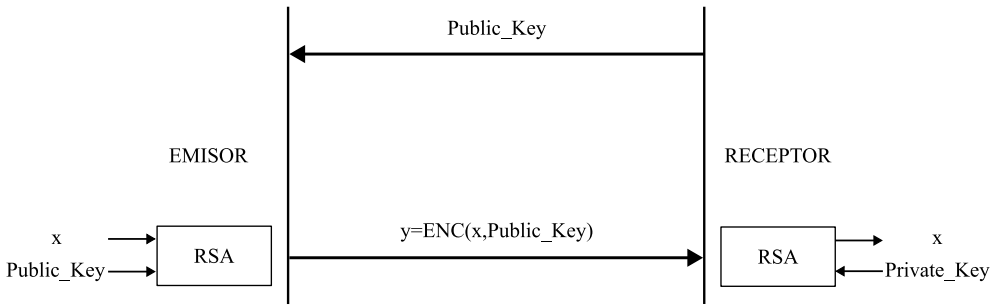


Figura 2.3: Esquema de una comunicación en la que se emplea un algoritmo asimétrico para cifrar el mensaje.

Una desventaja de estos cifradores es su complejidad, ya que debido principalmente a la carga computacional de las operaciones matemáticas en las que se basa su funcionamiento son considerablemente más lentos que los simétricos cuyo principio de operación se basa principalmente en operaciones lógicas como desplazamientos, sustituciones etc.

Por ello muchas veces estos algoritmos se utilizan para distribuir una clave que se utilizará como clave privada de un cifrador simétrico con el que se cifraran todos los mensajes a enviar a través del canal, la Figura 2.4 es un esquema de este proceso.

Algunos ejemplos de este tipo de algoritmos son: el RSA, basado en el problema de factorización de enteros, Elgamal, basado en el problema del logaritmo discreto y los *ECC*(*Eliptic Curve Cryptosystem*).

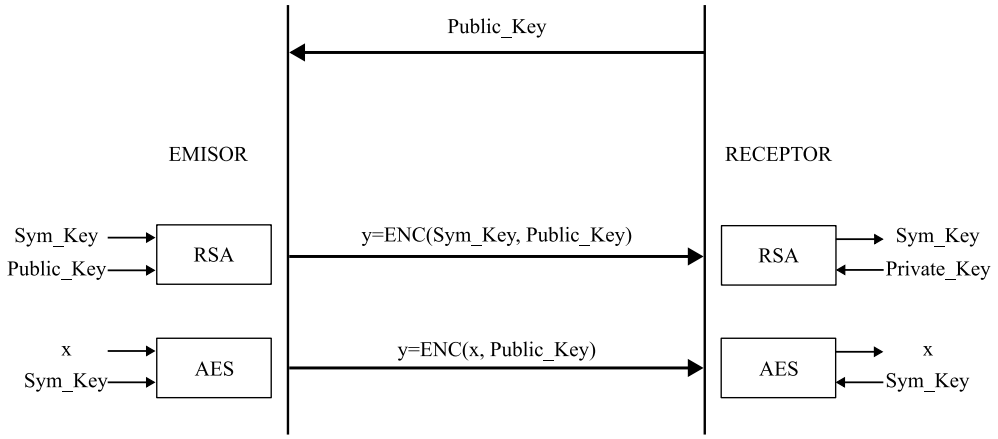


Figura 2.4: Esquema de una comunicación en la que se emplea un algoritmo asimétrico para establecer la clave privada de un cifrador simétrico.

2.3. Criptografía Simétrica

La criptografía simétrica es aquella que partiendo de una clave secreta que sólo es conocida por las partes autorizadas de una comunicación, y un mecanismo que emplea dicha clave, garantiza que la información que se transmite a través de un canal es secreta aunque se tenga acceso a dicho canal excepto para aquellas partes autorizadas en la misma.

Un ejemplo de cifrador simétrico sería el cifrador de sustitución, donde el mecanismo de cifrado sería la sustitución de unos caracteres por otros y la clave sería la tabla de sustitución. Otro ejemplo ya mencionado sería la maquina Enigma utilizada por el ejercito Nazi durante la segunda guerra mundial. En este caso el mecanismo de cifrado serían los rotores que emplea la máquina y la clave la configuración de éstos.

Dentro de la criptografía simétrica se diferencia entre cifradores de bloque y cifradores de flujo, el principio de funcionamiento de cada uno junto con algunos ejemplos son expuestos en las siguientes secciones.

2.3.1. Cifradores de Bloques

En el caso de los cifradores de bloque se cifra un bloque de texto plano usando un mecanismo de cifrado y una clave secreta, de modo que el texto cifrado depende de todos los bits de texto plano y de la clave secreta.

Algunos ejemplos de este tipo de cifradores son el *DES* o su variante *3DES*, o el estándar que se lleva utilizando desde 2001, el *AES*.

DES

El cifrador de bloques *Data Encryption Standard* o *DES*, por sus siglas en inglés, fue uno de los algoritmos de cifrado más usados antes de la aparición del *AES*.[3]

El *DES* es un cifrador seleccionado en el año 1974 por la Oficina Nacional de Estándares de los Estados Unidos (*NBS*), (que actualmente es el Instituto Nacional de Estándares y Tecnología de los Estados Unidos (*NIST*), por sus siglas en ingles). Fue el resultado de un proceso lanzado por esta organización para elegir un estándar de cifrado en este país. Estaba basado en una familia de cifradores anteriores conocida como *Lucifer*, y terminó recibiendo su nombre final

tras la aplicación de algunas modificaciones propuestas por la Agencia de Seguridad Nacional que fue la encargada de analizar la seguridad de las diferentes propuestas.

El DES cifra bloques de 64 bits (8 bytes) empleando una clave de 56 bits (7 bytes). Para cifrar un bloque de entrada son necesarias 16 rondas realizándose en cada una de ellas las misma operaciones. Además, en cada ronda se genera una subclave que se deriva a partir de la clave secreta utilizada. Este mecanismo recibe el nombre de *key schedule* y es bastante común en este tipo de algoritmos de cifrado.

Este cifrador esta basado en una red Feistel, esto significa que las operaciones ejecutadas para realizar el cifrado del texto plano son las mismas y se ejecutan en el mismo orden. La única diferencia entre el proceso de cifrado y descifrado es que cuando se este realizando el proceso de descifrado la subclave utilizada en cada ronda se debe de generar de forma inversa al cifrado.

En la actualidad el cifrador *DES* esta en desuso debido principalmente a que se identificaron vulnerabilidades derivadas de la reducida longitud de su clave secreta, y ha sido sustituido por el siguiente estándar seleccionado por el *NIST* el cifrador *AES*. Además de este último estándar existe una variante más segura del *DES* que se sigue utilizando en la actualidad que recibe el nombre *3DES*, ya que su operación es idéntica a la del *DES* pero se ejecuta 3 veces seguidas.

AES

El *AES* es el cifrador de bloques más usado hasta la época después del abandono del *DES*. Fue estandarizado en el año 2001 tras un proceso de selección lanzado por el *NIST* y su nombre antes de convertirse en estándar era *Rijndael*.[4]

Este cifrador trabaja con bloques de texto de 128 bits (16 bytes) y con una clave secreta cuya longitud se puede seleccionar entre 128 bits, 192 bits y 256 bits. Muy similar al funcionamiento del *DES* este algoritmo requiere de 10, 12 o 14 rondas en función del tamaño de la clave para ejecutar una operación de cifrado, y se debe generar para cada una de estas rondas una subclave que se deriva de la clave secreta empleada por el algoritmo. A diferencia del *DES*, este algoritmo no esta basado en una red Feistel por lo que las operaciones realizadas para el cifrado deben de ser invertidas a la hora de realizar el descifrado.

Un esquema con las diferentes capas que emplea el algoritmo se encuentra en la Figura 2.5. En esta se diferencian dos partes. En la izquierda aparecen las diferentes capas empleadas en cada ronda del algoritmo, mientras que la derecha muestra la generación de la subclave que se emplea en cada ronda del proceso de cifrado.

2.3.2. Cifradores de Flujo

En el caso de un cifrador de flujo la operación de cifrado se realiza mediante una operación, que usualmente consiste en la operación lógica XOR, entre un bit del texto plano y un bit de una çlave"que ha sido generada a partir de la clave secreta del cifrado. Esta çlave"generada es una secuencia pseudoaleatória de bits . Dentro de esta familia de cifradores simétricos existen dos variantes en función de si la clave con la que se cifra el texto plano depende únicamente de la clave secreta del cifrador (Cifradores de flujo síncronos), o si además esta clave depende también del texto cifrado (Cifradores de flujo asíncronos).

Estos cifradores se emplean principalmente en aplicaciones en las que no se disponen de muchos recursos de computación como por ejemplo en sistemas embebidos empleados en redes de sensores o IoT. Algunos ejemplos de cifradores de flujo son el *A5/1* que se emplea para el cifrado de la comunicación por voz en el estándar GSM empleado en comunicaciones móviles, y el *RC4* que se emplea para cifrar comunicaciones a través de internet.

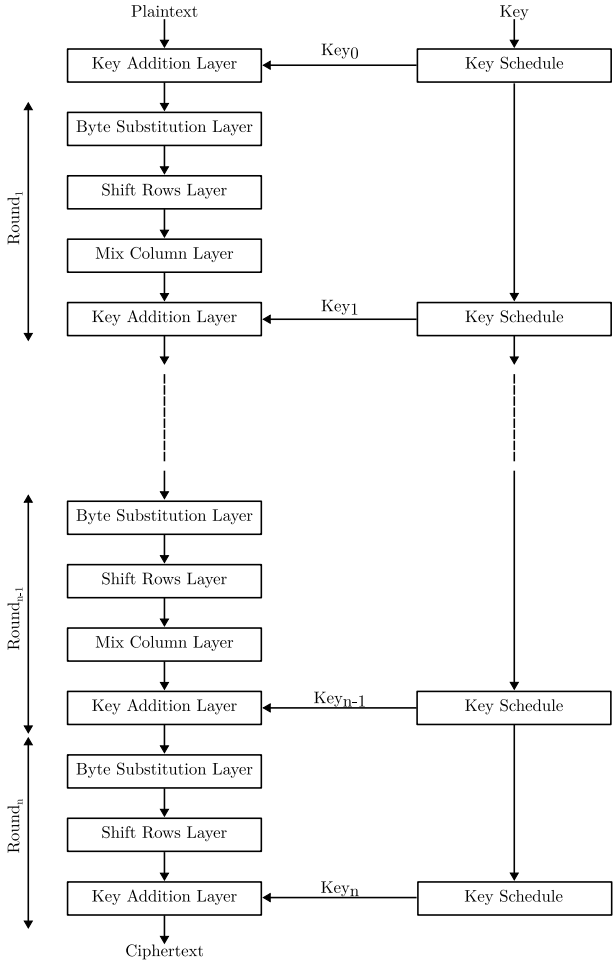


Figura 2.5: Operación del cifrador *AES*.

Trivium

Un ejemplo de cifrador de flujo es el *Trivium* [5], cuya estructura se observa en la Figura 2.6. Es una muestra clara de lo simple y el reducido consumo de recursos de este tipo de algoritmos.

Este cifrador fue uno de los finalistas del proceso de selección del proyecto eSTREAM organizado por *ECRYPT* en el perfil de cifradores de flujo orientados a aplicaciones hardware con un numero de recursos limitado y ha sido especificado como estándar internacional bajo la norma ISO/IEC 29192-3.

Emplea una clave secreta de 80 bits y un Vector de inicialización también de 80 bits. Consta de un estado interno de 288 bits que se divide en 3 NLFSR de 93, 84 y 111 bits conectados entre sí. Su operación se divide en dos fases, un primer periodo de inicialización de los registros en el que tanto la clave como el IV se almacenan en los 80 bits menos significativos del primer y segundo NLFSR respectivamente y se realiza el desplazamiento del registro durante 1152 ciclos. Tras el periodo de inicialización se procede a generar los bits de la clave con los que cifrar el texto plano mediante el desplazamiento de los 3 NLFSR.

2.4. Modos de Operación

Los algoritmos de cifrado simétricos se suelen emplear dentro de un esquema que genera datos cifrados para mensajes cuyo tamaño excede el tamaño del bloque de datos del cifrador. El

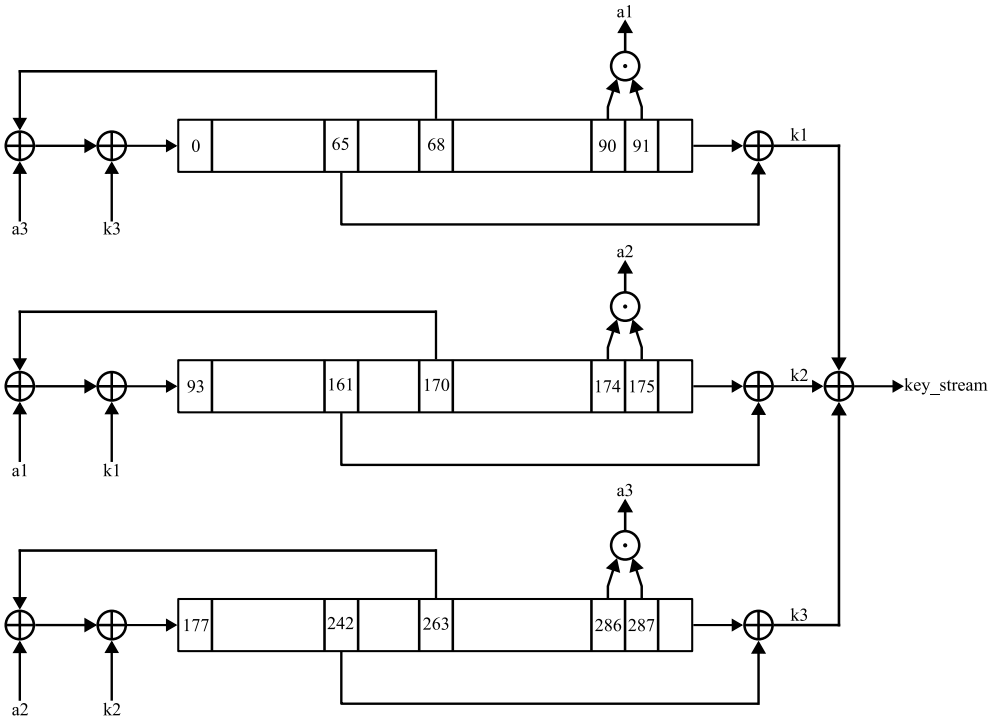


Figura 2.6: Esquema del cifrador de flujo Trivium.

NIST ha proporcionado una serie de recomendaciones para estos esquemas que son los llamados modos de operacion [6].

Entre los modos de operación se puede hacer dos distinciones, dependiendo de si el modo proporciona la capacidad de cifrar datos de longitud arbitraria, como es el caso de los modos *Cipher Feedback Mode*, *Output Feedback Mode* y *Counter Mode*, o esta tiene que ser múltiplo del bus de entrada del cifrador simétrico empleado dentro del modo de operación, como el *Electronic Code Book* y *Cipher Block Chaining Mode*.

En las siguientes paginas se muestran los esquemas de cada modo de operación, en las Figuras 2.7, 2.8, 2.9, 2.10, 2.11 y 2.15, así como una pequeña descripción de su funcionamiento acompañado de algunas de sus ventajas y desventajas más destacadas.

Electronic Code Book (ECB)

Este caso es el mas sencillo y divide el texto a cifrar en **n** bloques cuyo tamaño sera el del bus de entrada del cifrador de bloques empleado en el modo de funcionamiento, así para cifrar el texto se emplea el mismo algoritmo de cifrado con la misma clave **n** veces para generar las **n** salidas de texto cifrado.

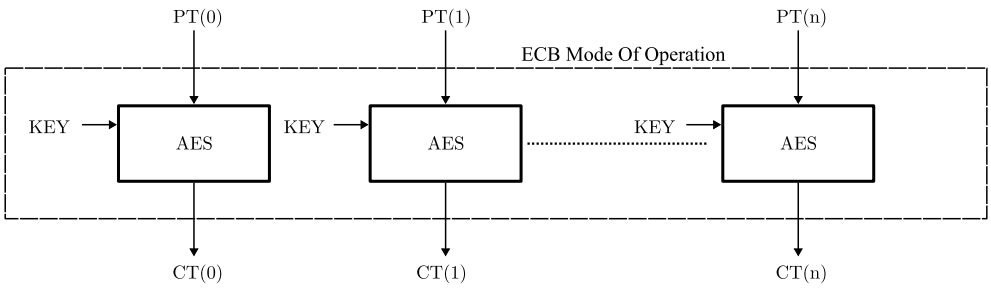


Figura 2.7: Modo de operación *ECB* empleando como bloque básico el cifrador AES.

Entre las ventajas de este modo de operación se encuentra la posibilidad de descifrar cada bloque por separado en caso de que se haya producido algún tipo de problema durante al comunicación y no se haya recibido el texto plano completo o en el caso de que algunos de los bloques este corrupto debido también a fallos durante la transmisión. Además este modo permite paralelizar el cifrado/descifrado de cada bloque mediante la implementación de **n** cifradores en bloque. Esto tiene como resultado la disminución de la latencia del proceso de cifrado pero por el contrario supone un aumento del área que puede llegar a ser prohibitivo.

La principal desventaja de este modo de operación es que mientras que se utilice la misma clave el resultado de cifrar el mismo texto plano será siempre el mismo, lo que puede dar lugar a que un tercero no autorizado en la transmisión obtenga alguna información acerca del texto cifrado con simplemente observar los datos transmitidos a través del canal de comunicación.

Cipher Block Chainig Mode (CBC)

Este modo de operación busca reducir los riesgos derivados de cifrar empleando el modo de operación anterior. Para ello introduce un dato adicional, el **IV** o vector de inicialización el cual se genera de manera pseudoaleatoria cada vez que se realiza una operación de cifrado/descifrado.

El principio de funcionamiento consiste en que la entrada del algoritmo de cifrado es el resultado de la operación XOR entre el bloque de texto plano a cifrar y un vector de inicialización para el caso del primer bloque a cifrar, y para los siguientes bloques a cifrar la XOR entre el texto plano a cifrar y el resultado del cifrado del bloque anterior. Este modo de operación consigue que el resultado de cifrar cada bloque dependa de los bloques cifrados anteriormente, esto junto con el uso de un vector de inicialización cada vez que se realice una transmisión de información hace que los datos transmitidos por el canal sean mas aleatorios a ojos de un posible tercero que este observando dicho canal. Una de las desventajas de este modo de operación es que se pierde

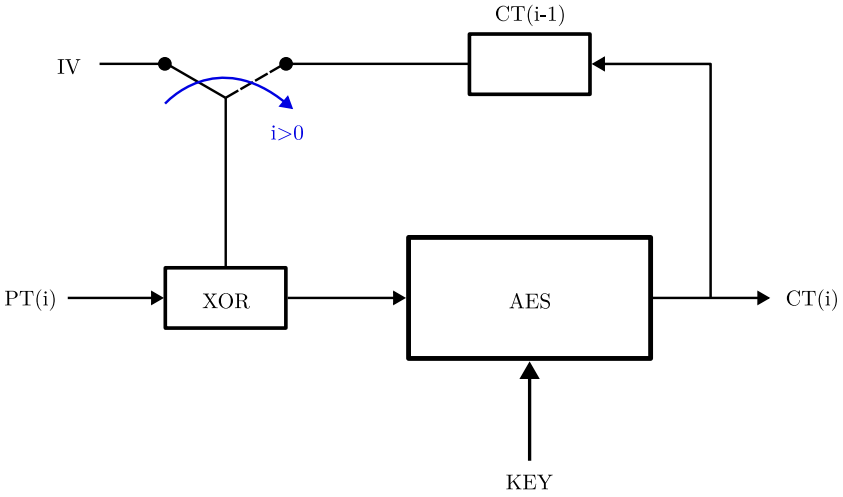


Figura 2.8: Modo de operación *CBC* empleando como bloque básico el cifrador AES.

la posibilidad de paralelización del cifrado de los bloques al depender el cifrado de cada bloque del resultado del cifrado del bloque anterior.

Output Feedback Mode (OFM)

Éste y los siguientes modos de operación cambian un poco la filosofía a la hora de ejecutar las operaciones de cifrado, ya que el texto cifrado no es la salida de un algoritmo de cifrado cuya entrada o bien es un bloque de texto plano o es un dato que depende tanto del bloque de texto plano a cifrar como de bloques anteriores o un vector de inicialización. En este modo la salida

del algoritmo de cifrado se utiliza como una subclave con la cual se cifra el texto plano mediante la aplicación de una operación XOR. Es decir se utiliza el algoritmo de cifrado para construir un cifrador en flujo cuya clave de salida tendrá el tamaño del bus de salida del algoritmo.

En este caso se emplea un **IV** para generar la primera subclave partiendo de la clave a emplear y del algoritmo de cifrado empleado como bloque básico y tras esto las siguientes subclaves son generadas partiendo de la subclave anterior. Finalmente el texto cifrado es el resultado de la operación XOR entre el bloque de texto plano que se este cifrando y la subclave correspondiente que se haya generado.

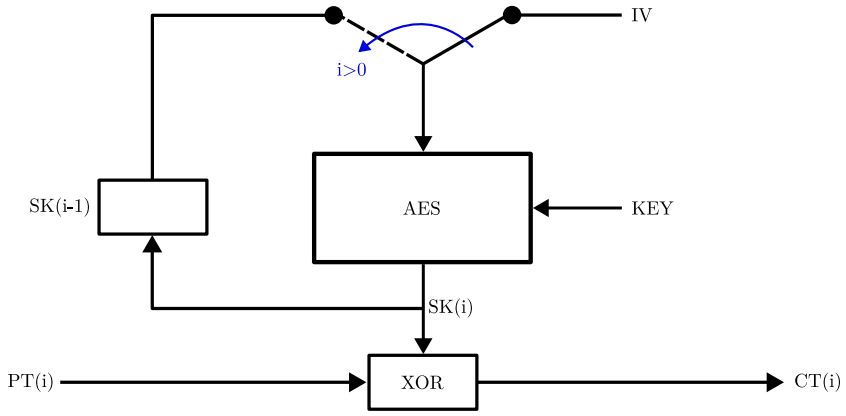


Figura 2.9: Modo de operación *OFM* empleando como bloque básico el cifrador AES.

Este modo de operación permite la precomputación de diferentes subclaves ya que estas no dependen de los bloques de texto plano, a diferencia del modo anterior.

Cipher Feedback Mode (CFM)

El modo de operación CFM es muy similar al descrito anteriormente con la única diferencia de que en este caso se utiliza el bloque de texto cifrado como entrada del algoritmo de cifrado para generar el siguiente valor de subclave. Esto hace que que no exista la posibilidad de precomputar las subclaves como en el caso del modo de operación anterior.

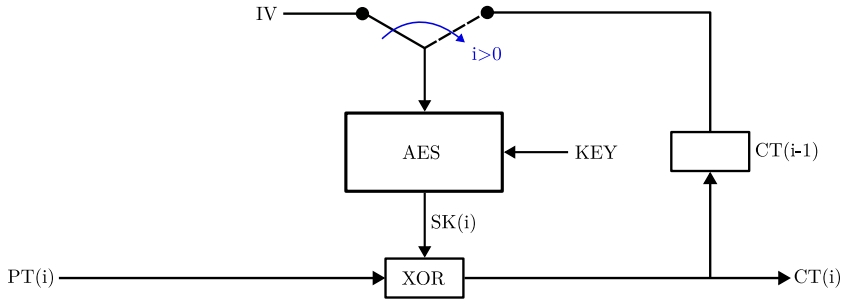


Figura 2.10: Modo de operación *CFM* empleando como bloque básico el cifrador AES.

Counter Mode (CTR)

Otra aproximación que se basa en el principio de utilizar la salida de un algoritmo de cifrado para generar la salida de un cifrador de flujo se basa en utilizar un contador como entrada del cifrador de bloques. El tamaño del contador debe de seleccionarse con cuidado, asegurando siempre que el numero de subclaves que se generan es suficiente antes de que el contador sature y se comience a generar los mismos valores de subclave. En este caso la entrada del algoritmo

de cifrado suele ser la concatenación entre un **IV** y un contador o un LFSR.

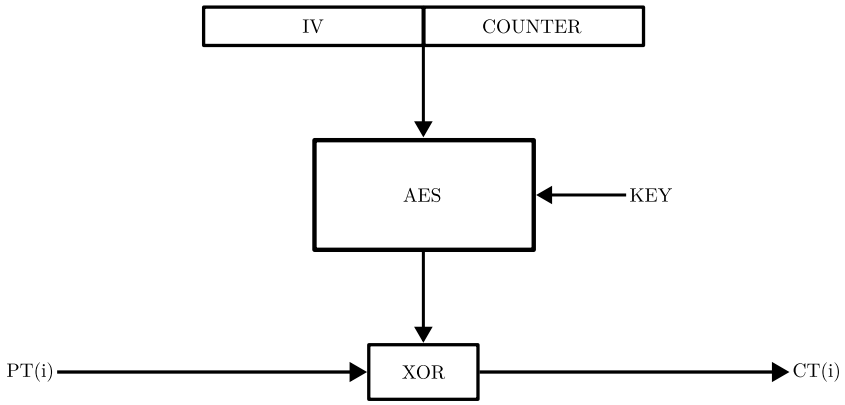


Figura 2.11: Modo de operación *CTR* empleando como bloque básico el cifrador AES.

Una ventaja de este modo de operación es que al no tener ninguna realimentación permite la paralelización del cifrado.

2.5. Mecanismos de autenticación de mensajes

Los algoritmos de cifrado simétricos presentados solo proporcionan confidencialidad a la hora de realizar la operación de cifrado, teniéndose que incluir algún mecanismo adicional para garantizar la autenticidad de la información cifrada transmitida. Este mecanismo consiste en la transmisión de información adicional que se conoce como ***Tags*** de autenticación o códigos de autenticación de mensajes (***MAC*** por sus siglas en ingles).

Existen tres procedimientos diferentes mediante los cuales se pueden generar estos códigos de autenticación, estos dependerán de en que momento del proceso de cifrado se generen y de su dependencia o influencia sobre la información que se esta cifrado en dicho proceso.

Estos tres procedimientos, observados en las Figuras 2.12, 2.13 y 2.14, ordenados de menor a mayor grado de seguridad son:

Encrypt-and-MAC (E&M)

El código de autenticación se genera a partir de la información a cifrar y no se tiene en cuenta durante el proceso de cifrado de esta.

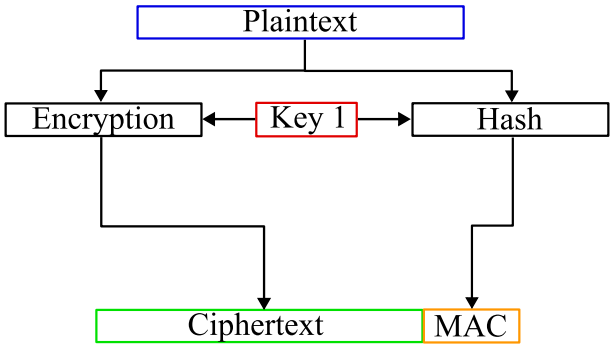


Figura 2.12: Esquema del procedimiento *Encrypt-and-MAC*.

MAC-then-Encrypt (MtE)

Primero se genera el código de autenticación en función de la información a cifrar como en el caso anterior, pero en este caso el proceso de cifrado es dependiente de dicho código.

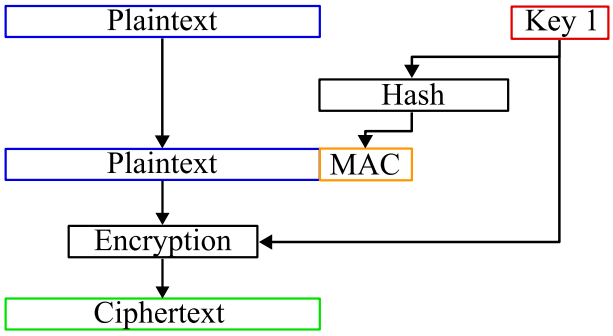


Figura 2.13: Esquema del procedimiento *MAC-then-Encrypt*.

Encrypt-then-MAC (EtM)

En este caso, el código de autenticación se genera tras realizarse el proceso de cifrado de los datos y depende de todo el proceso que se ha ejecutado hasta el momento de su generación.

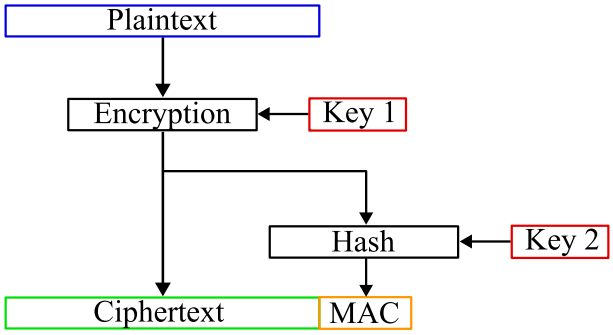


Figura 2.14: Esquema del procedimiento *Encrypt-then-MAC*.

Galois Counter Mode

Entre los modos de operación que además de proporcionar confidencialidad en la transmisión de información también permiten autenticar esta, destaca el modo de operación conocido como Galois Counter Mode.

Este emplea un modo de operación CTR para generar el texto cifrado en combinación con un multiplicador en campo de Galois el cual se utiliza para generar un código de autenticación de mensajes (MAC) siguiendo el procedimiento *EtM* descrito anteriormente. Además, este MAC no depende únicamente del resultado de cifrar un bloque de texto plano sino de un bloque adicional que se envía sin cifrar en la misma transmisión y que recibe el nombre de *Additional Authenticated Data*.

En la Figura 2.15 se muestra como para cifrar los bloques que forman el mensaje se utiliza el modo de operación CTR, y como el MAC se genera como el resultado de la operación XOR entre un valor constante (H) y el resultado de aplicar una multiplicación en campo de Galois entre esa misma constante y un valor intermedio g_n el cual depende del resultado del cifrado de todos los bloques que componen el mensaje y el *Additional Authenticated Data*. Obsérvese como

los valores intermedios g_x son, excepto en el caso de g_0 el cual es el resultado de la multiplicación en campo de Galois entre H y el *Additional Authenticated Data*, el resultado o de la operación XOR entre g_{x-1} y el dato cifrador, o de la multiplicación en campo de Galois entre g_{x-1} y la constante H.

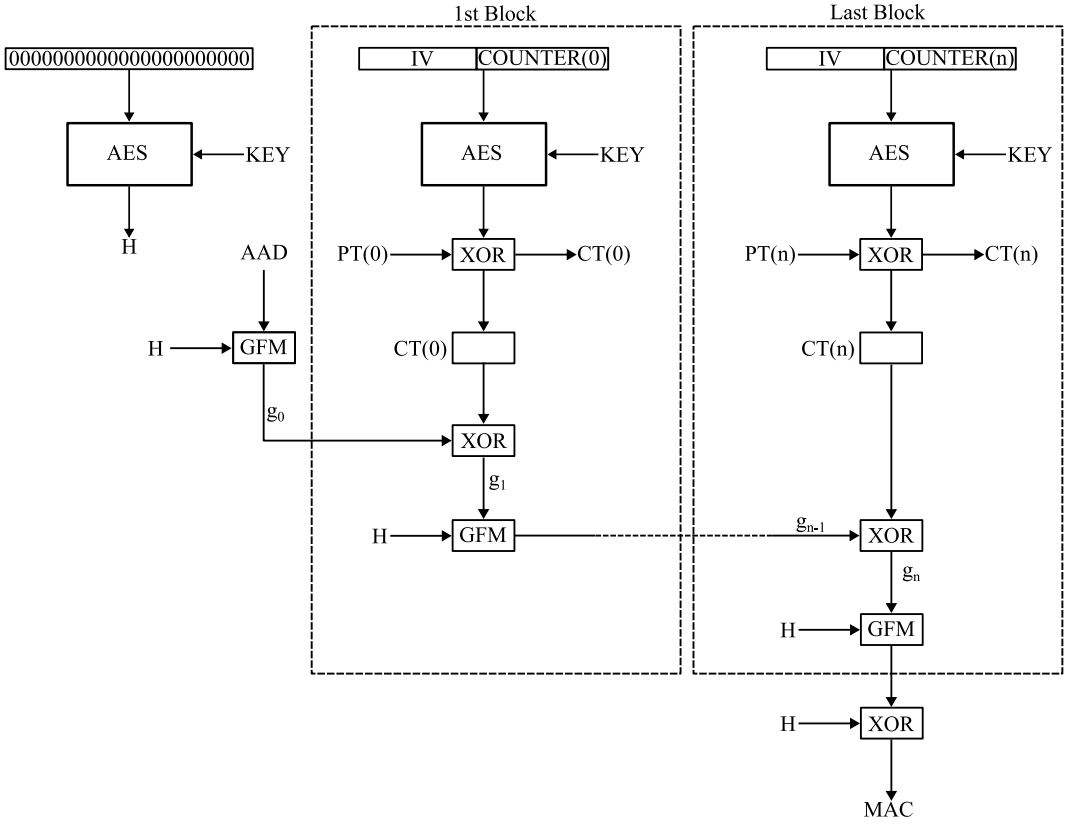


Figura 2.15: Modo de operación *GCM* empleando como bloque básico el cifrador AES.

2.6. Algoritmos de Cifrado con Autenticación *Lightweight*

Como se menciona en la sección anterior, los algoritmos de cifrado convencionales solo proporcionan confidencialidad a la hora de transmitir información, siendo necesario añadir una capa de funcionalidad adicional ya sea mediante hardware o software que permita autenticar la transmisión. Esto da lugar a un mayor consumo de recursos que junto con la implementación del algoritmo de cifrado puede llegar a ser prohibitivo a la hora de implementarlo en dispositivos con un nivel de recursos reducido.

Esta desventaja está cada vez más de manifiesto debido al aumento de aplicaciones de IoT, redes de sensores, y dispositivos *wereables* entre otras, muchas de las cuales pueden manejar información sensible de los usuarios y donde es necesario realizar no sólo un cifrado sino también una autenticación de los mensajes.

Teniendo esto en cuenta en el año 2013 se inicio la competición *CAESAR* organizada por un diverso grupo formado por investigadores de criptología de diversos países y comunidades, y que tenía como objetivo impulsar el desarrollo de algoritmos de cifrado con autenticación. En esta competición los algoritmos propuestos se clasificaron en 3 grupos: aquellos orientados a entornos con un numero de recursos restringidos (*Lightweight Applications*), orientados a aplicaciones de alto rendimiento y *Defense in Depth*. De estos tres grupos el más interesante en el contexto de este trabajo es el primero, cuyos finalistas presentados en el año 2019 son: el algoritmo de cifrado con autenticación Ascon, que es uno de los que se estudian en este documento y el ACORN.

En el mismo año que se anunciaron los finalistas de la competición *CAESAR* el *NIST*, lanzó un proceso con el objetivo de elegir un algoritmo de cifrado con autenticación *Lightweight* como estándar, ya que los estándares aprobados por el *NIST* hasta ese momento no proporcionaban el rendimiento deseado cuando se implementaban en dispositivos con recursos restringidos.

El alcance del proyecto abarca desde cifradores en bloque, esquemas de cifrado con autenticación, funciones *hash*, codigos de autenticación de mensajes, permutaciones criptográficas, y cifradores de flujo.

A día de hoy el proceso se encuentra en sus fase final, y de las 57 propuestas iniciales se han seleccionado 10 finalistas tras 2 rondas de selección. Entre estos finalistas se encuentran tanto el Ascon como el TinyJAMBU que son los dos algoritmos estudiados en este trabajo.

CAPÍTULO 3

Ascon

3.1. Introducción

El cifrador Ascon, diseñado por Christoph Dobraunig, Maria Eichlseder, Florian Mendel y Martin Schläffer, fue uno de los finalistas de la competición CAESAR en el año 2019 y lo es también del proceso de estandarización lanzado por el *NIST* ese mismo año.

Tiene un esquema de funcionamiento muy similar al del cifrador AES, al estar compuesto su elemento principal por una permutación que emplea diferentes capas. Otra semejanza con el AES es que el Ascon utiliza en cada ronda una constante que se genera de forma secuencial, al igual que el AES utiliza una clave derivada de la clave secreta en cada ronda en la que se ejecuta la permutación.

3.2. Especificación

La operación del cifrador se encuentra descrita en [7]. Aún así en este apartado se resumen los aspectos más básicos del principio de operación.

Existen diferentes variantes del Ascon las cuales dependerán de la anchura de la clave a utilizar y dato a cifrar/descifrar así como el numero de rondas a ejecutar por la permutación empleada como elemento básico del cifrador.

La variante propuesta, Ascon-128a, consta, como se observa en la Tabla 3.1, de una clave y un tamaño de bloque de datos de 128 bits y un numero de rondas a y b de 12 y 8 respectivamente.

	Tamaño en bits				Numero de rondas	
Nombre	Key	Nonce	Tag	Dato	p^a	p^b
Ascon-128	128	128	128	64	12	6
Ascon-128a	**128**	**128**	**128**	**128**	**12**	**8**

Tabla 3.1: Parámetros del cifrador Ascon en función de la variante seleccionada.

3.2.1. Registro de estado y notación

Todas las variantes del Ascon operan sobre un registro de estado (**S**) de 320 bits el cual es actualizado realizando una permutación **p** un numero de veces **a** o **b**, dependiendo del dato que se este procesando. Este registro de estados esta dividido en una parte externa ($\mathbf{S_r}$) de **r** bits y una parte interna ($\mathbf{S_c}$) de $c = 320 - r$ bits, donde **r** es el parámetro que indica la anchura del bloque de datos el cual en este caso es **128** bits.

El registro de estado a su vez se divide en 5 bloques de 64 bits con la intención de que la descripción de la permutación en la que se basa el funcionamiento del cifrador sea mas comprensible.

En la Tabla 3.2 se observa la notación empleada en la descripción del principio de operación del cifrador. Si se desea obtener una información más detallada se recomienda acudir a la especificación del cifrador [7].

K	Clave a emplear por el cifrador
N	Nonce a emplear por el cifrador
P,C,A,T	Texto plano, Texto Cifrado, Dato Asociado y Tag
S	Registro de estado del cifrador
S_r, S_c	Partes en las que se divide el registro de estados
p,p^a,p^b	Permutación p la cual se ejecuta un numero a o b de veces
$x \epsilon \{0,1\}^k$	Cadena de bits de longitud k, variable en caso de $k = *$
0^k	Cadena de bits de longitud k, todos los bits son 0
$\|x\|$	Longitud, en bits, de la cadena de bits x
$\lfloor x \rfloor_k$	Cadena de bits truncada a los k bits menos significativos
$\lceil x \rceil^k$	Cadena de bits truncada a los k bits más significativos
$x\|\|y$	Concatenación de las cadenas de bits x e y
$x \oplus y$	Operación XOR entre las cadenas x e y
$x \ggg i$	Rotación circular de i bits hacia la derecha de la cadena x

Tabla 3.2: Notación empleada en la descripción del cifrador Ascon.

Destacar que el Dato Asociado y el Nonce son intercambiables por el *Authenticated Associated Data* y el *IV* presentados en la sección 2.4 y tienen el mismo cometido, con la diferencia de que en este caso el Dato Asociado tiene una longitud variable. En el caso de los algoritmos de cifrado con autenticación, el Dato Asociado se emplea para añadir contexto a las transmisiones de una misma sesión mientras el Nonce es un dato utilizado una única vez por cada sesión de cifrado/descifrado.

3.2.2. Procedimiento a seguir para el procesado de los diferentes bloques de datos

En la Tabla 3.3 se muestran los bloques a procesar por el cifrador Ascon128a. Aunque en el caso del Ascon el bloque Nonce en lugar de ser procesado es almacenado para ser utilizado en diferentes ocasiones durante la operación del cifrador.

Orden	Bloque de datos
1	Carga de la clave
2	Carga del Nonce
3	Dato Asociado
4	Texto Plano/Cifrado
5	Tag

Tabla 3.3: Bloques de datos a procesar por el Ascon en orden de procesado.

Inicialización

Antes de comenzar con el procesado de los bloques ya mencionados se debe de inicializar el registro de estados del cifrador con un valor inicial. Para ello este se asigna con una combinación entre la clave y el Nonce a utilizar durante la operación y se ejecuta una primera permutación.

Algoritmo 1 Pseudocódigo del proceso de inicialización.

$S \leftarrow IV||K||N$
$S \leftarrow p^a(S) \oplus (0^{320-k}||K)$

El primer paso, como se muestra en el Algoritmo 1, a la hora de realizar un proceso de cifrado/descifrado será el siguiente:

1. Almacenar en el registro de estado la concatenación del Nonce, la clave y un vector de inicialización (IV) fijo y propio del algoritmo.

2. Almacenar en el registro de estado la operación XOR entre el resultado de aplicar la permutación p a veces al registro de estado y la clave a la cual se le ha aplicado un "padding" con ceros.

El valor del vector de inicialización para el Ascon-128a es:

$$\mathbf{IV = 0x80800c0800000000}$$

Procesado del dato asociado

El procesado del Dato Asociado consiste en la división de este bloque en bloques de más pequeños de 128 bits y en su introducción mediante la aplicación de una operación XOR en el registro de estado. Tras esta operación se vuelve a ejecutar la permutación empleada como elemento básico del cifrador.

Algoritmo 2 Pseudocódigo del procesado del dato asociado.

if $|A| > 0$ **then**
  $A_1...A_s \leftarrow$ se divide $(A||1||0^*)$ en bloques de 128 bits
  for $i = 1, ..., s$ **do**
    $S \leftarrow p^b((S_r \oplus A_i)||S_c)$
  end for
end if
$S \leftarrow S \oplus (0^{319}||1)$

Para el procesado del dato asociado se siguen, como se indica en el Algoritmo 2, los siguientes pasos:

1. Se divide el dato asociado en bloques de 128 bits, aplicando un pad en el último bloque en caso de que este esté incompleto. Este pad consiste en añadir un '1' en el bit inmediatamente superior al último bit del bloque y completar con '0' hasta obtener un bloque de 128 bits.

2. Cada bloque se introduce en el registro de estados mediante una operación XOR entre el dato asociado A_i y la parte interna de este S_r.

3. Se ejecuta la permutación p b veces.

4. Se repiten los dos pasos anteriores por cada bloque A_i.

5. Finalmente se almacena en el registro de estado la operación XOR de este con una cadena que contiene solo ceros y un uno en el bit más significativo.

Pseudocódigo del procesado del Texto Plano

El procesado del Texto Plano u operación de cifrado se realiza siguiendo el mismo procedimiento que para el Dato Asociado. La principal diferencia entre ambos procesamiento es que durante el del Texto Plano se genera una salida que se corresponde con la del Texto Cifrado.

Algoritmo 3 Pseudocódigo del procesado del Texto Plano.

$P_1...P_t \leftarrow$ se divide $(P||1||0^*)$ en bloques de 128 bits
for $i = 1, ..., t-1$ **do**
    $S_r \leftarrow S_r \oplus P_i$
    $C_i \leftarrow S_r$
    $S \leftarrow p^b(S)$
end for
$S_r \leftarrow S_r \oplus P_t$
$C_t \leftarrow \lfloor S_r \rfloor_{|P_t|}$

Para el procesado del Texto Plano siguiendo el procedimiento del Algoritmo 3:

1. Se divide el Texto Plano en bloques de 128 bits, aplicando un pad en el último bloque en caso de que este ultimo este incompleto.

2. Cada bloque se introduce en el registro de estados mediante una operación XOR entre el Texto Plano P_i y la parte interna del registro de estado S_r. Destacar que el resultado de esta operación se corresponde con un bloque de Texto Cifrado.

3. Se ejecuta la permutación p b veces.

4. Se repiten los dos pasos anteriores con todos los bloques excepto con el último.

5. Finalmente se introduce el último bloque de Texto Plano P_t en el registro de estados mediante la operación XOR entre la parte interna de este S_r y el último bloque del Texto Plano, correspondiéndose el resultado de esta última operación con el ultimo bloque de Texto Cifrado.

Procesado del Texto Cifrado

El procesado del Texto Cifrado u operación de Descifrado es idéntico al del Texto Plano, con la salvedad de que en este caso la entrada del cifrador se corresponde con la salida del mismo durante la operación de Cifrado. Además de esto también existe una diferencia en el orden de aplicación de las operaciones, que se puede observar al comparar los Algoritmos 3 y 4.

Algoritmo 4 Pseudocódigo del procesado del Texto Cifrado.

$C_1...C_t \leftarrow$ se divide $(C||1||0^*)$ en bloques de 128 bits
for $i = 1, ..., t-1$ **do**
    $P_i \leftarrow S_r \oplus C_i$
    $S \leftarrow C_i||S_c$
    $S \leftarrow p^b(S)$
end for
$P_t \leftarrow \lfloor S_r \rfloor_{|C_t|} \oplus C_t$
$S_r \leftarrow S_r \oplus (P_t||1||0^*)$

El procesado del Texto Cifrado, indicado en el Algoritmo 4, es muy similar al del Texto Plano y sigue el siguiente procedimiento:

1. Se divide el Texto Cifrado en bloques de 128 bits, aplicando un pad en el último bloque en caso de que este ultimo este incompleto.
2. Se calcula el Texto Plano resultado del descifrado mediante la operación XOR entre el bloque de Texto Cifrado C_i y la parte interna del registro de estados S_r.
3. Se introduce el Texto Cifrado en la parte interna del registro de estados S_r recuperándose el valor del registro de estados durante el proceso de cifrado.
4. Se ejecuta la permutación p b veces.
5. Se repiten los 3 pasos anteriores con todos los bloques excepto con el último.
6. Se calcula el ultimo bloque de Texto Plano P_t resultado de descifrar el último bloque de Texto Cifrado C_t mediante la operación XOR entre C_t y la parte interna del registro de estado del cifrador.
7. Finalmente se introduce en la parte interna del registro de estado el resultado de la operación XOR de esta parte del estado y el Texto Plano P_t con su pad correspondiente.

Proceso de finalización

El proceso de finalización consiste en la generación y transmisión de un MAC/Tag, que depende de todo el proceso anterior, en el caso de que se este realizando una operación de Cifrado. O en la comparación entre la entrada del cifrador, la cual debe de ser asignada durante este proceso con el MAC/Tag generado durante el cifrado, con el MAC/Tag generado durante el proceso de Descifrado el cual debe de coincidir con el de Cifrado para garantizar la autenticidad de la comunicación.

Algoritmo 5 Pseudocódigo del proceso de finalización.

$S \leftarrow p^a(S \oplus (0^r||K||0^{320-r-k}))$
$T \leftarrow \lceil S \rceil^{128} \oplus \lceil K \rceil^{128}$

El proceso de finalización, como se indica en el Algoritmo 5, sigue los siguiente pasos:

1. Se ejecuta la permutación p a veces al resultado de la operación XOR entre el registro de estado y una cadena de bits que consisten en la concatenación de 128 ceros, la clave y el numero de ceros restante hasta completar la anchura del registro de estado.
2. Se genera el Tag mediante la operación XOR entre los 128 bits mas significativos del estado S y la clave K.

3.2.3. Permutación

La permutación consiste en la aplicación, sobre el registro de estado, de 3 operaciones combinacionales que reciben el nombre p_L, p_S y p_C. Estas consisten en una capa de difusión lineal, una capa de sustitución y una adición de una constante que depende de la ronda que se este ejecutando de la permutación, un esquema de estas operaciones se puede observar en las Figuras 3.1, 3.2 y 3.3.

Para la descripción de la permutación se divide el registro de estado en 5 palabras de 64 bits cada una, obteniéndose como resultado: $S = x_0||x_1||x_2||x_3||x_4$.

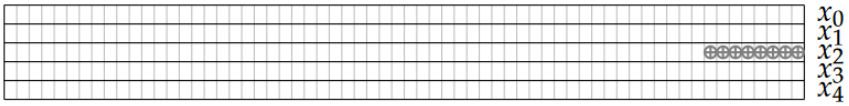


Figura 3.1: Esquema de la operación realizada por la capa de adición de constantes.

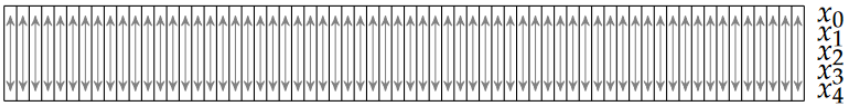


Figura 3.2: Esquema de la operación realizada por la capa de sustitución.

Figura 3.3: Esquema de la operación realizada por la capa de difusión lineal.

Capa de adición de constantes

Esta capa consiste en aplicar una operación XOR al los bits menos significativos de la palabra x_2 con una constante que depende de la ronda de la permutación que se este ejecutando (Figura 3.1). En la Tabla 3.4 se puede observar las constantes que se aplican dependiendo de la ronda de las permutaciones p^a y p^b.

Ronda de p^a	Ronda de p^b	Constante c_r
0		0x00000000000000f0
1		0x00000000000000e1
2		0x00000000000000d2
3		0x00000000000000c3
4	0	0x00000000000000b4
5	1	0x00000000000000a5
6	2	0x0000000000000096
7	3	0x0000000000000087
8	4	0x0000000000000078
9	5	0x0000000000000069
10	6	0x000000000000005a
11	7	0x000000000000004b

Tabla 3.4: Constantes aplicadas sobre la palabra x_2 en la capa p_c.

Capa de sustitución

Consiste en la asignación del registro de estado mediante la aplicación en paralelo de 64 S-box de 5 bits de entrada donde esta será la concatenación del bit **i** de las palabras $x_0, ..., x_4$ en las que se divide el registro de entrada($x = x_{4_i}||x_{3_i}||x_{2_i}||x_{1_i}||x_{0_i}$). En la Tabla 3.5 se muestra

el contenido de la s-box mientras que en la Figura 3.4 se observa la operación lógica que estas s-box implementan.

El objetivo de esta operación es que los bits de cada palabra en la que se divide el registro de estado, sea dependientes de los mismos bits de las palabras restantes.

x	$S(x)$	x	$S(x)$
0x0	0x4	0x10	0x1E
0x1	0xB	0x11	0x13
0x2	0x1F	0x12	0x7
0x3	0x14	0x13	0xE
0x4	0x1A	0x14	0x0
0x5	0x15	0x15	0xD
0x6	0x9	0x16	0x11
0x7	0x2	0x17	0x18
0x8	0x1B	0x18	0x10
0x9	0x5	0x19	0xC
0xA	0x8	0x1A	0x1
0xB	0x12	0x1B	0x19
0xC	0x1D	0x1C	0x16
0xD	0x3	0x1D	0xA
0xE	0x6	0x1E	0xF
0xF	0x1C	0x1F	0x17

Tabla 3.5: Comportamiento de la S-box empleada en la capa de sustitución.

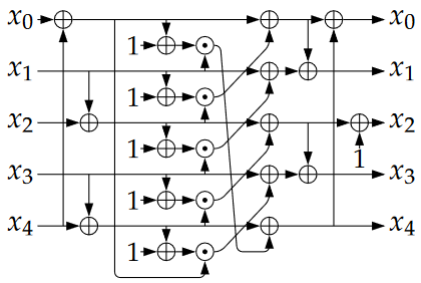


Figura 3.4: Operaciones boolenas que implementa la S-box.

Capa de difusión lineal

Es la encargada de proporcionar difusión entre los bits de cada palabra en la que se ha dividido el registro de estado (Figura 3.3).

Realiza la siguientes operaciones:

$$x_0 \leftarrow x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28)$$
$$x_1 \leftarrow x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39)$$
$$x_2 \leftarrow x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6)$$
$$x_3 \leftarrow x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17)$$
$$x_4 \leftarrow x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41)$$

3.3. Diseño

En la Figura 3.5 se puede observar la interfaz empleada para el control del cifrador, así en las Tablas 3.6 y 3.7 aparecen las señales de esta junto a una pequeña descripción de su uso.

Las señales empleadas se dividen en: buses de datos, como **dataIn** y **dataOut**, señales de control, que son **keyUpdate**, **dValid**, **dEob**, **dEop**, **init**, **partial**, **partialBits** y **decryptIn**, y señales de indicación como **outValid**, **authValid** y **dReady**.

Destacar que aunque el bus de entrada tenga un tamaño de 32 bits, el tamaño de los bloques como se indica en el apartado de especificación es de 128 bits, luego los bloques completos se introducen en 4 ciclos de reloj siendo este numero de ciclos inferior en el caso de que se trate de un bloque parcial.

El diseño ha sido realizado en VHDL cumpliendo con las restricciones de las principales herramientas de síntesis.

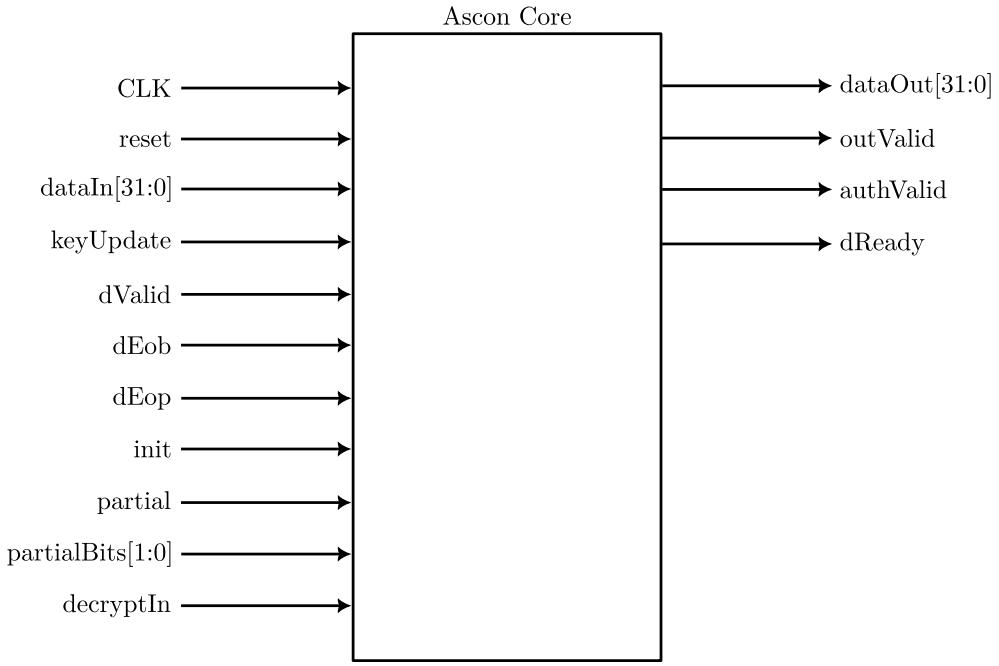


Figura 3.5: Interfaz propuesta para el cifrador Ascon.

Señal	Tamaño	Descripción
CLK	1 bit	Señal de reloj
reset	1 bit	Señal de reset activo en alta
dataIn	32 bits	Bus de entrada de datos
keyUpdate	1 bit	Señal que indica que se va a realizar la carga de una nueva clave
dValid	1 bit	Señal que indica la validez del dato de entrada del bus INPUT
dEob	1 bit	Señal que indica que se esta procesando el último dato de un bloque
dEop	1 bit	Señal que indica que no se va a procesar el siguiente bloque
init	1 bit	Señal que marca el comienzo de una operación de cifrado/descifrado
partial	1 bit	Señal que indica que el ultimo dato de un bloque es parcial
partialBits	2 bit	Señal que indica el numero de bytes que forman una palabra parcial
decryptIn	1 bit	Señal que indica si la operación a realizar es un cifrado o un descifrado

Tabla 3.6: Señales de entrada empleadas por el cifrador Ascon.

Señal	Tamaño	Descripción
dataOut	32 bits	Bus de salida del crifrador
outValid	1 bit	Señal que indica que hay un dato valido en el bus de salida
authValid	1 bit	Señal que indica que la validez de una operación de descifrado
dReady	1 bit	Señal que indica que el cifrador esta listo para recibir un dato de entrada

Tabla 3.7: Señales de salida empleadas por el cifrador Ascon.

3.3.1. Estructura del Diseño

La jerarquía del diseño consta únicamente 2 componentes. El Top en el que se encuentra descrita la maquina de estados de control del cifrador así como la lógica combinacional necesaria para generar las salidas y el módulo *asconPermutation* en el que se describe la permutación que emplea el algoritmo.

asconPermutation

Consiste en la descripción hardware de la permutación empleada por el algoritmo expuesta en el apartado anterior. Los puertos de este módulo aparecen listados en la Tabla 3.8.

La permutación se utiliza como elemento de realimentación del registro de estados. Para ello recibe como entradas el valor actual del registro de estados y la constante a añadir en cada ronda, y tiene una salida con el valor a asignar en el registro de estados en el siguiente flanco activo de reloj.

Señal	Tamaño	Descripción
stateIn	320 bits	Entrada que se asignará con el valor del registro de estado del cifrador después de cada ronda
rCon	4 bits	Entrada que se corresponde con la constante a añadir en cada ronda por la capa de adición de constantes descrita en el apartado anterior
stateOut	320 bits	Salida de la permutación con la que se actualizara el registro de estado

Tabla 3.8: Señales de entrada y salida de la permutación del Ascon.

Ascon

Es el módulo de mayor jerarquía del diseño, en el se encuentra tanto la descripción de la maquina de estados que implementa el algoritmo como la lógica combinacional empleada para generar la salida de datos del cifrador así como la instancia del módulo de la permutación anterior.

Las señales de entrada y de salida del módulo son las de la interfaz del cifrador y están descritas en las Tablas 3.6 y 3.7.

3.3.2. Descripción de la operación del cifrador

El cifrador ha sido diseñado con una señal de reset asíncrona (*reset*) activa en alto de modo que siempre que el valor de este bit de entrada sea '1' el cifrador se inicializará a un estado inicial.

Una vez se haya aplicado el reset, el cifrador se mantendrá en un estado de espera hasta que se produzca un cambio en las señales *keyUpdate* o *init*, donde la primera se empleará para iniciar el proceso de carga de la clave en su registro correspondiente y la segunda indicará el comienzo del proceso de cifrado o descifrado en función del valor de la entrada *decryptIn* como se observa en la Tabla 3.9.

decryptIn	Operación a realizar
'0'	Cifrado
'1'	Descifrado

Tabla 3.9: Operación a realizar por el cifrador según la señal de entrada *decryptIn*.

Proceso de carga de la clave

Este proceso se ejecuta al encontrarse el cifrador en estado de espera y cambiar el valor de la entrada *keyUpdate* a '1'. Destacar que esta señal es de mayor prioridad que la señal *init*, por lo que en el caso de que el valor de ambas fueran '1' de manera simultanea se procedería a cargar la clave en lugar de dar comienzo el proceso de cifrado/descifrado indicado por la señal *decryptIn*.

El proceso de carga de clave (Figura 3.6) consistirá en la introducción utilizando el bus de entrada *input* de 4 palabras de 32 bits que forman los 128 bits de la clave que se utilizará durante la operación del cifrador, estas palabras se deben de asignar al bus de entrada durante los 4 ciclos de reloj que dura el periodo de carga tras la indicación con la señal *keyUpdate*.

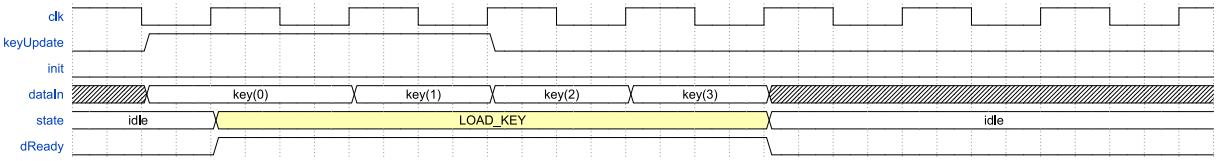


Figura 3.6: Diagrama temporal del proceso de carga de la clave en el cifrador Ascon.

Comienzo del funcionamiento del cifrador

Para comenzar una operación de cifrado, el cifrador parte del estado de espera y se le indicará empleando las señales *init* e *decryptIn* el comienzo de la operación y la operación a realizar respectivamente.

La señal *init* es activa en alta, luego cuando su valor sea '1' el cifrador comenzará a operar.

La señal *decryptIn* indica la operación a ejecutar según lo indicado en la Tabla 3.9.

Una vez haya dado comienzo la operación del cifrador se asignará el bus de entrada *input* con los 32 bits menos significativos del Nonce, y se indicará que este dato es válido empleando la señal *dValid* la cual es activa en alta.

Carga del Nonce

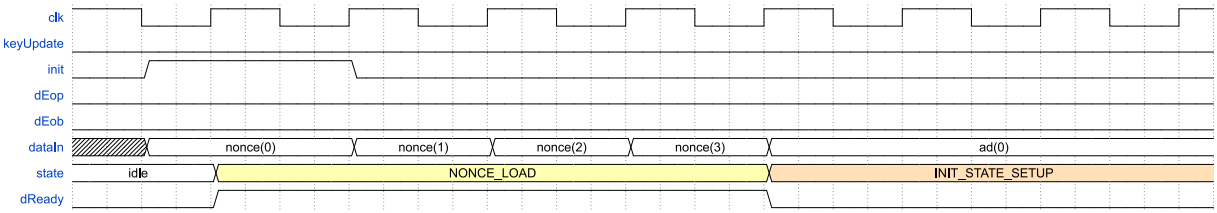
Tras indicar el inicio de la operación de cifrado/descifrado, se pasa a cargar el Nonce en su registro correspondiente. Esto se hace de la misma manera que para la clave, es decir, se debe de actualizar el bus de entrada con las palabras del Nonce en los 4 ciclos de reloj que dura el estado de carga posterior a la indicación con la señal *init*.

Tras la carga del Nonce se asigna el bus de entrada con la primera palabra del siguiente bloque a procesar, dependiendo este bloque de una combinación de las señales *dEob* y *dEop* como se indica en la Tabla 3.10.

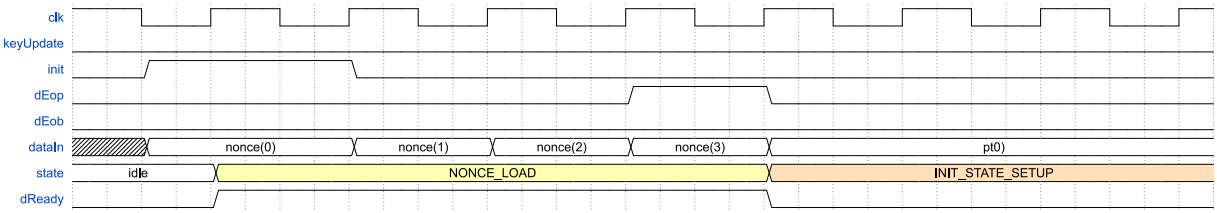
En la Figura 3.7 se muestra el proceso de carga del Nonce en el cifrador, diferenciándose entre los casos en el que el siguiente bloque a procesar es el Dato Asociado, Texto Plano/Cifrado o el Tag.

dEob	*dEop*	Siguiente bloque a procesar
0	0	Dato Asociado
0	1	Texto Plano/Cifrado
1	x	Tag

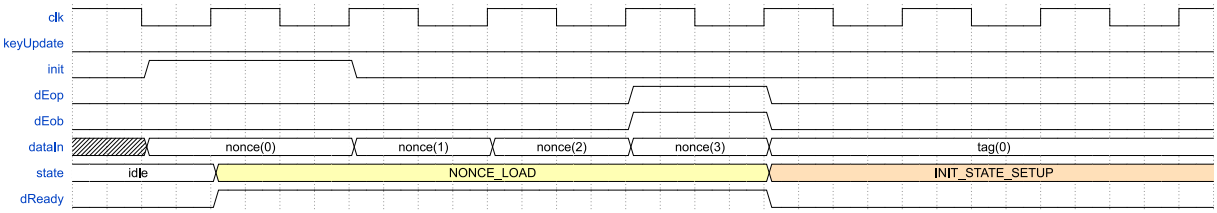
Tabla 3.10: Siguiente bloque a procesar por el Ascon en función del valor de las entradas *dEop* y *dEob* al procesar la última palabra del Nonce.



(a) Proceso de carga del Nonce en el cifrador Ascon en el caso de que el próximo bloque a procesar sea el Dato Asociado.



(b) Proceso de carga del Nonce en el cifrador Ascon en el caso de que el próximo bloque a procesar sea el Texto Plano/Cifrado.



(c) Proceso de carga del Nonce en el cifrador Ascon en el caso de que el próximo bloque a procesar sea el Tag.

Figura 3.7: Diagrama temporal del proceso de carga del Nonce

Procesado de los diferentes bloques

En la Tabla 3.11 se puede observar los distintos bloques de datos que procesa el cifrador en orden de procesado, donde las señales más empleadas a la hora de procesar los diferentes bloques son: *dEob*, *dEop*, *partial* y *partialBits*.

Orden	Bloque de datos
1	Dato Asociado
2	Texto Plano/Cifrado
3	Tag

Tabla 3.11: Bloques de datos a procesar por el cifrador en orden de procesado.

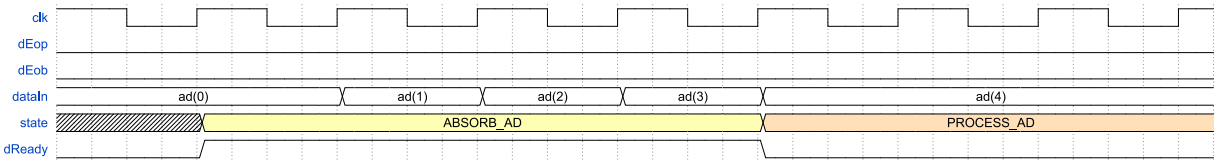
Procesado del Dato Asociado

En el caso de que se vaya a procesar el bloque de Dato Asociado la primera palabra de éste se asignará al bus de datos *input* tras la carga en el registro correspondiente de la última palabra del Nonce. Tras esto se espera a que la señal *dReady* indique que se puede actualizar el valor del bus de entrada con las siguientes palabras que formen el correspondiente bloque de Dato Asociado. Las palabras del Dato Asociado se asignan al bus de entrada en los 4 ciclos de reloj posteriores al primer periodo tras la indicación de la señal *dReady*.

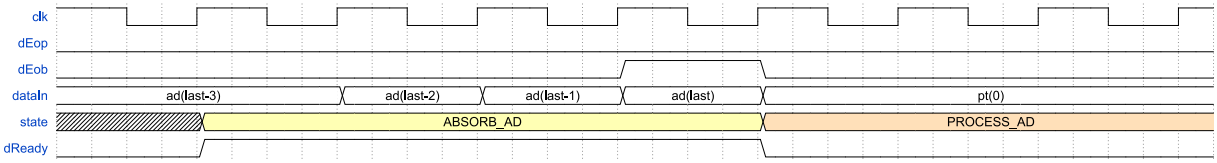
Durante el procesado de la última palabra del bloque de Dato Asociado las señales *dEob* y *dEop* se asignan dependiendo del siguiente bloque que se desee procesar como se indica en la Tabla 3.12. De nuevo se muestra en la Figura 3.8 ejemplos del procesado de un bloque de Dato Asociado en función de si se va a seguir procesando el Dato Asociado, o si por el contrario el siguiente bloque a procesar será el Texto Plano/Cifrado o el Tag.

dEob	*dEop*	Siguiente bloque a procesar
0	0	Dato Asociado
1	0	Texto Plano/Cifrado
1	1	Tag

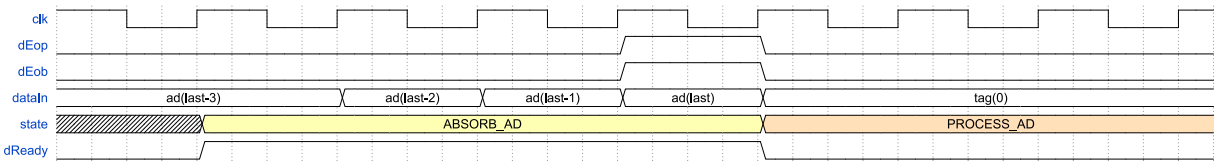
Tabla 3.12: Siguiente bloque a procesar por el Ascon en función del valor de las entradas *dEop* y *dEob* al procesar la última palabra del Dato Asociado.



(a) Proceso de carga del Dato Asociado en el cifrador Ascon en el caso de que el tamaño de este sea superior a 128 bits.



(b) Proceso de carga del último bloque de Dato Asociado en el cifrador Ascon en el caso de que el siguiente bloque a procesar sea el Texto Cifrado.



(c) Proceso de carga del último bloque de Dato Asociado en el cifrador Ascon en el caso de que el siguiente bloque a procesar sea el Tag.

Figura 3.8: Diagrama temporal del procesado del Dato Asociado

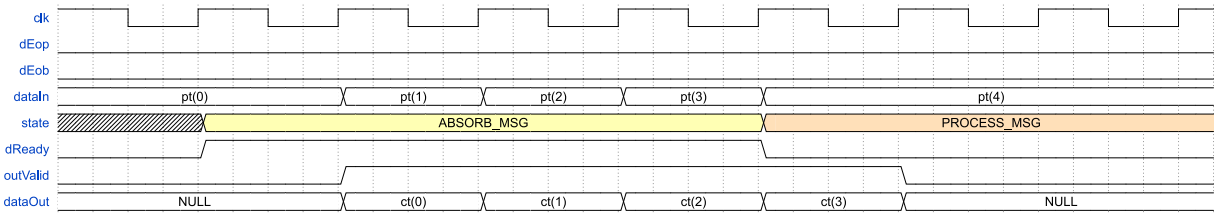
Procesado del Texto Plano/Cifrado

El procesado de los bloques de Texto Plano/Cifrado se realiza de la misma manera que para el Dato Asociado. La única diferencia consiste en la combinación de las señales *dEob* y *dEop* durante el procesado de la última palabra que compone el bloque de Texto Plano/Cifrado, ya que al tratarse este del último bloque opcional solo se emplea la señal *dEob* para indicar el final del proceso de Cifrado/Descifrado como se muestra en la Tabla 3.13. En la Figura 3.9 se observa el procesado del Texto Plano en función de si se va a procesar otro bloque de Texto Plano o si este ha sido el último bloque y se va a pasar a procesar el Tag.

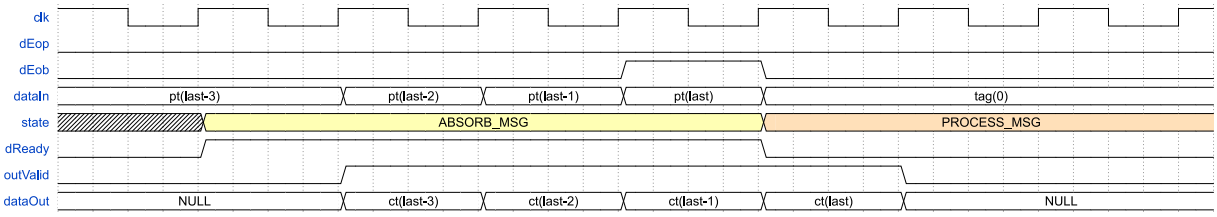
dEob	*dEop*	Siguiente bloque a procesar
0	0	Texto Plano/Cifrado
1	0	Tag

Tabla 3.13: Siguiente bloque a procesar por el Ascon en función del valor de las entradas *dEop* y *dEob* al procesar la última palabra del Texto Plano/Cifrado.

Durante el procesado de este bloque, la señal de salida *outValid* indica que el bus de salida se ha asignado con una palabra resultado de haber Cifrado/Descifrado la palabra de Texto Plano/Cifrado del bus de entrada.



(a) Procesado del Texto Plano en el cifrador Ascon en el caso de que el tamaño de este sea superior a 128 bits.



(b) Procesado del último bloque del Texto Plano.

Figura 3.9: Diagrama temporal del procesado del Texto Plano

Proceso de Finalización

Tras el procesado del Texto Plano/Cifrado se debe de realizar la generación o autenticación del Tag dependiendo de si se esta realizando una operación de cifrado o descifrado respectivamente.

En el caso del cifrado no es necesario asignar el bus de entrada con ningún valor, sin embargo durante el descifrado se debe de asignar las 4 palabras que forman el Tag al bus de entrada de la misma manera que se hace con el Dato Asociado y Texto Plano/Cifrado. Además, la salida *authValid* indica, cuando se esta descifrando, la validez de la operación de descifrado. Un ejemplo del proceso de finalización cuando se esta descifrando se muestra en la figura 3.10.

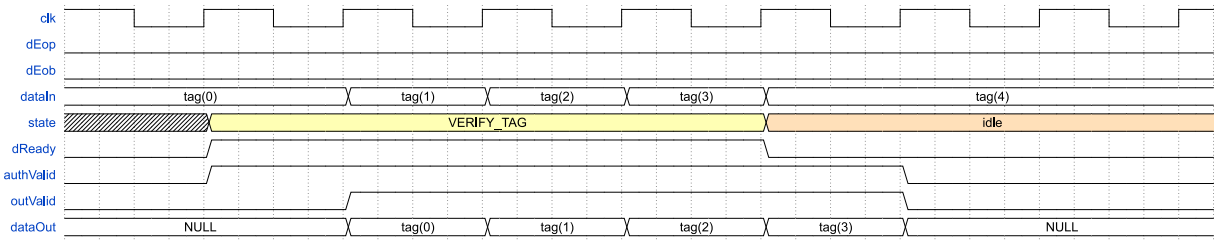


Figura 3.10: Diagrama temporal del procesado del Tag durante una operación de descifrado.

Procesado de Bloques Parciales

En el procesado del Dato Asociado y del Texto Plano/Cifrado se puede dar el caso de que el último bloque sea parcial, teniendo la última palabra de este un tamaño inferior o igual a 32 bits, la manera de proceder se muestra en las Figuras 3.11 y 3.12 y es la siguiente:

Durante el ciclo de reloj que permanece asignada la ultima palabra al bus de entrada y las señales *dEob* y *dEop* como se indica en las secciones anteriores, se debe de asignar a '1' la señal *partial* así como la señal *partialBits* según se indica en la Tabla 3.14.

Tamaño de Palabra	Valor *partialBits*
1 byte	0b01
2 bytes	0b10
3 bytes	0b11
4 bytes	0b00

Tabla 3.14: Valor de señal partialBits dependiendo del tamaño del bloque.

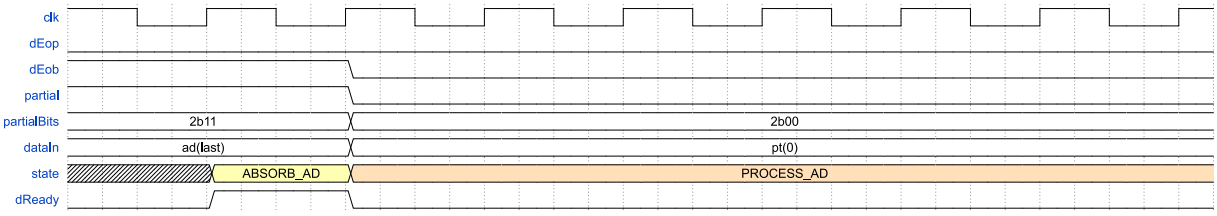


Figura 3.11: Proceso de carga de un bloque de Dato Asociado formado por una única palabra de 3 bytes.

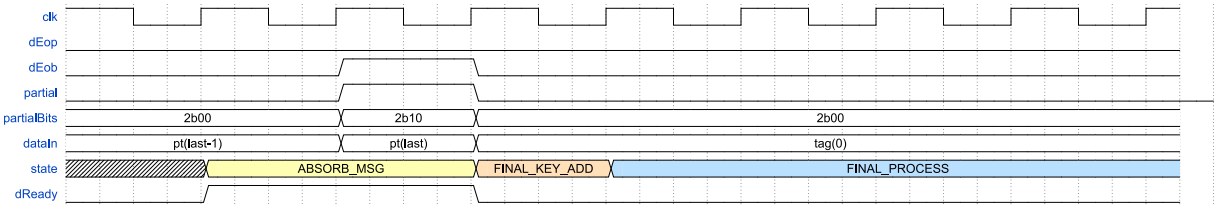


Figura 3.12: Proceso de carga de un bloque de Texto Plano formado por dos palabras de las cuales la última esta compuesta por 2 bytes.

3.4. Verificación Funcional

Para comprobar el correcto funcionamiento del diseño se han utilizado los vectores de test propuestos por los diseñadores. Así se ha realizado una prueba de cifrado y descifrado para cada

tipo de operación que puede realizar este cifrador, estas operaciones se muestran en la Tabla 3.15.

En la Tabla 3.15, las operaciones 1 y 2 tienen las siglas AEAD y AE respectivamente. Estas vienen de la expresión en inglés *Authenticated Encryption with Associated Data* y *Authenticated Encryption* respectivamente, donde la diferencia entre ambas operaciones es el uso o no del bloque de Dato Asociado. Las operaciones restantes consisten en la generación de un MAC que depende únicamente del Nonce empleado en el proceso (Operación 3), o del Nonce y del Dato Asociado (Operación 4).

	Nonce	Dato Asociado	Texto Plano/Cifrado	MAC/Tag
Operación 1 (AEAD)	✓	✓	✓	✓
Operación 2 (AE)	✓	-	✓	✓
Operación 3	✓	-	-	✓
Operación 4	✓	✓	-	✓

Tabla 3.15: Tipo de operación a realizar por el Ascon en función de los bloques de datos procesados durante la operación

Los vectores de test utilizados para la verificación de los diferentes tipos de operación que puede realizar el Ascon son:

Operación 1 (AEAD)

- *Clave*: 0x8728308800DAC1DDA6D2FFFCAF81961A
- *Nonce*: 0xACFD295E495808ACDD29405D6E0A2144
- *Dato Asociado*: 0xAE19165A00B51BFD1282F0A6024D619D
- *Texto Plano/Cifrado*:
  - *Texto Plano*: 0x58CEF788A45AE9B53128983D68C711DA
  - *Texto Cifrado*: 0x447D8572C17975B4849139F475B998B1
- *MAC/Tag*: 0xA75AB2308FB9AF163903E1D4ED6D85E1

Operación 2 (AE)

- *Clave*: 0x035D92CD14B938FA79E65D4BCCD65E78
- *Nonce*: 0x10E4B34C9ACCC8D609F610D8B1215189
- *Texto Plano/Cifrado*:
  - *Texto Plano*: 0x7DC350E4753AD8365F64B8EFD6E295CC
  - *Texto Cifrado*: 0x02DAC49E757F88A4A72124612D8D97BB
- *MAC/Tag*: 0x26F42D36C9A366E994649025C55F4B9B

Operación 4

- *Clave*: 0xB5A02D0D421C6C970AD7E42326C2CDEE
- *Nonce*: 0x4B1D6AE5B8C5CAD8D90B2D446F87822F
- *Dato Asociado*: 0xF8C8A6B22AF4B213BF55F2D86F27E85F
- *MAC/Tag*: 0x26CDCACC2BE243E966CBC10B09241056

CAPÍTULO 4

TinyJAMBU

4.1. Introducción

El algoritmo de cifrado con autenticación TinyJAMBU, diseñado por Hongjun Wu y Tao Huang, es uno de los candidatos seleccionados en la última fase del proceso de estandarización organizado por el *NIST* en 2019. Es una variación de tamaño reducido del algoritmo JAMBU que fue uno de los seleccionados en la tercera ronda de la competición *CAESAR*.

Emplea como elemento principal para el procesado de los diferentes bloques un único NLFSR, lo que permite que el consumo de recursos sea mucho más reducido que sus competidores y hace ineludible una comparación con cifradores de flujo que utilizan estos elementos como el cifrador *Trivium*.

4.2. Especificación

En este aparatado se expone una breve descripción del principio de operación del cifrador TinyJAMBU, la operación del cifrador se encuentra descrita con más detalle en el documento [8].

Se diferencian tres variantes de este cifrador dependiendo de si la longitud de clave que emplee 128, 192 o 256 bits, aunque en este documento se describe el que emplea una clave de 128 bits mostrándose los parámetros de éste en la Tabla 4.1. Destacar que todas las variantes procesan los datos en bloques de 32 bits.

	Tamaño en bits				
Nombre	Clave	Nonce	Tag	Dato	Registro de estado
TinyJAMBU128	**128**	**96**	**64**	**32**	**128**
TinyJAMBU192	192	96	64	32	128
TinyJAMBU256	256	96	64	32	128

Tabla 4.1: Parámetros del cifrador TinyJAMBU128.

4.2.1. Notación y Registro de Estado

En la Tabla 4.2 se expone la notación utilizada para describir el principio de funcionamiento del cifrador.

Todas las variantes del TinyJAMBU emplean un registro de estados de 128 bits que coincide con el contenido del NLFSR (Figura 4.1) empleado para realizar la operación de permutación. La operación del NLFSR viene descrita en el Algoritmo 6. Destacar que la permutación emplea como entrada el bit más significativo del registro de clave que realiza un desplazamiento serie mientras que esta se ejecuta.

Algoritmo 6 Pseudocódigo de la operación del NLFSR.

StateUpdate(S,K,i):

$feedback = s_0 \oplus s_{47} \oplus (\sim (s_{70} \& s_{85})) \oplus s_{91} \oplus k_i$
for $j = 0, ..., 126$ **do**
  $s_j = s_{j+1}$
  $s_{127} = feedback$
end for

$\oplus$	Operación XOR
&	Operación AND
$\sim$	Operación NOT
$\|\|$	Concatenación
mod	Operación módulo
$\lfloor x \rfloor$	Valor truncado al entero inferior
AD, ad_i	Dato Asociado, bit i del Dato Asociado
adlen	Longitud del dato asociado en bits
K, k_i	Clave, bit i de la clave
klen	Longitud de la clave en bits
M, m_i	Texto plano, bit i del Texto Plano
mlen	Longitud del Texto Plano en bits
C, c_i	Texto plano, bit i del Texto Plano
NONCE, $nonce_i$	Nonce, bit i del Nonce
T, T_i	Tag, bit i del Tag
S,s_i	Los 128 bits del registro de estado, bit i del registro de estado
P_n	Permutación de n rondas del registro de estado
partialLength	Longitud en bits de un dato parcial
FrameBits	Valor constante dependiente del bloque : FrameBits= 0b001 para el Nonce FrameBits= 0b011 para el Dato Asociado FrameBits= 0b101 para el Texto Plano/cifradoNonce FrameBits= 0b111 para la finalizaciónNonce

Tabla 4.2: Notación empleada en la descripción del cifrador TinyJAMBU.

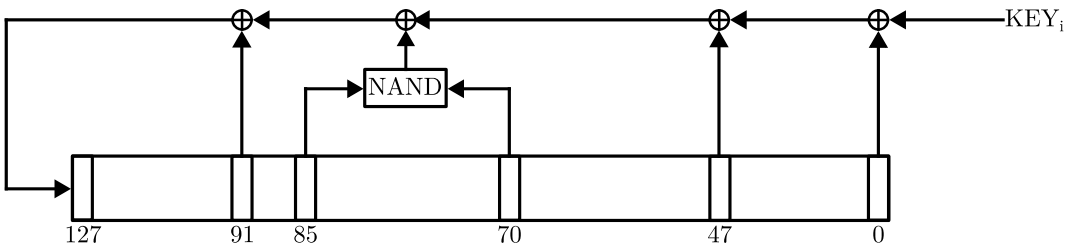


Figura 4.1: NLFSR empleado en el cifrador TinyJAMBU.

4.2.2. Procedimiento a seguir para el procesado de los diferentes bloques de datos

Como en el caso del Ascon, en la Tabla 4.3 se muestran los bloques de datos a procesar por el TinyJAMBU.

En el caso del TinyJAMBU y excepto en el proceso de inicialización, todos los bloques se procesan de manera muy similar. Este procesado consiste en la introducción de una constante en el registro de estados la cual dependerá del bloque que se esté procesando. Tras esto se ejecuta la permutación que utiliza el TinyJAMBU como elemento básico un numero de veces que depende del bloque que se está procesando. Finalmente se introduce el dato del bloque a procesar en el registro de estado mediante una operación XOR.

Destacar que en el caso del Dato Asociado y el Texto Plano/Cifrado, y en el caso en que se trate de bloques parciales, además del proceso descrito en el párrafo anterior se debe de introducir en el registro de estados y junto con el dato a procesar el tamaño de este en bytes.

Orden	Bloque de datos
1	Carga de la clave
2	Nonce
3	Dato Asociado
4	Texto Plano/Cifrado
5	Tag

Tabla 4.3: Bloques de datos a procesar por el TinyJAMBU en orden de procesado.

Inicialización del Registro de estado

Este proceso consiste en la inicialización del NLFSR que emplea como elemento básico el TinyJAMBU mediante la ejecución de la permutación durante los primeros 1024 ciclos de reloj de la operación.

Algoritmo 7 Pseudocódigo del proceso de inicialización del registro de estado.

$S \leftarrow 0$
$S \leftarrow P_{1024}(S)$

El proceso de inicialización, como se indica en el Algoritmo 7, consiste en:

1. Inicializar el registro de estados a cero.

2. Realizar la permutación implementada por el NLFSR durante 1024 ciclos de reloj.

Procesado del Nonce

Algoritmo 8 Pseudocódigo del procesado del Nonce.

for $i = 0, ..., 2$ **do**
  $s_{[38,...,36]} = s_{[38,...,36]} \oplus FrameBits_{[2,...,0]}$
  $S \leftarrow P_{640}(S)$
  $s_{[127,...,96]} = s_{[127,...,96]} \oplus nonce_{[32i+31,...,32i]}$
end for

El procesado de las 3 palabras de 32 bits que componen el Nonce (Algoritmo 8) sigue los siguientes pasos:

1. Actualizar el registro de estados mediante la asignación de los bits 38 a 36 de la operación XOR entre los bits 38 a 36 del estado y el valor de FrameBits correspondiente al procesado del Nonce.

2. Ejecutar la permutación durante 640 ciclos

3. Introducir en el registro de estados el resultado de la operación XOR entre los bits 127 a 96 del registro de estados y la palabra correspondiente del Nonce.

4. Repetir el proceso anterior para las 3 palabras del Nonce

Procesado del Dato Asociado

Algoritmo 9 Procesado del Dato Asociado.

if $(adlen \ mod \ 32) = 0$ **then**
  for $i = 0, ..., \lfloor adlen/32 \rfloor$ **do**
    $s_{[38,...,36]} = S_{[38,...,36]} \oplus FrameBits_{[2,...,0]}$
    $S \leftarrow P_{640}(S)$
    $s_{[127,...,96]} = s_{[127,...,96]} \oplus ad_{[32i+31,...,32i]}$
  end for
else
  $s_{[38,...,36]} = S_{[38,...,36]} \oplus FrameBits_{[2,...,0]}$
  $S \leftarrow P_{640}(S)$
  $s_{[96+partialLength-1,...,96]} = s_{[96+partialLength-1,...,96]} \oplus ad_{[32i+partialLength-1,...,32i]}$
  $s_{[33,32]} = s_{[33,32]} \oplus (partialLength/8)$
end if

El procedimiento a seguir para el procesado del bloque de dato asociado (Algoritmo 9) es:

1. Actualizar el registro de estados mediante la asignación de los bits 38 a 36 de la operación XOR entre los bits 38 a 36 del estado y el valor de FrameBits correspondiente al procesado del Dato Asociado.

2. Ejecutar la permutación durante 640 ciclos.

3. Introducir en el registro de estados el resultado de la operación XOR entre los bits 127 a 96 del registro de estados y la palabra correspondiente del Dato Asociado.

En caso de que el bloque de AD este compuesto únicamente por palabras de 32 bits se debe de repetir el procedimiento anterior para todos los bloques. En el caso de que la ultima palabra sea una palabra parcial (longitud inferior a 32 bits), se debe de seguir el siguiente procedimiento:

1. Actualizar el registro de estados mediante la asignación de los bits 38 a 36 de la operación XOR entre los bits 38 a 36 del estado y el valor de FrameBits correspondiente al procesado del Dato Asociado.

2. Ejecutar la permutación durante 640 ciclos.

3. Introducir en el registro de estados el resultado de la operación XOR entre los bits 127 a el bit correspondiente según el tamaño de la palabra parcial y la última palabra parcial del Dato Asociado.

4. Finalmente se actualiza el registro de estados asignando a los bits 33 a 32 el resultado de la operación XOR entre estos bits y dos bits que indican el tamaño en bytes de la última palabra del Dato Asociado.

Procesado del Texto Plano

Algoritmo 10 Pseudocódigo del procesado Texto Plano.

if $(mlen\ mod\ 32) = 0$ **then**
  for $i = 0, ..., \lfloor mlen/32 \rfloor$ **do**
    $s_{[38,...,36]} = S_{[38,...,36]} \oplus FrameBits_{[2,...,0]}$
    $S \leftarrow P_{1024}(S)$
    $s_{[127,...,96]} = s_{[127,...,96]} \oplus m_{[32i+31,...,32i]}$
    $c_{[32i+31,...,32i]} = s_{[95,...,64]} \oplus m_{[32i+31,...,32i]}$
  end for
else
  $s_{[38,...,36]} = S_{[38,...,36]} \oplus FrameBits_{[2,...,0]}$
  $s \leftarrow P_{1024}(S)$
  $s_{[96+partialLength-1,...,96]} = S_{[96+partialLength-1,...,96]} \oplus m_{[32i+partialLength-1,...,32i]}$
  $c_{[32i+partialLength-1,...,32i]} = s_{[95,...,64]} \oplus m_{[32i+partialLength-1,...,32i]}$
  $s_{[33,32]} = s_{[33,32]} \oplus (partialLength/8)$
end if

El procedimiento a seguir para el procesado del bloque de Texto Plano (Algoritmo 10) es:

1. Actualizar el registro de estados mediante la asignación de los bits 38 a 36 de la operación XOR entre los bits 38 a 36 del estado y el valor de FrameBits correspondiente al procesado del Texto Plano/cifrado.
2. Ejecutar la permutación durante 1024 ciclos.
3. Introducir en el registro de estados el resultado de la operación XOR entre los bits 127 a 96 del registro de estados y el Texto Plano.
4. Generar la palabra correspondiente de Texto Cifrado mediante la operación XOR entre los bits 95 a 64 del estado y la palabra correspondiente de Texto Plano.

En caso de que el bloque de Texto Plano este compuesto únicamente por palabras de 32 bits se debe de repetir el procedimiento anterior para todos los bloques. En el caso de que la ultima palabra sea una palabra parcial (longitud inferior a 32 bits), se debe de seguir el siguiente procedimiento:

1. Actualizar el registro de estados mediante la asignación de los bits 38 a 36 de la operación XOR entre los bits 38 a 36 del estado y el valor de FrameBits correspondiente al procesado del Texto Plano/cifrado.
2. Ejecutar la permutación durante 1024 ciclos.
3. Introducir en el registro de estados el resultado de la operación XOR entre los bits, del registro de estados, 127 a el bit correspondiente según el tamaño de la palabra parcial y la última palabra parcial del Texto Plano.
4. Generar la palabra correspondiente de Texto Cifrado mediante la operación XOR entre los bits, del estado, 95 a el bit correspondiente según el tamaño de la palabra parcial y la palabra de texto plano.
5. Finalmente se actualiza el registro de estados asignando a los bits 33 a 32 el resultado de la operación XOR entre estos bits y dos bits que indican el tamaño en bytes de la última palabra del Texto Plano.

Procesado del Texto Cifrado

Algoritmo 11 Pseudocódigo del procesado Texto Cifrado.

if $(mlen \ mod \ 32) = 0$ **then**
  for $i = 0, ..., \lfloor mlen/32 \rfloor$ **do**
    $s_{[38,...,36]} = s_{[38,...,36]} \oplus FrameBits_{[2,...,0]}$
    $S \leftarrow P_{1024}(S)$
    $m_{[32i+31,...,32i]} = s_{[95,...,64]} \oplus c_{[32i+31,...,32i]}$
    $s_{[127,...,96]} = s_{[127,...,96]} \oplus m_{[32i+31,...,32i]}$
  end for
else
  $s_{[38,...,36]} = S_{[38,...,36]} \oplus FrameBits_{[2,...,0]}$
  $S \leftarrow P_{1024}(S)$
  $m_{[32i+partialLength-1,...,32i]} = s_{[96+partialLength,...,96]} \oplus c_{32i+partialLength-1,...,32i}$
  $s_{[96+partialLength-1,...,96]} = s_{[96+partialLength-1,...,96]} \oplus m_{[32i+partialLength-1,...,32i]}$
  $s_{[33,32]} = s_{[33,32]} \oplus (partialLength/8)$
end if

El procedimiento a seguir para el procesado del bloque de texto cifrado, como se indica en el Algoritmo 11, es:

1. Actualizar el registro de estados mediante la asignación de los bits 38 a 36 de la operación XOR entre los bits 38 a 36 del estado y el valor de FrameBits correspondiente al procesado del Texto Plano/Cifrado.
2. Ejecutar la permutación durante 1024 ciclos.
3. Generar la palabra correspondiente del Texto Plano mediante la operación XOR entre los bits 95 a 64 del estado y la palabra correspondiente de Texto Cifrado.
4. Introducir en el registro de estados el resultado de la operación XOR entre los bits 127 a 96 del registro de estados y el Texto Cifrado.

En caso de que el bloque de texto cifrado este compuesto únicamente por palabras de 32 bits se debe de repetir el procedimiento anterior para todos los bloques. En el caso de que la ultima palabra sea una palabra parcial (longitud inferior a 32 bits), se debe de seguir el siguiente procedimiento:

1. Actualizar el registro de estados mediante la asignación de los bits 38 a 36 de la operación XOR entre los bits 38 a 36 del estado y el valor de FrameBits correspondiente al procesado del Texto Plano/Cifrado.
2. Ejecutar la permutación durante 1024 ciclos.
3. Generar la palabra correspondiente del Texto Plano mediante la operación XOR entre los bits del estado 95 a el bit correspondiente según el tamaño de la palabra parcial y la última palabra parcial de Texto Cifrado.
4. Introducir en el registro de estados el resultado de la operación XOR entre los bits 127 a el bit correspondiente según el tamaño de la palabra parcial y la última palabra parcial del Texto Cifrado.
5. Finalmente se actualiza el registro de estados asignando a los bits 33 a 32 el resultado de la operación XOR entre estos bits y dos bits que indican el tamaño en bytes de la última palabra del Texto Cifrado.

Proceso de Finalización

Algoritmo 12 Pseudocódigo del proceso de finalización.

$s_{[38,...,36]} = s_{[38,...,36]} \oplus FrameBits_{[2,...,0]}$
$S \leftarrow P_{1024}(S)$
$t_{[31,...,0]} = s_{[95,..,64]}$
$s_{[38,...,36]} = s_{[38,...,36]} \oplus FrameBits_{[2,...,0]}$
$S \leftarrow P_{640}(S)$
$t_{[63,...,32]} = s_{[95,..,64]}$

Como se muestra en el Algoritmo 12,el procedimiento a seguir en el proceso de finalización sigue los siguientes pasos:

1. Actualizar el registro de estados mediante la asignación de los bits 38 a 36 de una operación XOR entre los bits 38 a 36 del estado y el valor de FrameBits correspondiente al proceso de finalización.

2. Ejecutar la permutación durante 1024 ciclos.

3. Generar la primera palabra del tag que se corresponde con los bits 95 a 64 del registro de estado.

4. Actualizar el registro de estados mediante la asignación de los bits 38 a 36 de una operación XOR entre los bits 38 a 36 del estado y el valor de FrameBits correspondiente al proceso de finalización.

5. Ejecutar la permutación durante 640 ciclos

6. Generar la segunda palabra del tag que se corresponde con los bits 95 a 64 del registro de estado.

4.3. Diseño

En la Figura 4.2 se puede observar la interfaz empleada para el control de la operación del cifrador, así en las Tablas 4.4 y 4.5 aparecen las señales empleadas para el control de éste acompañadas de una pequeña descripción de su uso.

Las señales empleadas se dividen en: buses de datos, como **dataIn** y **dataOut**, señales de control, que son **keyUpdate**, **dValid**, **dEob**, **dEop**, **init**, **partial**, **partialBits** y **decryptIn**, y señales de indicación como **outValid**, **authValid** y **dReady**.

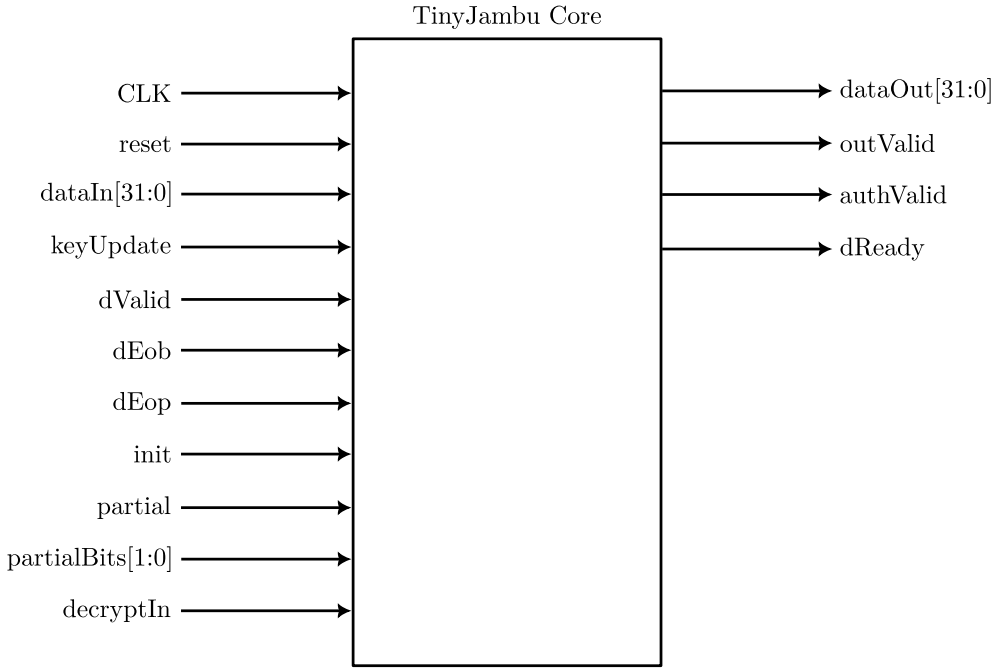


Figura 4.2: Interfaz propuesta para el cifrador TinyJAMBU.

Señal	Tamaño	Descripción
CLK	1 bit	Señal de reloj
reset	1 bit	Señal de reset activo en alta
input	32 bits	Bus de entrada de datos
keyUpdate	1 bit	Señal que indica que se va a realizar la carga de una nueva clave
dValid	1 bit	Señal que indica la validez del dato de entrada del bus INPUT
dEob	1 bit	Señal que indica que se esta procesando el último dato de un bloque
dEop	1 bit	Señal que indica que no se va a procesar el siguiente bloque
init	1 bit	Señal que marca el comienzo de una operación de cifrado/descifrado
partial	1 bit	Señal que indica que el ultimo dato de un bloque es parcial
partialBits	2 bit	Señal que indica el numero de bytes que forman una palabra parcial
decryptIn	1 bit	Señal que indica si la operación a realizar es un cifrado o un descifrado

Tabla 4.4: Señales de entrada empleadas por el cifrador TinyJAMBU.

Señal	Tamaño	Descripción
output	32 bits	Bus de salida del crifrador
outValid	1 bit	Señal que indica que hay un dato valido en el bus de salida
authValid	1 bit	Señal que indica la autenticidad de una operación de descifrado
dReady	1 bit	Señal que indica que el cifrador esta listo para recibir un dato de entrada

Tabla 4.5: Señales de salida empleadas por el cifrador TinyJAMBU.

4.3.1. Estructura del diseño

La jerarquía del diseño presenta los siguientes componentes:

- TinyJAMBUTop
- keyReg
- stateReg
- TinyJAMBUControl

Destacar que en este diseño se emplea un generic, *CONCURRENT*, el cual permite paralelizar la operación del NLFSR de modo que se puedan llegar a ejecutar hasta 32 desplazamientos del NLFSR en el mismo ciclo de reloj, lo cual reduce significativamente la latencia del cifrador.

A continuación se describe con mas detalle cada uno de los componentes.

keyReg

Se trata del registro en el que se almacena la clave que se utilizará durante la operación del cifrador, la finalidad de este registro es la proporcionar el bit o bits de entrada para el NLFSR que utiliza el cifrador durante la ejecución de la permutación. Los puertos de este módulo aparecen descritos en la tabla 4.6.

La carga de la clave en el registro se realiza de forma serie en bloques de 32 bits, esta operación se realiza según el valor de la señal *update* la cual actúa como una señal de enable para el almacenamiento de la entrada.

Además el registro dispone de una señal *shift*, para realizar el desplazamiento mediante el cual se obtiene el bit o bits más significativos que serán usados como entrada del NLFSR en el que se basa la permutación del cifrador.

Señal	Tamaño	Descripción
clk	1 bit	Señal de reloj
reset	1 bit	Señal de reset asíncrono activo en alto
update	1 bit	Señal activa en alto que señala cuando se debe de almacenar la entrada en el registro
shift	1 bit	Señal activa en alto que señala cuando se debe de realizar un desplazamiento serie del registro
keyout	CONCURRENT bits	Salida del registro de clave que se utilizará como entrada del NLFSR que se emplea como permutación

Tabla 4.6: Señales de entrada y salida del registro de clave.

stateReg

Consiste en el NLFSR de 128 bits que se utiliza como base para la permutación, este dispone de una señal de enable y carga en paralelo, *nlfsrEn* y *nlfsrLoad* respectivamente, de cuya combinación depende si se realizará un desplazamiento serie del NLFSR o una carga en paralelo con algunas de las entradas que se indican el sección de la especificación descrita anteriormente.

Tanto una descripción más detallada de las señales de entrada y salida como de la operación del NLFSR se muestra en las Tablas 4.7 y 4.8.

Señal	Tamaño	Descripción
clk	1 bit	Señal de reloj
nlfsrReset	1 bit	Señal de reset asíncrono activo en alto
nlfsrEn	1 bit	Señal activa en alto de enable del NLFSR
nlfsrLoad	1 bit	Señal activa en alto de carga en paralelo del NLFSR
key	CONCURRENT bits	Bits del registro de clave que se utilizan como entrada del NLFSR
stateRegIn	128 bits	Entrada paralelo del
stateRegOut	128 bits	Salida que contiene el valor de la permutación y que se utilizara para generar los bloques de salida del cifrador

Tabla 4.7: Señales de entrada y salida del NLFSR del cifrador.

nlfsrEn	nlfsrLoad	Operación del NLFSR
0	0	Se mantiene el estado actual
0	1	Se mantiene el estado actual
1	0	Operación de desplazamiento serie
1	0	Operación de carga en paralelo

Tabla 4.8: Operación del NLFSR en función de las señales nlfsrEn y nlfsrLoad.

TinyJAMBUControl

En este módulo se describe la maquina de estados empleada para el control del cifrador, mediante la cual se generan las señales de control del los módulos keyReg y stateReg, como la constante frameBits descrita en la especificación y otras señales de control empleadas para la generación del dato de salida del cifrador y que se encuentran descritas con más detalle en la tabla 4.9.

Señal	Tamaño	Descripción
clk	1 bit	Señal de reloj
reset	1 bit	Señal de reset asíncrono activo en alto
dValid	1 bit	Señal de entrada activa en alto que indica que en el bus de entrada del cifrador hay un dato valido
dEob	1 bit	Señal de entrada activa en alto que indica cuando se va a terminar de procesar un bloque
dEop	1 bit	Señal de entrada activa en alto que indica que no se va a procesar el siguiente bloque del algoritmo de cifrado con autenticación
init	1 bit	Señal de entrada activa en alto de inicio de la operación
decrypt	1 bit	Señal de entrada activa en alto que indica que operación se esta ejecutando.
keyUpdate	1 bit	Señal de entrada activa en alto que indica que se va a cargar la clave
dataInput	32 bits	Bus de entrada que se utilizará para realizar la comparación entre el dato de entrada durante la verifiación del Tag y el Tag generado durante el proceso
tagData	32 bits	Entrada del Tag generado durante la operación y que se emplea para ser comparado con el dato de entrada durante la verificación del Tag
nlfsrEn	1 bit	Salida que es utilizada como la entrada de enable del componente stateReg
nlfsrLoad	1 bit	Salida que es utilizada como la entrada de load del componente stateReg
nlfsrReset	1 bit	Salida que es utilizada como la entrada de enable del componente stateReg
keyLoad	1 bit	Salida que es utilizada como la entrada de load del componente keyReg
keyShift	1 bit	Salida que es utilizada como la entrada shift del componente keyReg
dReady	1 bit	Salida que indica que se puede asignar el bus de entrada con el siguiente dato a procesar
decryptCtrl	1 bit	Salida empleada por el bloque combinacional con el que se genera los datos que se introducen en el registro de estados del cifrador
outValid	1 bit	Salida que indica que en el bus de salida hay disponible un dato
bitSel	1 bit	Salida que contiene el valor de la permutación y que se utilizara para generar los bloques de salida del cifrador
frameBits	3 bits	Constante frameBits que se utilizan para la actualización del registro de estado
authValid	1 bits	Salida que indica la autenticidad de una comunicación tras el proceso de descifrado
tagGen	1 bit	Salida que se emplea por el bloque combincacional que genera el dato de salida para distinguir entre la generación del texto cifrado/plano o y el Tag
done	1 bit	Salida que indica que la operación del cifrador ha finalizado

Tabla 4.9: Señales de entrada y salida del control del cifrador.

TinyJAMBUTop

Es el módulo de mayor jerarquía del diseño, en el se interconectan los componentes expuestos anteriormente y esta descrita la lógica combinacional empleada tanto para la generación de la salida como para la entrada en paralelo del NLFSR que será necesaria a la hora de realizar una carga en paralelo de este.

Las entradas y salidas de este módulo son las del cifrador y están expuestas en las Tablas 4.4 y 4.5.

4.3.2. Descripción de la operación del cifrador

El cifrador ha sido diseñado con una señal de reset (*reset*) activa en alta de modo que siempre que el valor de este bit de entrada sea '1' el cifrador se inicializará a un estado inicial por defecto.

Una vez se haya aplicado el reset, el cifrador se mantendrá en un estado de espera hasta que se produzca un cambio en las señales *keyUpdate* o *init*, donde la primera se empleará para iniciar el proceso de carga de la clave en su registro correspondiente y la segunda indicará el comienzo del proceso de cifrado o descifrado en función del valor de la entrada *decryptIn*.

Para el procesado de los datos de los diferentes bloques se utilizan la entrada *dValid* y la salida *dReady*. Permaneciendo el cifrador en un estado de espera hasta que no se recibe la indicación de que el dato en el bus de entrada es valido, *dValid* a '1', tras esto se procesa el dato correspondiente según el bloque que se este procesando y se indica durante un ciclo de reloj, con *dReady* a '1', que se puede asignar el bus de entrada con el siguiente dato a procesar.

El funcionamiento de estas señales se observa en las siguientes secciones junto con la manera de procesar cada bloque de datos.

Proceso de carga de la clave

Este proceso se ejecuta al encontrarse el cifrador en estado de espera y cambiar el valor de la entrada *keyUpdate* a '1'. Destacar que esta señal es de mayor prioridad que la señal *init*, por lo que en el caso de que el valor de ambas fuera '1' de manera simultanea se procedería a cargar la clave en lugar de dar comienzo el proceso indicado por la señal *decryptIn*.

El proceso de carga de clave consistirá, como se observa en la Figura 4.3, en la introducción utilizando el bus de entrada *input*, de 4 palabras de 32 bits que forman los 128 bits de la clave que se utilizará durante la operación del cifrador. En la Figura 4.3 se observa como sería la carga de estas palabras, destacar como se permanece en el estado *wait_key* cuando el valor de la señal *dValid* no es igual a '1'.

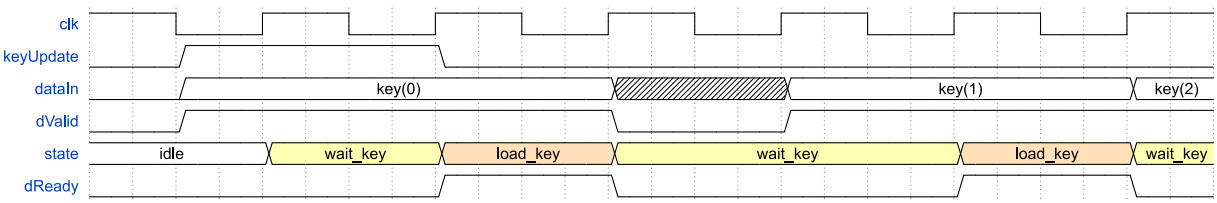


Figura 4.3: Ejemplo del proceso de carga de la clave.

Comienzo del funcionamiento del cifrador

Para comenzar una operación de cifrado, como se muestra en la Figura 4.4, el cifrador partirá del estado de espera y se le indicará empleando las señales *init* e *decryptIn* el comienzo de la operación y la operación a realizar respectivamente.

La señal *init* es activa en alta, luego cuando su valor sea '1' el cifrador comenzará a operar.

La señal *decryptIn* indica la operación a ejecutar según lo indicado en Tabla 4.10:

decryptIn	Operación a realizar
'0'	Cifrado
'1'	Descifrado

Tabla 4.10: Señal de entrada *decryptIn*.

Una vez haya dado comienzo la operación del cifrador se asignará el bus de entrada *input* con los 32 bits menos significativos del Nonce, y se indicará que este dato es valido empleando la señal *dValid*.

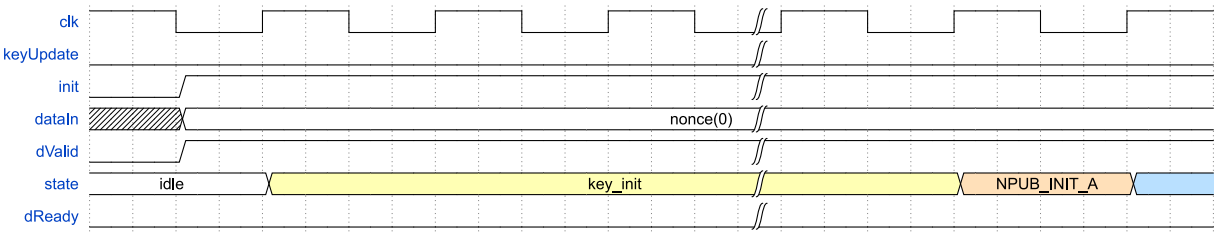


Figura 4.4: Diagrama temporal del comienzo de operación del TinyJAMBU.

Procesado de los diferentes bloques de datos

Las señales principales, además de *dValid* y *dReady*, empleadas para el control del procesado de los bloques son: *dEob*, *dEop*, *partial* y *partialBits*. Es importante diferenciar entre el procesado del Nonce que tiene un tamaño fijo de 96 bits como se indicaba en la descripción de la especificación, y el procesado del dato asociado y el Texto Plano/cifrado que tienen un tamaño variable.

Procesado del Nonce

Una vez iniciada la operación, con el bus de entrada asignado con los 32 bits menos significativos del Nonce y *dValid* indicando la validez de estos datos, al terminar de procesar esta primera palabra se actualizará el bus de entrada cuando la señal *dReady* lo indique. Destacar que al ser el Nonce un dato de tamaño fijo tan solo se debe de realizar esta operación 3 veces para procesar las 3 palabras de este bloque. Un ejemplo del procesado de la segunda palabra del Nonce y que se ejecuta de manera idética para el resto de palabras de este bloque se puede observar en la Figura 4.5.

Dependiendo de los valores de los bits *dEop* y *dEob*, el siguiente bloque a procesar será uno u otro según se indica en la Tabla 4.11. En las Figuras 4.6a , 4.6b y 4.6c se muestra conforme a la información de esta tabla el procesado de la última palabra del Nonce en los casos en que el siguiente bloque sea el Dato Asociado, Texto Plano/Cifrado o Tag respectivamente.

dEob	*dEop*	Siguiente bloque a procesar
0	0	Dato Asociado
0	1	Texto Plano/Cifrado
1	X	Tag

Tabla 4.11: Siguiente bloque a procesar por el TinyJAMBU en función del valor de las entradas *dEop* y *dEob* al procesar la última palabra del Nonce.

Tras el procesado de la última palabra del Nonce se debe de asignar el bus de entrada *input* con la palabra menos significativa del siguiente bloque a procesar.

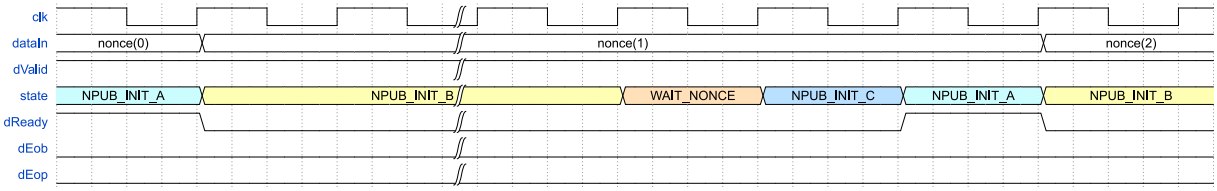
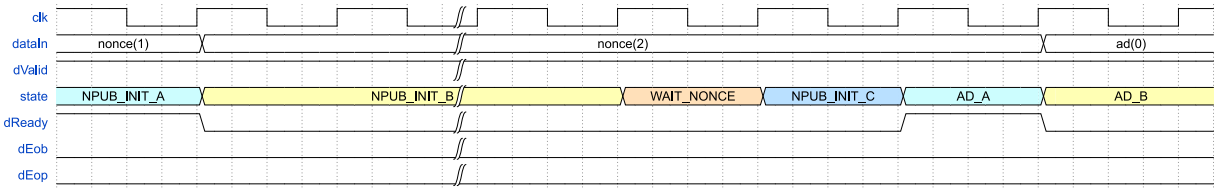
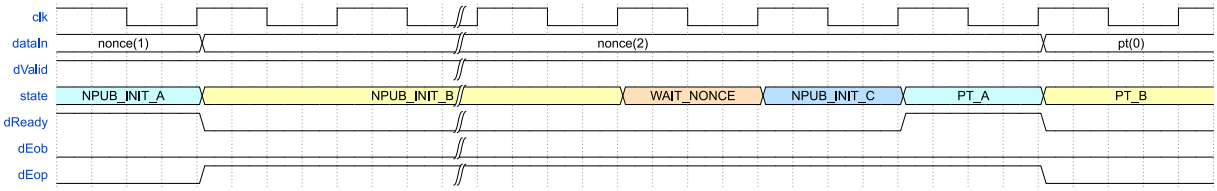


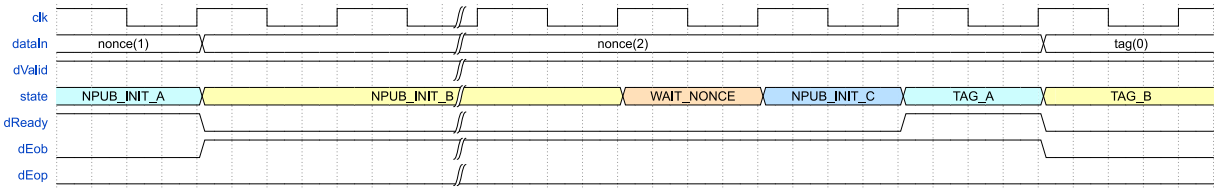
Figura 4.5: Diagrama temporal del procesado de la segunda palabra del Nonce.



(a) Procesado de la última palabra del Nonce si el siguiente bloque a procesar va a ser el Dato Asociado.



(b) Procesado de la última palabra del Nonce si el siguiente bloque a procesar va a ser el Texto Plano/Cifrado.



(c) Procesado de la última palabra del Nonce si el siguiente bloque es el Tag.

Figura 4.6: Diagrama temporal del procesado de la última palabra del Nonce.

Procesado del Dato Asociado

El procesado del Dato Asociado se realiza siguiendo el mismo procedimiento que para el Nonce con la diferencia de que como este bloque es de tamaño variable es necesario indicar cuando se está procesando la ultima palabra mediante el uso de la señal *dEob*.

Cuando se este procesando la última palabra de este bloque se deberán de asignar las señales

dEob y *dEop* como se indica en la Tabla 4.12 según el siguiente bloque que se desee procesar.

dEob	*dEop*	Siguiente bloque a procesar
0	0	Dato Asociado
1	0	Texto Plano/Cifrado
1	1	Tag

Tabla 4.12: Siguiente bloque a procesar por el TinyJAMBU en función del valor de las entradas *dEop* y *dEob* al procesar la última palabra del Dato Asociado.

Así el procedimiento es el siguiente:

Cuando la señal *dReady* indique que se ha terminado de procesar la palabra correspondiente del Dato Asociado se asigna el bus de entrada *input* con la siguiente palabra, permaneciendo *dEob* y *dEop* a '0' siempre que no se trate de la última palabra de este bloque. En el caso de que se trate de la última palabra del Dato Asociado además de asignar el bus de entrada con la palabra correspondiente se debe indicar asignando un '1' a la señal *dEob* que se trata de la última palabra de esta bloque.

Si tras el procesado del Dato Asociado no se deseará procesar el texto Plano/Cifrado se debe de indicar de la misma forma que se hace con la señal *dEob*, mediante la asignación de *dEop* a '1' durante el procesado de la ultima palabra del Dato Asociado. En las Figuras 4.7, 4.8a y 4.8b se muestra el procesado del Dato Asociado dependiendo de si se va a seguir procesando el Dato Asociado o se ha terminado con este bloque y el siguiente será el Texto Plano/Cifrado o el Tag.

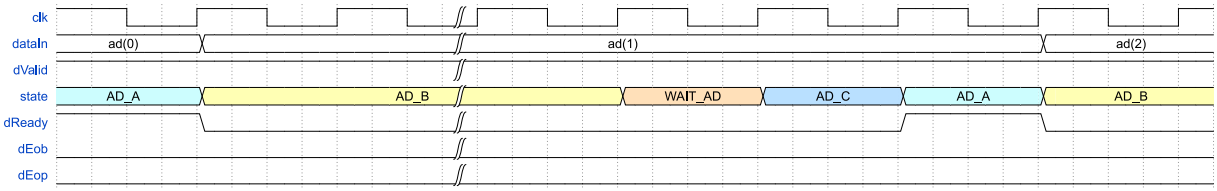
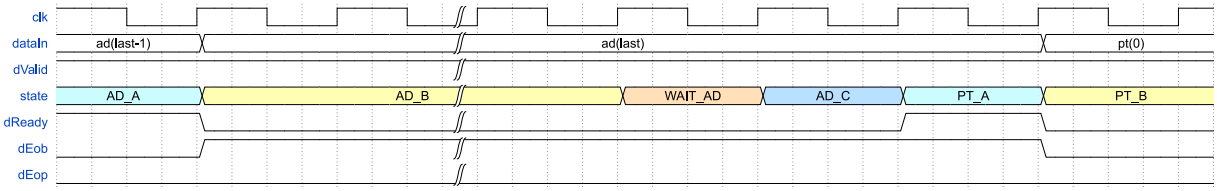
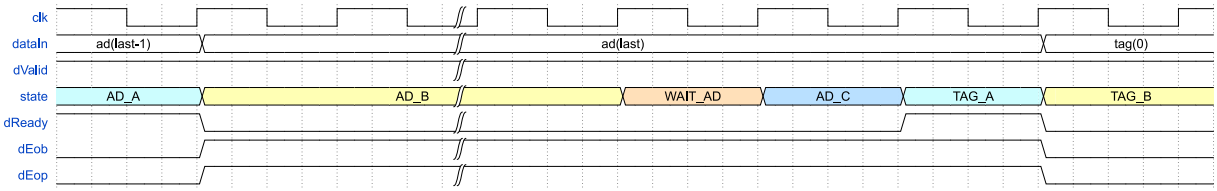


Figura 4.7: Diagrama temporal del procesado de la segunda palabra del Dato Asociado.



(a) Procesado de la última palabra del Dato Asociado si el siguiente bloque a procesar va a ser el Texto Plano/Cifrado.



(b) Procesado de la última palabra del Dato Asociado si el siguiente bloque es el Tag.

Figura 4.8: Diagrama temporal del procesado de la última palabra del Dato Asociado.

Procesado del Texto Plano/Cifrado

El procesado del Texto Plano/Cifrado se realiza de forma idéntica al bloque anterior. Cuando la señal *dReady* indique que se ha terminado de procesar la palabra correspondiente del Texto Plano/Cifrado se asigna el bus de entrada *input* con la siguiente palabra, como se observa en la Figura 4.9. En el caso de que se trate de la última palabra del Texto Plano/Cifrado (Figura 4.10) se debe además de asignar el bus de entrada con la palabra correspondiente indicar asignando un '1' a la señal *dEob* que se trata de la última palabra de este bloque.

Para obtener el resultado del cifrado/descifrado de cada palabra del bloque se debe de esperar a que la señal *outValid* indique con un '1' que en el bus de salida *output* esta asignado con una salida valida.

Cuando se este procesando la última palabra de este bloque se deberán de asignar las señales *dEob* y *dEop* como se indica en la Tabla 4.13 dependiendo del siguiente bloque que se desee procesar.

dEob	*dEop*	Siguiente bloque a procesar
1	0	Tag

Tabla 4.13: Siguiente bloque a procesar por el TinyJAMBU en función del valor de las entradas *dEop* y *dEob* al procesar la última palabra del Texto Plano/Cifrado.

Una vez que se haya procesado la última palabra del bloque y cuando la señal *dReady* lo indique se debe de asignar al bus de entrada *input* la primera palabra del Tag en el caso de que se esté realizando una operación de descifrado, en caso contrario no es necesario asignar el bus de entrada.

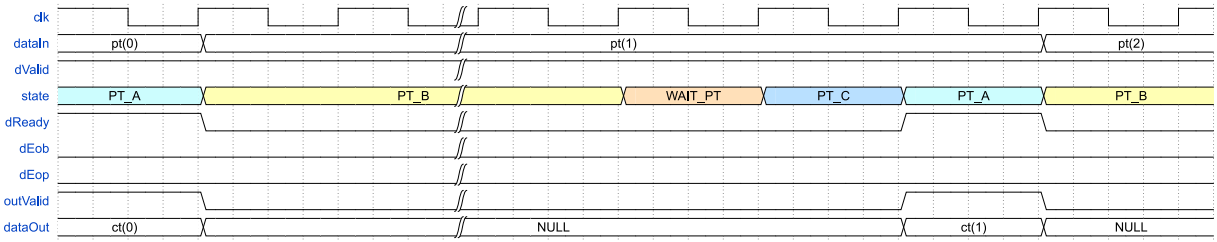


Figura 4.9: Diagrama temporal del procesado de la segunda palabra del Texto Plano.

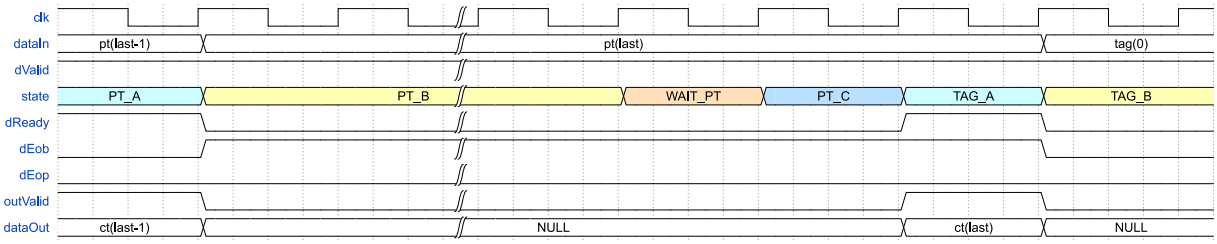


Figura 4.10: Diagrama temporal del procesado de la última palabra del Texto Plano.

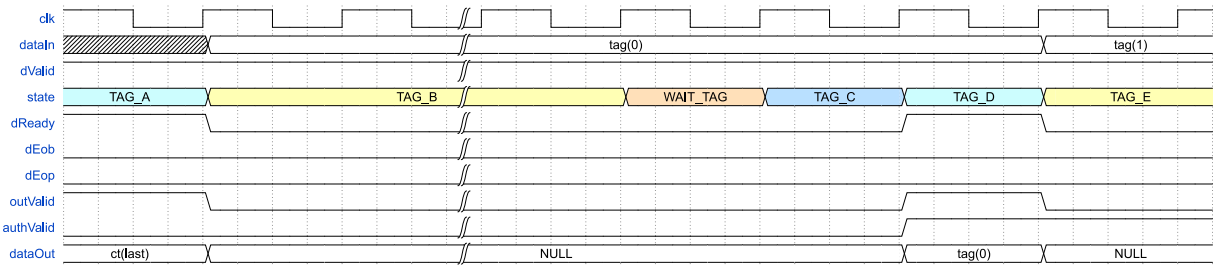
Procesado del Tag

Tras el procesado del Texto Plano/Cifrado, se asigna el bus de entrada con la primera palabra del Tag en caso de que se este realizando una operación de descifrado y se espera como en casos anteriores a que la señal *dReady* indique que se ha procesado esta, tras esto se asigna el bus de entrada *input* con la segunda palabra del bloque.

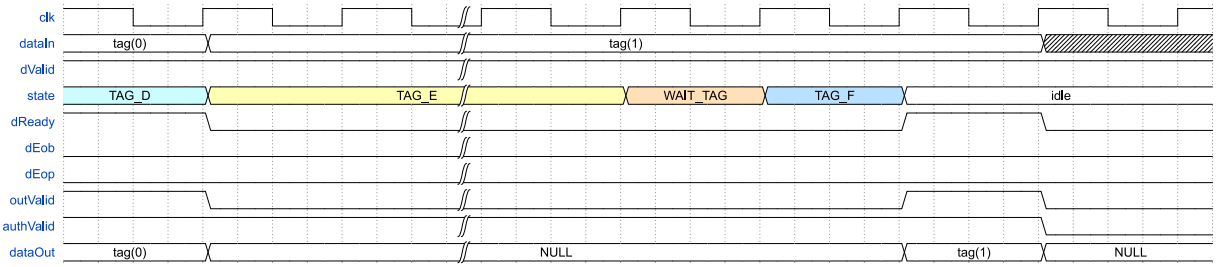
Como en el caso del Texto Plano/Cifrado la señal *outValid* indicará que en el bus de salida *output* se encuentra la palabra correspondiente del Tag generada durante el proceso de finalización.

Además en el caso de estar realizándose una operación de descifrado la señal *authValid* indicará si el Tag generado durante le cifrado coincide con el generado durante el descifrado, sirviendo esta señal como una señal de validación del proceso de descifrado.

En las Figuras 4.11a y 4.11b se muestra el diagrama temporal correspondiente a la verificación de las dos palabras que componen el Tag en el caso de que se este realizando la operación de descifrado.



(a) Procesado de la primera palabra del Tag durante una operación de descifrado.



(b) Procesado de la segunda palabra del Tag durante un operación de descifrado.

Figura 4.11: Diagrama temporal del procesado del Tag durante un operación de descifrado

Tratamiento de palabras parciales

En el procesado del Dato Asociado y del Texto Plano/Cifrado se puede dar el caso de que la última palabra sea una palabra parcial, con un tamaño inferior a 32 bits, la manera de proceder, conforme a la Figura 4.12,en este caso es la siguiente:

Tras la indicación de *dReady* además de asignar la ultima palabra al bus de entrada y las señales *dValid*, *dEob* y *dEop* como se indica en las secciones anteriores, se debe de asignar con un '1' la señal *partial* así como la señal *partialBits* se debe asignar con su valor correspondiente siguiendo la Tabla 4.14.

Tamaño de la palabra	Valor de partialBits
1 bytes	0b01
2 bytes	0b10
3 bytes	0b11
4 bytes	0b00

Tabla 4.14: Valores de la señal partialBits.

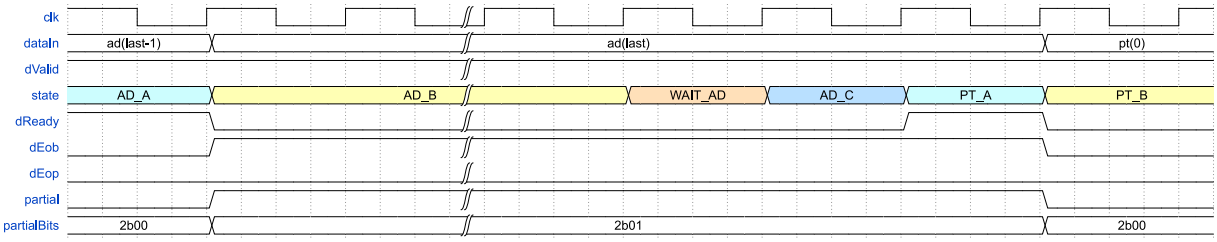


Figura 4.12: Diagrama temporal del procesado de una última palabra parcial del Dato Asociado.

4.3.3. Verificación Funcional

Como en el caso del Ascon, para verificar un correcto diseño del cifrador se han utilizado los vectores de test proporcionados por los diseñadores. Para ello se ha seleccionado un vector para cada tipo de operación que puede realizar el TinyJAMBU, las cuales son idénticas que en el Ascon y se muestran en la Tabla 4.15.

	Nonce	Dato Asociado	Texto Plano/Cifrado	MAC/Tag
Operación 1 (AEAD)	✓	✓	✓	✓
Operación 2 (AE)	✓	-	✓	✓
Operación 3	✓	-	-	✓
Operación 4	✓	✓	-	✓

Tabla 4.15: Tipo de operación a realizar por el TinyJAMBU en función de los bloques de datos procesados durante la operación.

Los vectores de test utilizados para la verificación de los diferentes tipos de operación que puede realizar el TinyJAMBU son:

Operación 1 (AEAD)

- *Clave*: 0xA589B134C5841EBC18C5E65682C61828
- *Nonce*: 0x632A9ED5A21DAE46E68879A2
- *Dato Asociado*: 0x278096CB
- *Texto Plano/Cifrado*: 0x5528B608/0xFE640955
- *MAC/Tag*: 0xA3133DCFCB9D2AA7

Operación 2 (AE)

- *Clave*: 0xD7F2483B0D018FAFC7DDFBA756905692
- *Nonce*: 0xC14E64B3A28A345A948059DC
- *Texto Plano/Cifrado*: 0x7C/0x22
- *MAC/Tag*: 0xCB7A0A1EC8AA341E

Operación 3

- *Clave*: 0x000102030405060708090A0B0C0D0E0F
- *Nonce*: 0x000102030405060708090A0B
- *MAC/Tag*: 0xED7B37CC6E9BDC7B

Operación 4

- *Clave*: 0x000102030405060708090A0B0C0D0E0F
- *Nonce*: 0x000102030405060708090A0B
- *Dato Asociado*: 0x0001020304050600
- *MAC/Tag*: 0x98787FBB9AFC6BD1

CAPÍTULO 5

TinyJAMBU Low Power

5.1. Introducción

En la actualidad el consumo de potencia de los dispositivos es uno de los factores más importantes en el diseño de dispositivos portátiles. Los cifradores descritos en secciones anteriores tienen como objetivo su implementación en este tipo de dispositivos, los cuales como se menciona anteriormente están teniendo un incremento muy elevado de su uso en aplicaciones de *IoT*, *wereables* y redes de sensores entre otros.

El diseño de los algoritmos presentados en este trabajo ya tiene como objetivo además de un consumo de recursos reducido, que también se vea reducido su consumo de potencia. Esto lo consiguen mediante el uso de permutaciones más sencillas que en los algoritmos que los preceden así como del uso de componentes muy sencillos, como el único NLFSR que emplea el TinyJAMBU para su permutación.

En este capitulo se proponen 4 variaciones de realización del NLFSR del TinyJAMBU que tienen como objetivo la reducción del consumo de potencia mediante la reducción de la actividad de conmutación del registro lo que da lugar a una reducción de consumo de potencia dinámica, ya que el consumo de potencia dinámico como se observa en la expresión 5.1 depende linealmente de la actividad de conmutación (α). En esta expresión, f es la frecuencia de operación, C la capacidad extraída del layout y V_{dd} la fuente de alimentación.

$$P_d = a \cdot f \cdot C \cdot V_{dd}^2 \tag{5.1}$$

Las modificaciones del NLFSR del cifrador se basan en la propuesta de *C.Piguet*, la cual consiste en la paralelización de un registro de manera que este se dividida en N registros iguales cuyo tamaño y frecuencia de operación serán N veces menor al original [9].

Los diseños propuestos realizan una paralelización parcial (*Mixed Parallel Low Power, MPLP*), o completa, (*Full Parallel Low Power, FPLP*), del NLFSR del cifrador paralelizado en 2 registros. Además del tipo de paralelización también existen dos alternativas según sea la señal de reloj de reloj de los registros paralelizados. Una es mediante el uso de una señal interna de *enable* que indique cual de los dos registros debe operar en un flanco activo de reloj, la otra alternativa es emplear una señal de reloj de frecuencia mitad respecto a la frecuencia del NLFSR original y que cada uno de los registros resultado de la paralelización opere con un flanco activo diferente.

Las diferentes propuestas con una breve descripción aparecen en la Tabla 5.1.

Propuesta	Descripción
FPLPEn	Paralelización completa del NLFSR que emplea una señal interna de *enable* para su operación
MPLPEn	Paralelización parcial del NLFSR que emplea una señal interna de *enable* para su operación
FPLP	Paralelización completa del NLFSR que emplea una señal de reloj de frecuencia mitad para su operación
MPLP	Paralelización parcial del NLFSR que emplea una señal de reloj de frecuencia mitad para su operación

Tabla 5.1: Propuestas de paralelización del NLFSR del TinyJAMBU.

5.2. Paralelización del NLFSR

La paralelización del NLFSR del TinyJAMBU presentado en la sección 4.1 consiste, en el caso de la alternativa *FPLP*, en la división del registro del NLFSR de 128 bits en dos registros de 64 bits cada uno, mientras que en la alternativa *MPLP* únicamente se paralelizan las porciones del registro sobre las que no se realiza ninguna carga en paralelo, estas son, teniendo en cuenta la especificación descrita en la sección 4.2, las comprendidas entre los bits 63 a 40 y 31 a 0.

Además de la paralelización del registro, será necesario generar, a partir de los registros paralelizados, los bits empleados para la realimentación del NLFSR. Esto se consigue mediante el empleo de multiplexores que seleccionan entre dos bits de cada registro y cuya señal de control será la señal de *enable* o la señal de reloj dividido (*clk_div*) dependiendo del tipo de alternativa a realizar.

En la Figura 5.1 se observa un esquema de la paralelización *FPLP*. Se aprecia la división del registro del NLFSR en dos registros de 64 bits en el que el superior contiene los bits impares y el inferior los pares del registro del NLFSR original. Destacan también los multiplexores empleados para la generación de los bits usados para el cálculo de la realimentación, siendo la señal de control de estos multiplexores la señal *en* en el caso de que se trate de la modificación con *enable* o *clk_div* en el caso de la que funciona con dos señales de reloj.

El esquema para la paralelización *MPLP* se puede observar en la Figura 5.2. En este esquema se aprecia como las porciones no paralelizadas se corresponden con aquellas comprendidas por los bits 127 a 64 y 39 a 32, las cuales se ven afectadas por operaciones de carga en paralelo que son indicadas en la sección 4.2. Como en el caso anterior destacan los multiplexores empleados para la generación de los bits necesarios para la realimentación del NLFSR así como para la entrada serie de la porción sin paralelizar que comprende los bits del 39 al 32.

5.2.1. Paralelización Low Power con señal de *enable*

Respecto al diseño de la alternativo con señal de *enable*, el único modulo que se ha visto modificado de la jerarquía presentada en la sección 4.3.1 ha sido el módulo *stateReg*. En este caso los puertos de entrada y salida son idénticos a los de la Tabla 4.7, la única diferencia está en el uso de una señal interna de *enable* que se inicializa a '0' al producirse un reset asíncrono y que conmuta su valor cada vez que se produce una operación de desplazamiento en serie. Es esta señal la que indica qué registro paralelizado es el que se desplaza en cada ciclo de reloj.

5.2.2. Paralelización Low Power con señal de reloj dividida

Para esta alternativa también es necesaria la modificación del componente *stateReg*, el cual tendrá una entrada de reloj adicional (*clk_div*) respecto a lo mostrado en la Tabla 4.7. En este caso los registros paralelizados realizan una operación de desplazamiento serie en el flanco de subida si se trata del registro impar y en el de bajada si se trata del registro par.

Además de esta modificación también es necesario generar una señal interna en el módulo *TinyJAMBUTop* y que servirá como entrada del módulo *TinyJAMBUContol* cuya función es la de indicar cuando se debe comenzar a realizar una permutación con el objetivo de que el primer bit de realimentación se asigne en el primer ciclo de reloj al bit correspondiente al bit 127 del NLFSR original.

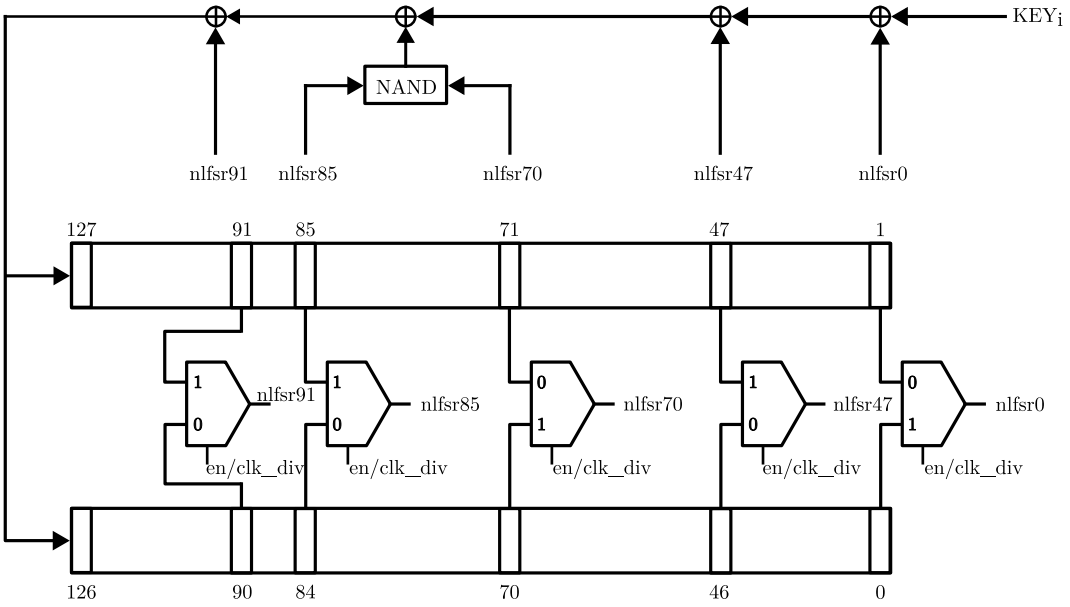


Figura 5.1: Paralelización *FPLP* del NLFSR del TinyJAMBU.

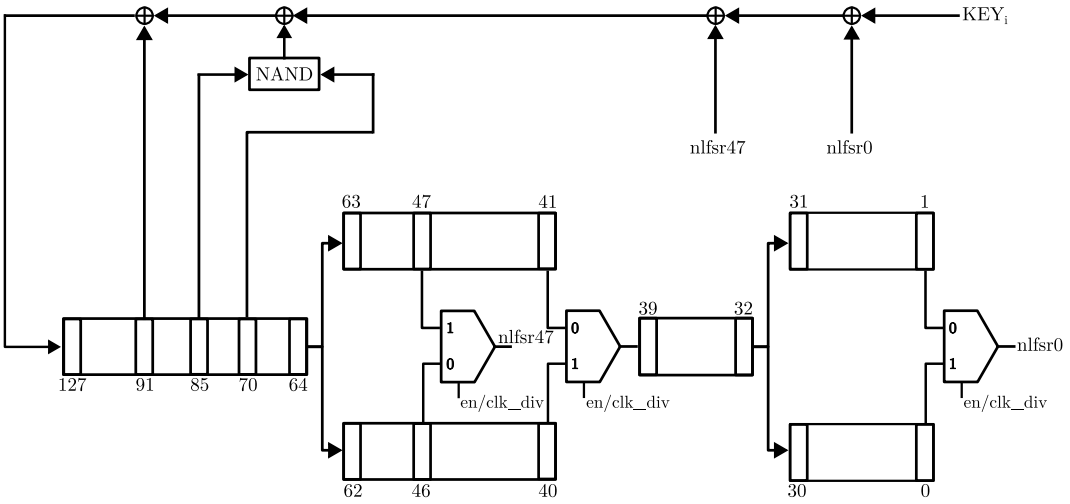


Figura 5.2: Paralelización *MPLP* del NLFSR del TinyJAMBU.

5.3. Verificación Funcional

Para verificar el funcionamiento de los diseños paralelizados se han empleado los mismos vectores de test proporcionados por el NIST que en la sección 4.3.3.

Operación 1 (AEAD)

- *Clave*: 0xA589B134C5841EBC18C5E65682C61828
- *Nonce*: 0x632A9ED5A21DAE46E68879A2
- *Dato Asociado*: 0x278096CB
- *Texto Plano/Cifrado*: 0x5528B608/0xFE640955
- *MAC/Tag*: 0xA3133DCFCB9D2AA7

Operación 2 (AE)

- *Clave*: 0xD7F2483B0D018FAFC7DDFBA756905692
- *Nonce*: 0xC14E64B3A28A345A948059DC
- *Texto Plano/Cifrado*: 0x7C/0x22
- *MAC/Tag*: 0xCB7A0A1EC8AA341E

Operación 3

- *Clave*: 0x000102030405060708090A0B0C0D0E0F
- *Nonce*: 0x000102030405060708090A0B
- *MAC/Tag*: 0xED7B37CC6E9BDC7B

Operación 4

- *Clave*: 0x000102030405060708090A0B0C0D0E0F
- *Nonce*: 0x000102030405060708090A0B
- *Dato Asociado*: 0x0001020304050600
- *MAC/Tag*: 0x98787FBB9AFC6BD1

CAPÍTULO 6

Implementación Hardware

6.1. Flujo de diseño y herramientas utilizadas

Tanto para la implementación en FPGA como en ASIC de todos los diseños descritos en los Capítulos 3, 4 y 5, se ha seguido un flujo de diseño digital básico cuyos pasos se muestran en la Figura 6.1, en la cual se observa que tras finalizar cada uno de los procesos de simulación que aparecen y en el caso de que los resultados no hayan sido satisfactorios se deben de repetir alguno de los pasos previos, ya sea la modificación de las opciones de la herramienta de place&route, las restricciones del diseño en la etapa de síntesis o incluso la descripción hardware del diseño.

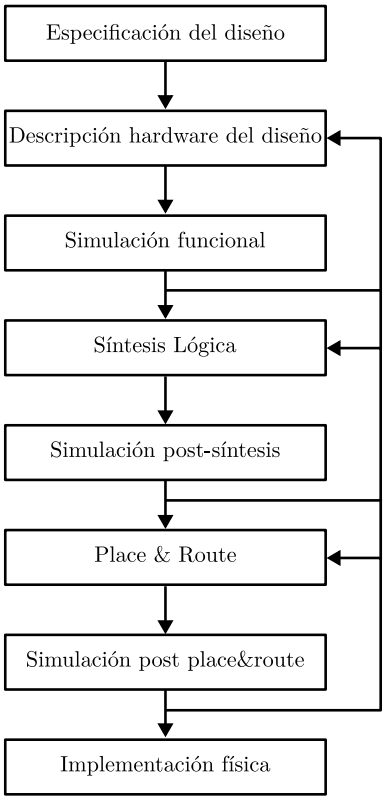


Figura 6.1: Flujo de diseño digital empleado.

Para la implementación física del cifrador en FPGA se ha empleado el sofware ISE Design Suite del fabricante Xilinx para todos los pasos descritos anteriormente, mientras que para la implementación en una tecnología de 65 nm del fabricante TSMC se ha empleado el suite de diseño de *Cadence*, utilizándose Genus para la síntesis del diseño, Innovus para el Layout e Incisive y SimVision para la simulación del diseño en las etapas de simulación funcional, post-síntesis y post-place&route.

6.2. Implementación en FPGA

En las Tablas 6.1 y 6.2 se muestra el consumo de recursos resultado de la implementación de los cifradores en un dispositivo xc7a100tcsg324-3 de la familia Artix-7 de Xilinx. Se puede observar como el TinyJAMBU tiene un consumo de recursos bastante inferior al del Ascon, lo que lo convierte en un candidato muy interesante a la hora de implementar un algoritmo de cifrado en un dispositivo con un número de recursos reducido. Esta información se ha obtenido de los informes proporcionados por el software Xilinx ISE tras realizar el proceso de Place & Route, el informe *Place and Route Report* proporciona la información de los recursos mientras que el informe *Post-PAR Static Timing Report* proporciona información acerca del mínimo período de reloj al que puede operar el diseño.

Además, en las Tablas 6.3 y 6.4 se muestran la comparación de potencia de las diferentes versiones del TinyJAMBU y de sus registros de estados respectivamente.

Para obtener estas medidas de potencia se ha generado un fichero SAIF (*Switching Activity Interchange Format*) a partir de una simulación *Post Place and Route*, para la cual se ha utilizado el simulador ISim del fabricante Xilinx, en la que se han aplicado los patrones indicados en la sección 5.3. Este fichero se utiliza como entrada para el software xPower de Xilinx, el cual proporciona los datos de potencia dinámica de la lógica empleada en cada diseño que se observan en las Tablas 6.3 y 6.4. En estas tabla se puede observar también el porcentaje de reducción de potencia de cada versión respecto a la versión original (TinyJAMBU1Bit).

Los resultados son bastantes favorables observándose una reducción de potencia en las versiones FPLP y MPLP, siendo las que utilizan dos relojes en las que se produce una reducción de potencia superior.

Ascon	Luts	Slices	Slice Regs	Min Period (ns)
Valor	1842	695	639	5,344 (187,126MHz)

Tabla 6.1: Prestaciones del cifrador Ascon128a en un dispositivo xc7a100tcsg324-3.

Cifrador	Luts	Slices	Slice Regs	Min Period (ns)
TinyJAMBU1Bit	331	111	322	3,575 (279,720MHz)
TinyJAMBU32Bit	328	126	319	4,555 (219,540MHz)
TinyJAMBUFPLPEn	351	140	323	3,543 (282,250MHz)
TinyJAMBUMPLPEn	309	103	322	4,292 (232,990MHz)
TinyJAMBUFPLP	342	143	323	4,003 (249,810MHz)
TinyJAMBUMPLP	311	105	322	5,290 (189,040MHz)

Tabla 6.2: Prestaciones del cifrador TinyJAMBU128 en un dispositivo xc7a100tcsg324-3.

Cifrador	$P_{Din.}$	% de reducción
TinyJAMBU1Bit	3,53mW	0 %
TinyJAMBU32Bit	3,44mW	2,550 %
TinyJAMBUFPLPEn	2,74mW	22,380 %
TinyJAMBUMPLPEn	3,12mW	11,615 %
TinyJAMBUFPLP	2,67mW	24,363 %
TinyJAMBUMPLP	3,09mW	12,465 %

Tabla 6.3: Medidas de potencia del cifrador TinyJAMBU128 en un dispositivo xc7a100tcsg324-3.

Cifrador	$P_{Din.}$	% de reducción
TinyJAMBU1Bit	1,34mW	0 %
TinyJAMBU32Bit	2,02mW	−50,746 %
TinyJAMBUFPLPEn	0,82mW	38,806 %
TinyJAMBUMPLPEn	0,86mW	35,821 %
TinyJAMBUFPLP	0,810mW	39,552 %
TinyJAMBUMPLP	0,800mW	40,299 %

Tabla 6.4: Medidas de potencia del registro de estados del cifrador TinyJAMBU128 en un dispositivo xc7a100tcsg324-3.

6.3. Implementación en ASIC

De forma similar al caso de la implementación en FPGA, en las Tablas 6.5 y 6.6 se puede observar el consumo de recursos en términos de numero de celdas y área del circuito, así como el mínimo periodo de reloj al que pueden operar estos diseños. Como en el caso de la implementación en FPGA, el diseño del TinyJAMBU consumo un numero de celdas muy inferior. Esta información se obtiene de los reportes de área y temporización del software Innovus de Cadence.

Las medidas de potencia que se muestran en la Tabla 6.7 muestran también una reducción de la potencia de conmutación, aunque en este caso se observa como las versiónes MPLP presentan una potencia de conmutación mayor que la versión original.

Las medidas de potencia se han obtenido en este caso a partir un fichero VCD (*Value Change Dump*) generado mediante una simulación *Post Place and Route* usando el software Incisive de Cadence. Este fichero contiene la información de conmutación del circuito durante la simulación, en la que se han ejecutado las operaciones de cifrado y descifrado de los vectores de test que aparecen en la sección 5.3.

Para obtener la información de la potencia de conmutación se ha usado el software Innvous de Cadence el cual utiliza la información del circuito y el fichero VCD para realizar una estimación de la potencia de conmutación del mismo.

Ascon	Celdas	Área de celdas (μm^2)	Área Total (μm^2)	Min Period (ns)
Valor	3638	16497,72	30147,878	8,3(120,482MHz)

Tabla 6.5: Prestaciones del cifrador Ascon128a diseñado en una tecnología de 65 nm de TSMC.

Cifrador	Celdas	Área de celdas (μm^2)	Área Total (μm^2)	Min Period (ns)
TinyJAMBU1Bit	920	4872,920	8593,11	4,6(217,390MHz)
TinyJAMBU32Bit	963	5175,720	8605,563	5,1(196,070MHz)
TinyJAMBUFPLPEn	969	4980,600	8612,563	4,7(212,766MHz)
TinyJAMBUMPLPEn	969	5000,76	8888,268	4,8(208,333MHz)
TinyJAMBUFPLP	960	4979,16	8877,570	5,2(194,140MHz)
TinyJAMBUMPLP	1137	5612,4	8974,103	6,8(146,700MHz)

Tabla 6.6: Prestaciones del cifrador TinyJAMBU128 diseñado en una tecnología de 65 nm de TSMC.

En las Figuras 6.2 a 6.3 se pueden observar capturas del layout de los cifradores Ascon y las diferentes versiones del TinyJAMBU.

Cifrador	Potencia Total	Potencia de conmutación	% de reducción
TinyJAMBU1Bit	0,63905mW	0,34283mW	0 %
TinyJAMBU32Bit	0,64108mW	0,37866mW	−10,451 %
TinyJAMBUFPLPEn	0,60547mW	0,31156mW	9,121 %
TinyJAMBUMPLPEn	0,65810mW	0,35107mW	−2,404 %
TinyJAMBUFPLP	0,39536mW	0,24059mW	29,822 %
TinyJAMBUMPLP	0,59207mW	0,34710mW	−1,246 %

Tabla 6.7: Medidas de potencia del cifrador TinyJAMBU128 diseñado en una tecnología de 65 nm de TSMC.

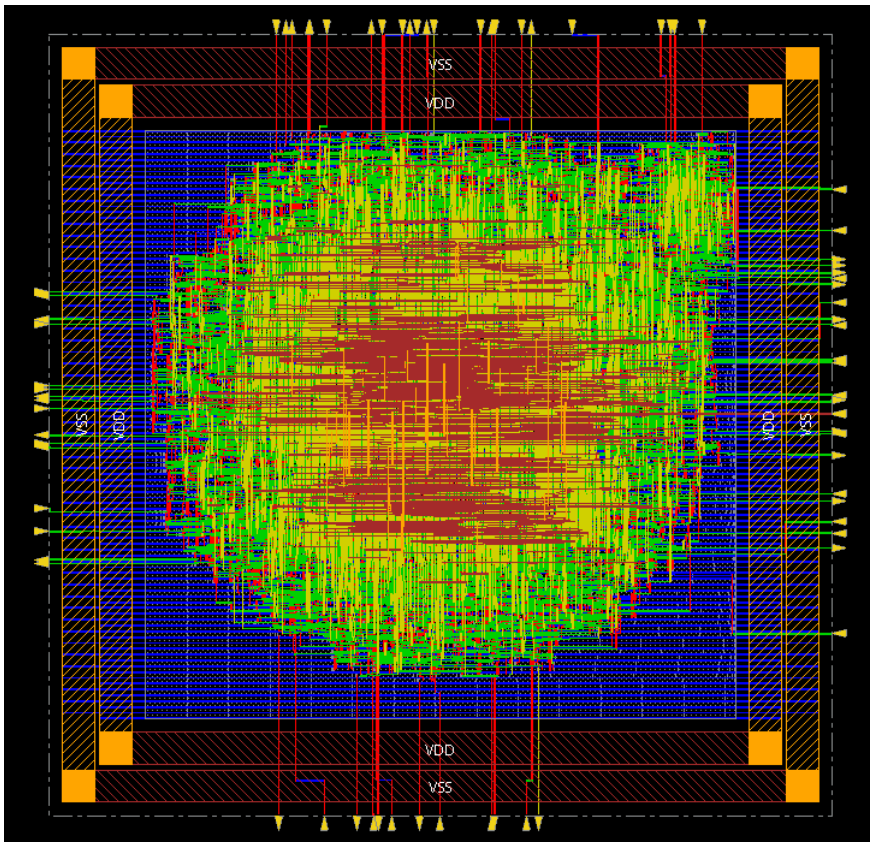


Figura 6.2: Layout de la implementación del cifrador Ascon.

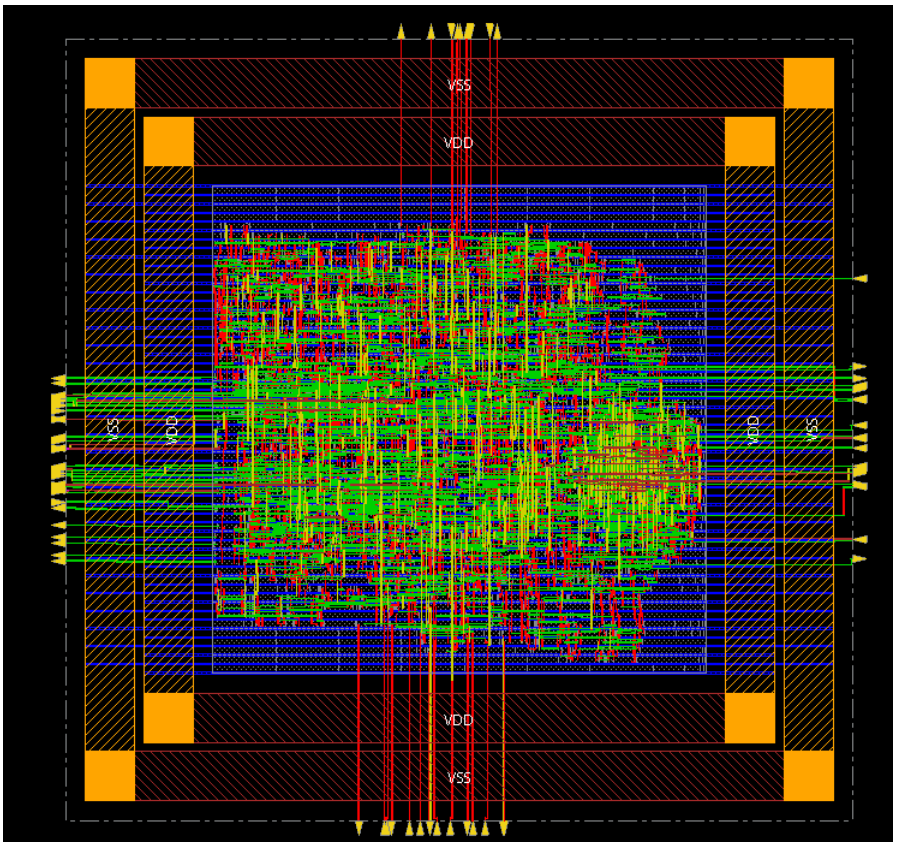


Figura 6.3: Layout de la implementación del cifrador TinyJAMBU.

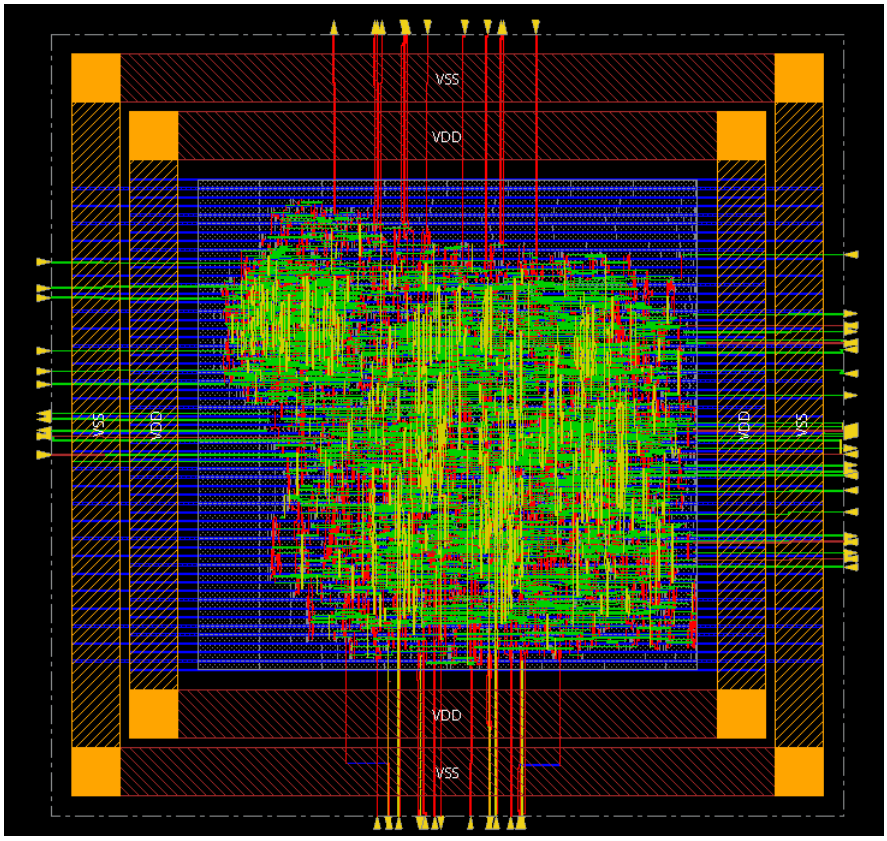


Figura 6.4: Layout de la implementación del cifrador TinyJAMBU cuyo NLFSR se ha paralelizado para realizar 32 permutaciones simultaneas.

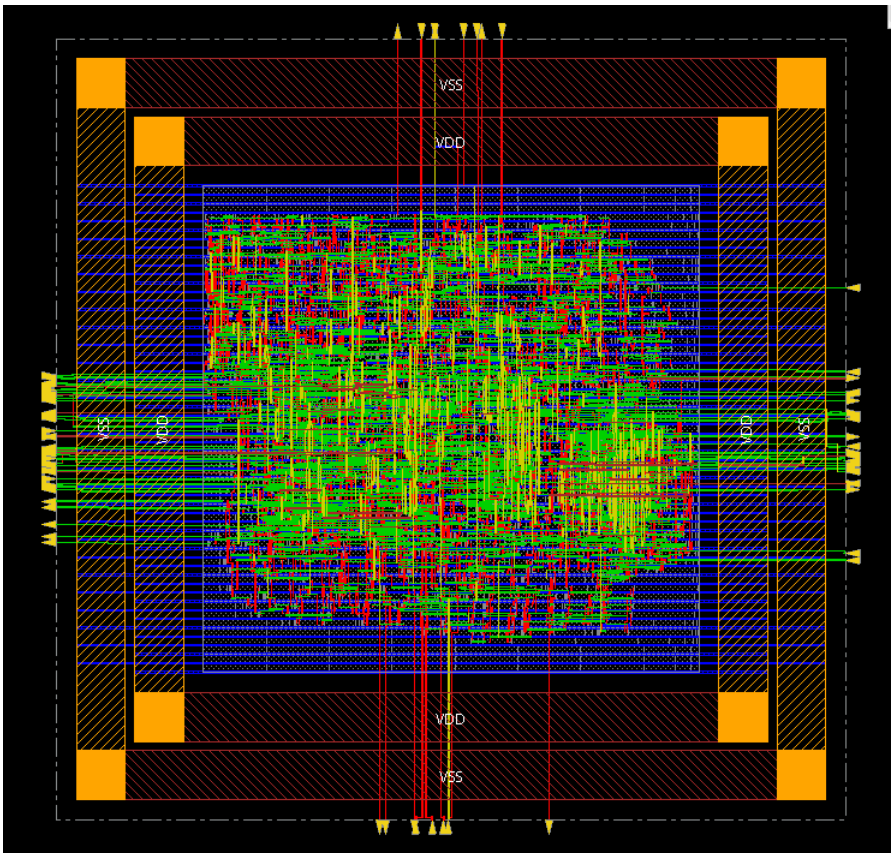


Figura 6.5: Layout de la implementación del cifrador TinyJAMBUFPLP con señal de enable.

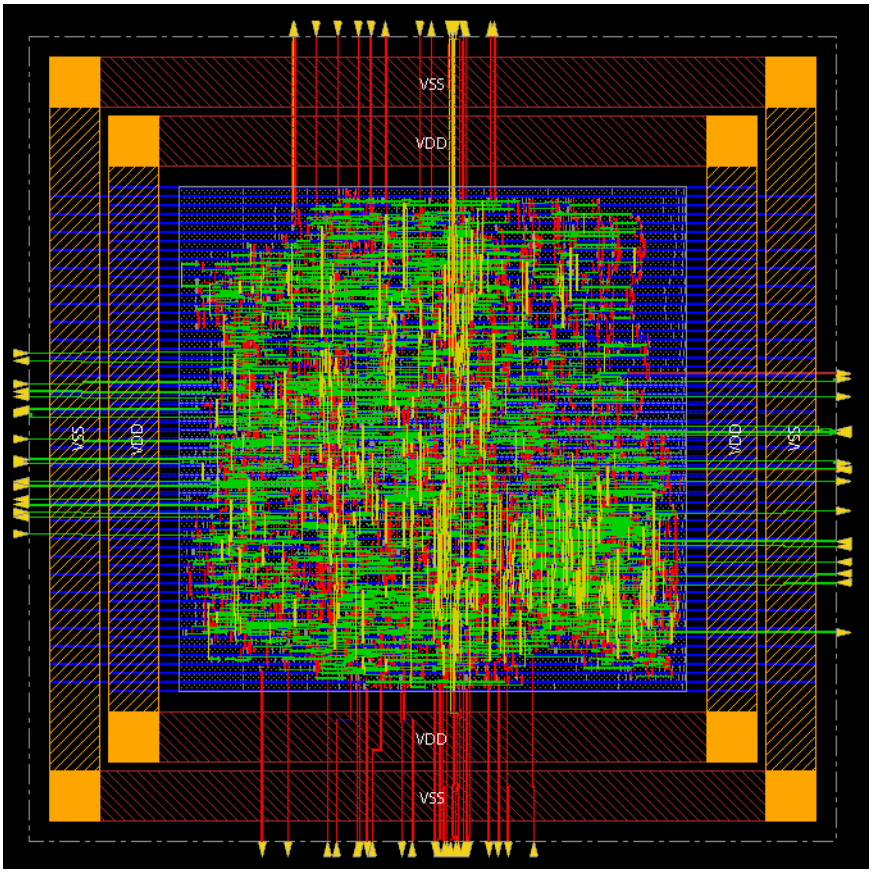


Figura 6.6: Layout de la implementación del cifrador TinyJAMBUMPLP con señal de enable.

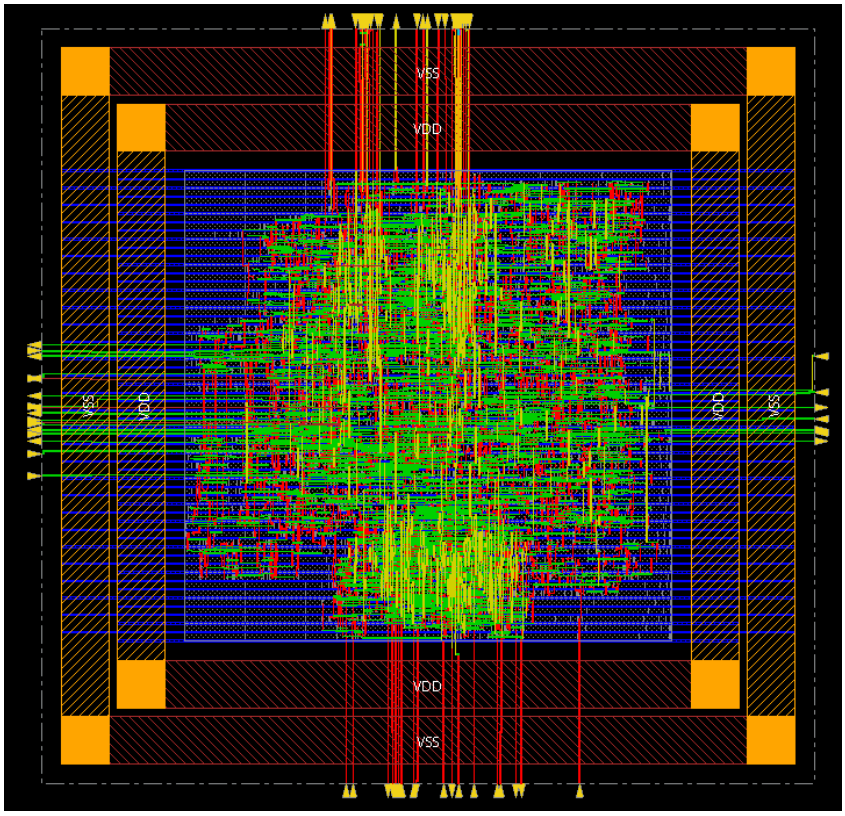


Figura 6.7: Layout de la implementación del cifrador TinyJAMBUFPLP que trabaja con dos señales de reloj.

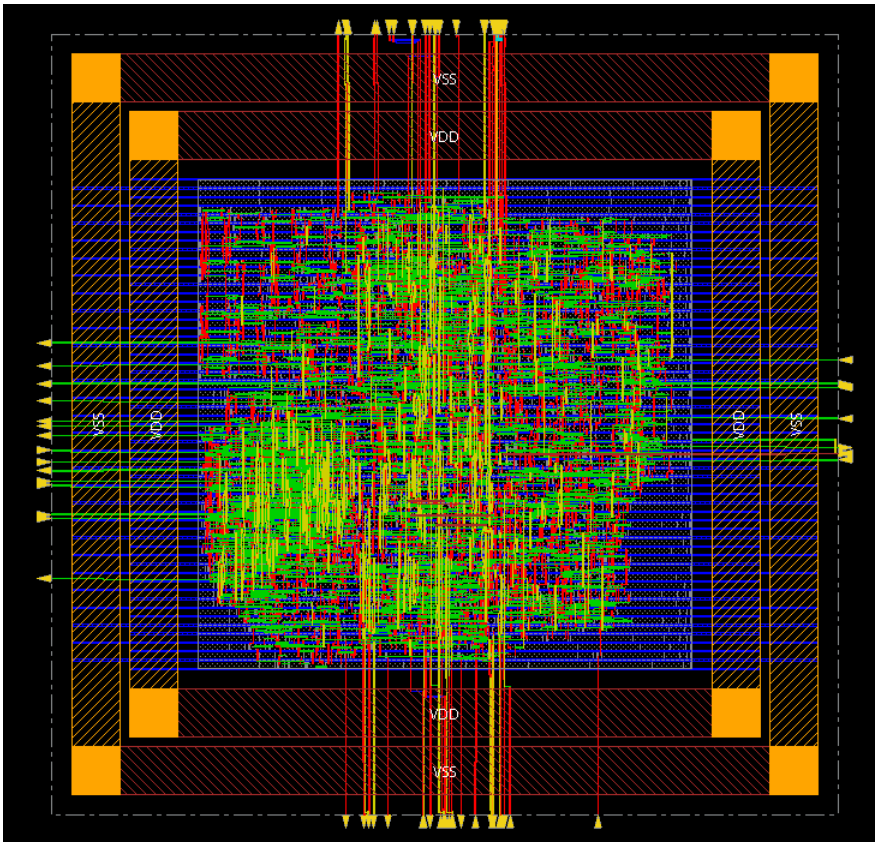


Figura 6.8: Layout de la implementación del cifrador TinyJAMBUMPLP que trabaja con dos señales de reloj.

CAPÍTULO 7

Conclusiones y Futuros Trabajos

En este trabajo se han realizado una de serie de tareas y se han obtenido una serie de resultados que a continuación se detallan:

Se han estudiado los distintos tipos de cifrado existentes en la actualidad y se ha centrado el estudio especialmente sobre los cifradores lightweigh AEAD y las propuestas que en los últimos años se han presentado en los concursos y proyectos convocados al efecto.

Se han analizado los cifradores finalistas del proyecto de criptografía lightweight del NIST y se han seleccionado dos de ellos para su implementación hardware, el cifrador Ascon y el cifrador TinyJAMBU.

Se ha estudiado el algoritmo del cifrador Ascon y se ha realizado una implementación hardware de dicho cifrador. El diseño se ha realizado en VHDL cumpliendo las restricciones impuestas por las principales herramientas de síntesis. Se ha definido una interfaz para el control de su funcionamiento y se han realizado verificaciones para comprobar el correcto funcionamiento del cifrador. Se han utilizado los datos de test ofrecidos por los autores del algoritmo y se ha comprobado el correcto funcionamiento del diseño.

Se ha estudiado el algoritmo del cifrador TinyJAMBU y se ha realizado una implementación hardware de dicho cifrador. Esta implementación se ha parametrizado de forma que la operación de desplazamiento que hay que realizar en el registro de estado pueda realizarse en 2, 4, 8, 16 o 32 bits. La descripción también se ha realizado en VHDL y cumpliendo con las restricciones de las principales herramientas de síntesis. La verificación se ha llevado a cabo utilizando los datos ofrecidos por los autores del algoritmo.

Se han llevado a cabo implementaciones hardware de los diseños realizados en tecnologías FPGA y ASIC. Para la tecnología FPGA se ha utilizado un dispositivo de la familia Artix de Xilinx (Serie 7) mientras que para la implementación ASIC se ha utilizado la tecnología TSMC-65 nm. En los dos casos se ha realizado un análisis de los recursos consumidos y de la máxima frecuencia de operación.

Para los diseños del cifrador TinyJAMBU se ha realizado un análisis del consumo de potencia, tanto de la versión normal como de las cuatro versiones que tienen aplicada la técnica de reducción del consumo de potencia. Este análisis se ha llevado a cabo tanto en la tecnología FPGA como en la tecnología ASIC empleando ficheros con formato SAIF y VCD respectivamente.

Sobre la tecnología FPGA se observa una reducción del consumo dinámico de las versiones paralelizadas FPLP y MPLP del TinyJAMBU, siendo las versiones FPLP las que ofrecen una mayor reducción de potencia. También se puede observar como aquellas versiones que emplean dos relojes en lugar de señal de enable son las que ofrecen un porcentaje de reducción de potencia superior.

En el caso de la tecnología ASIC los resultados obtenidos muestran únicamente una reducción de la potencia en la versión FPLP del cifrador, mientras que la versión MPLP llega incluso a tener una potencia superior a la versión original. En el caso de la alternativa FPLP también es la versión que emplea dos relojes la que presenta una mayor reducción de potencia.

Este trabajo ha permitido aplicar de forma práctica conocimientos adquiridos en este máster como son:

La realización de diseño hardware digital.

La utilización de flujos de diseño y de herramientas tanto para tecnologías FPGA como ASIC.

El conocimiento de las fuentes de consumo de potencia y la aplicación de técnicas de reducción de dicho consumo.

Futuros Trabajos

Como futuros trabajos se propone realizar las medidas experimentales del consumo de potencia de los diseños realizados durante el trabajo. Para el caso de tecnologías FPGA se cuenta con una placa como es la Sakura que está preparada para la realización de este tipo de medidas. Para tecnologías ASIC se contará en un futuro próximo con un circuito integrado en el que se ha colaborado en la fase de diseño y que incluye los cifradores TinyJAMBU desarrollados en el marco de este trabajo. Este ASIC ha sido desarrollado en el marco del proyecto ARES (PID2020-116664RB-I00) y está fabricado en la tecnología TSMC de 65 nm.

Bibliografía

[1] C. Paar and J. Pelzl, *Understanding Cryptography*, 2010.

[2] L. H. Encinas, *La Criptografía*, 2016.

[3] NIST, "Data encryption standard (des)," 1999.

[4] NIST, "Advanced encryption standard (aes)," 2001.

[5] C. D. Cannière and B. Preneel, "Trivium specifications," 2008.

[6] NIST, "Recommendation for block cipher modes of operation: Methods and techniques," 2001.

[7] F. M. Christoph Dobraunig, Maria Eichlseder and M. Schläffer, "Ascon," 2021.

[8] H. Wu and T. Huang, "Tinyjambu: A family of lightweight authenticated encryption algorithms (version 2)," 2021.

[9] C. Piguet, "Logic desgin for low-power cmos circuits," 1995.