Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Reproducción de ataque de fallos en el algoritmo Advanced Encryption Standard utilizando simulación HDL

Autor: Paola Serrano Pinilla

Tutor: Hipólito Guzmán Miranda

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Reproducción de ataque de fallos en el algoritmo Advanced Encryption Standard utilizando simulación HDL

Autor:
Paola Serrano Pinilla

Tutor:
Hipólito Guzmán Miranda
Profesor Contratado Doctor

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Carrera: Reproducción de ataque de fallos en el algoritmo Advanced Encryption Standard utilizando simulación HDL

Autor: Paola Serrano Pinilla

Tutor: Hipólito Guzmán Miranda

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Dedico este trabajo a mi familia como reconocimiento a los esfuerzos que han hecho para permitirme estudiar esta carrera. También darles las gracias por su apoyo incondicional, por levantarme cada vez que me he caído y por ser el motor de mi vida. Quiero agradecer a mis amigos, sobre todo a los que han sido compañeros de viaje en este camino, por ser siempre un pilar fundamental en esta carrera de fondo. Por último, a mi tutor, por ser siempre comprensivo y ayudarme en todo lo que he necesitado y en lo que todavía no sabía que necesitaba.

Paola Serrano Pinilla

Sevilla, 2018

Resumen

Hoy en día, los algoritmos criptográficos están presentes en prácticamente todas las aplicaciones computacionales. Normalmente pasan desapercibidos a la mayoría de usuarios pero están presentes en muchas de las operaciones que se realizan de forma cotidiana, sobre todo a través de internet, como son el almacenamiento de datos confidenciales, las comunicaciones seguras, el control de accesos o las tarjetas inteligentes (Smart cards) de las que se sirven los bancos. En multitud de ocasiones y con vistas a aumentar la velocidad de cálculo y el consumo de potencia, estos algoritmos se implementan en procesadores hardware como, por ejemplo, las citadas Smart cards.

El principal problema de estas implementaciones hardware es que son susceptibles a ser vulneradas mediante los denominados ataques de fallos. En este tipo de ataques, el atacante corrompe la implementación física para modificar el estado interno del dispositivo, por ejemplo, con proyección láser sobre el chip de silicio. Gracias a este tipo de errores inducidos, el atacante puede obtener información de utilidad a través de la comparación de parejas de textos encriptados con y sin error. En concreto, el ataque busca inferir información acerca de los bits de la clave secreta que, en la criptología actual, es la máxima responsable del grado de seguridad del sistema.

Este trabajo consiste en la implementación de un ataque de fallos de tipo "Differential Fault Analysis" sobre el algoritmo criptográfico más usado en la actualidad, el Advanced Encryption Standard (AES). El ataque será reproducido en simulación con el fin de determinar la viabilidad del mismo antes de llevarlo a cabo en un entorno real y físico como una FPGA (Field Programmable Gate Array), este último paso se contemplará como posible trabajo futuro. El trabajo concluirá con unos comentarios acerca de la efectividad del ataque y con una recopilación de las posibles medidas de seguridad que se podrían implementar para proteger el sistema.

Abstract

Nowadays, cryptographic algorithms are incorporated in practically all computational applications. Usually they are unnoticed by most of the users, but they are present in many of the operations that are performed daily, especially through the Internet, such as the storage of confidential data, secure communications, access control or the smart cards used by banks. On many occasions and to increase the calculation speed and the power consumption, these algorithms are implemented in hardware processors such as -the aforementioned- Smart cards.

The main problem with these hardware implementations is that they are susceptible to being violated through fault attacks. In this type of attack, the attacker corrupts the physical implementation to modify the internal state of the device, for example, with laser projection on the silicon chip. Thanks to this type of induced errors, the attacker can obtain useful information through the comparison of pairs of texts encrypted with and without error. The attack seeks, in particular, to infer information about the bits of the secret key that, in current cryptology, is the maximum responsible for the system security.

This project consists in the implementation of a fault attack so-called "Differential Fault Analysis" on the most widely used cryptographic algorithm, the Advanced Encryption Standard (AES). The attack will be reproduced in simulation in order to evaluate its feasibility before carrying it out in a real and physical environment such as an FPGA (Field Programmable Gate Array), this last step will be considered as a possible future work. The project will conclude with some comments about the effectiveness of the attack and a summary of possible security measures that could be implemented to protect the system.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

No confíes tu secreto ni al más íntimo amigo; no podrías pedirle discreción si tú mismo no la has tenido.

- Ludwig van Beethoven -

Este trabajo consiste en la reproducción de un ataque de fallos sobre un algoritmo criptográfico seguro, el Advanced Encryption Standard (AES). Se ha escogido dicho algoritmo como objeto de estudio del trabajo porque, hoy en día, es el más usado en el mundo en ámbitos de seguridad y, por tanto, resulta extremadamente interesante desentrañar sus vulnerabilidades y estudiar las posibles medidas de seguridad que le son aplicables.

A lo largo del proyecto se explicarán detenidamente las etapas de cifrado del algoritmo y se estudiarán los ataques más exitosos realizados hasta la fecha con el fin de entender las vulnerabilidades del criptosistema y aprender a protegerlo.

1.1 Objetivos del Trabajo

El objetivo principal del proyecto es llevar a cabo la reproducción en simulación de un ataque de fallos sobre el Advanced Encryption Standard para, en base a él, proponer medidas de seguridad frente a este tipo de ataques denominados Differential Fault Analysis.

Para conseguir burlar la seguridad del sistema y extraer la clave secreta, es de vital importancia conocer a fondo la estructura interna del algoritmo, así como los ataques más exitosos publicados. De esta forma se podrán apreciar los puntos débiles del sistema y se podrá elegir, en consecuencia, uno de los ataques estudiados como base para la implementación del ataque que se lleva a cabo en este trabajo.

Nótese que el objetivo final de este proyecto no es el romper la seguridad del sistema como sería propio de atacantes maliciosos sino, gracias a ello, proponer medidas reales y eficientes para protegerlo.

1.3 Alcance del trabajo

Como se ha comentado antes, este trabajo presenta una simulación en lenguaje VHDL del ataque escogido y, por tanto, deja el camino abierto a su continuación con la implementación del ataque estudiado en un entorno real como la FPGA. El trabajo se puede considerar entonces como un

estudio en profundidad de las vulnerabilidades del sistema y de la viabilidad del ataque en sí. Sin embargo, no se le puede restar importancia pues, las tareas desarrolladas en este proyecto, son imprescindibles para poder implementar con éxito el ataque en un sistema real.

1.4 Organización de la memoria

El presente trabajo se ha organizado en 7 capítulos que se han pretendido ordenar de forma que faciliten la total comprensión del objeto de estudio y de su desarrollo a lo largo del proyecto. Son los siguientes:

- Introducción

Este capítulo está dedicado a presentar el tema central del trabajo, así como el alcance y los objetivos que se han perseguido durante su desarrollo.

- Revisión histórica y conceptos básicos

Este capítulo está destinado a recalcar lo importante que ha sido y que es la criptografía desde tiempos inmemoriales hasta nuestros días. En primer lugar, definiendo en que consiste esta ciencia y exponiendo su desarrollo a lo largo de la historia atendiendo a cómo ha sabido adaptarse a las necesidades de cada periodo. A continuación, se explican conceptos básicos para el entendimiento del trabajo como son los tipos de criptografía y el criptoanálisis y, por último, se ha desarrollado el nacimiento del AES y cómo llego a ser el algoritmo de cifrado en software y hardware por excelencia.

- Revisión del algoritmo

Este capítulo está dedicado al estudio y análisis del algoritmo en sí, es decir, en qué consiste y cómo funciona. Para ello, se han detallado todas las operaciones y transformaciones que implementa tanto sobre el texto como sobre la clave secreta. Para finalizar se ha diseñado un diagrama resumen que recoge todas ellas en el orden en el que son efectuadas.

- Ataques de fallos sobre AES

Este capítulo comienza con una introducción teórica a los principales tipos de ataques que se llevan a cabo sobre el AES. A continuación, se inicia la descripción de algunos de los ataques más efectivos que se han publicado hasta la fecha para, finalmente, concluir con el que ha sido establecido como modelo para la fase de simulación.

- Reproducción del ataque de fallos en simulación

En este capítulo se recogen todos los pasos a seguir para la implementación del ataque elegido en simulación con lenguaje HDL. Primero, se exponen los requisitos previos, después, se explica cómo se ha diseñado el código que implementa el ataque y qué se pretende conseguir con él y, por último, se hace una valoración de los resultados y experiencias.

- Medidas de protección

Este capítulo recoge algunas de las medidas de protección frente a ataques de fallos más implementadas en el AES actualmente.

- Conclusiones y trabajos futuros

Este capítulo está dedicado a las conclusiones que se han podido extraer del desarrollo del trabajo, así como los posibles trabajos futuros a los que abre la puerta este proyecto.

2 REVISIÓN HISTÓRICA Y CONCEPTOS BÁSICOS

El objetivo de este capítulo es hacer un repaso por la historia de la criptografía, así como, definir los principales conceptos teóricos que se han de conocer para alcanzar la completa compresión de este trabajo.

2.1 Definición de criptografía

La criptografía es un ámbito de la criptología que se dedica a alterar las representaciones lingüísticas de un mensaje mediante procedimientos o claves secretas de un modo enigmático, es decir, de forma que el texto alterado solo sea inteligible para aquel que conozca las claves para descifrarlo. Los objetivos principales de la criptografía son: garantizar la confidencialidad del mensaje, la integridad de la información, la posibilidad de vinculación de un documento o transacción a una persona o sistema de gestión y la autenticación del comunicador [1].

2.2 Revisión histórica

El nacimiento de la criptografía se remonta miles de años atrás. El ser humano ha sentido desde tiempos inmemoriales, la necesidad de ocultar secretos o esconder información a intrusos o enemigos. Las primeras técnicas criptográficas son casi tan antiguas como la escritura y fueron implementadas por algunas de las primeras civilizaciones.

2.2.1 Criptografía clásica

Sus primeras aplicaciones estuvieron relacionadas con el envío de mensajes entre tropas aliadas durante tiempos de guerra. De esta forma, aunque un mensajero fuera interceptado por el enemigo, éste sería incapaz de obtener información de valor de él ya que, la clave para descifrar el mensaje no estaba en conocimiento del encargado de transportarlo [2].

Con este fin, los espartanos crearon, en el 400 a.C., el primer sistema de criptografía por transposición, la Escítala. Este sistema consistía en un cilindro de madera, de un determinado diámetro, al que se enrollaba una cinta de cuero y sobre la que se escribía el mensaje siguiendo su dimensión longitudinal. De esta forma, al desenrollar la cinta, las letras aparecían desordenadas. Para leer el mensaje era necesario que el receptor contara con un cilindro del mismo diámetro que original sobre el cual, enrollando la cinta, se podía volver a leer el mensaje. Este tipo de criptosistema se denomina de transposición porque las letras del mensaje no son sustituidas por otros caracteres, sino que sólo se altera el orden de las mismas [3].

Un par de siglos más tarde, surgieron los sistemas de criptografía por sustitución en los que, cada letra del mensaje correspondía a uno o más caracteres [4]. Algunos ejemplos, según el método de sustitución, fueron el Tablero de Polibio (200 a.C.) en el que la sustitución se llevaba a cabo con una tabla, el Cifrado del César (100 a.C.) en el que cada letra se sustituía por la que se encuentra tres

posiciones más adelante en el alfabeto o, más tarde, el Disco de Alberti (1466 d.C.) en el que la sustitución se llevaba a cabo según dos discos concéntricos, ambos móviles. Éste último supuso un gran avance para la época.

Figura 1. Escítala espartana

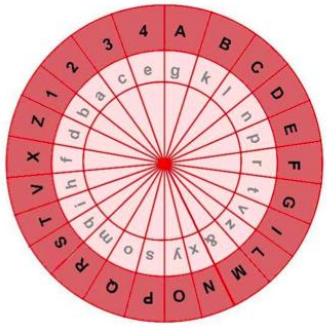

Figura 2. Disco de Alberti

Durante el periodo transcurrido entre estos descubrimientos y la Segunda Guerra Mundial, el interés por la criptografía fue en aumento motivado, sobre todo, por los conflictos bélicos que tuvieron lugar durante la fecha. Sin embargo, la mayoría de los criptosistemas empleados no dejaban de ser modelos de sustitución poli-alfabéticos relativamente básicos. Algunos ejemplos son el cifrado de Cuatro Cuadros que se servía de cuatro matrices alfabéticas para realizar la sustitución o el, un poco más elaborado, cifrado ADFGVX desarrollado por los alemanes para la Primera Guerra Mundial que, con una matriz de 7x7, sustituía cada letra del mensaje por la A, la D, la F, la G, la V o la X [5].

En el siglo XX el inventor alemán Arthur Scherbius se propuso sustituir los rudimentarios métodos de cifrado utilizados por Alemania durante la Primera Guerra Mundial, recurriendo a la tecnología en lugar de al papel y al lápiz [4]. Con este fin y junto con Richard Ritter, desarrolló la máquina de rotores electromagnética que se conoció como Enigma. Ésta era muy semejante a una máquina de escribir ya que se componía de un teclado, una unidad modificadora y un tablero en el que se mostraba el mensaje cifrado. Cada letra del texto a cifrar se escribía tal cual sobre el teclado y la unidad modificadora se encargaba de transformarla en otra diferente antes de mostrarla en el tablero. Su funcionamiento consistía en que cada vez que se pulsaba una letra, la unidad modificadora giraba un veintiseisavo de vuelta (para un alfabeto de 26 letras) y mostraba en el tablero otra diferente. Con este sistema, el texto final era bastante fácil de descifrar, por lo que, Scherbius decidió incluir una segunda modificadora que giraba una posición por cada vuelta completa de la primera.

Algo más tarde se incluyó un tercer disco modificador que giraba una posición cuando el segundo daba una vuelta completa, de esta forma, se conseguía una clave de:

$$26*26*26=17576 \text{ posiciones}$$

La máquina contaba, además, con otro componente, el Reflector que era la clave para descifrar el mensaje. Para conseguirlo, el receptor, debía contar con otra máquina enigma con los rotores situados en la misma posición que la emisora y, a través del Reflector, introduciendo el mensaje cifrado en el tablero se mostraba el original.

Finalmente, se añadieron dos características más: en primer lugar, se hicieron los rotores intercambiables y, como tenían 6 posiciones posibles, la envergadura de la clave aumentaba aún más y, por último, se introdujo un clavijero mediante el cual se podían intercambiar letras en grupos de 6.

Así se consiguió que la máquina ofreciera billones de posibilidades de clave que, además, los alemanes cambiaban cada día durante la Segunda Guerra Mundial dando a sus enemigos sólo 24 horas para descifrar. Se convirtió en el sistema de cifrado más seguro del momento.

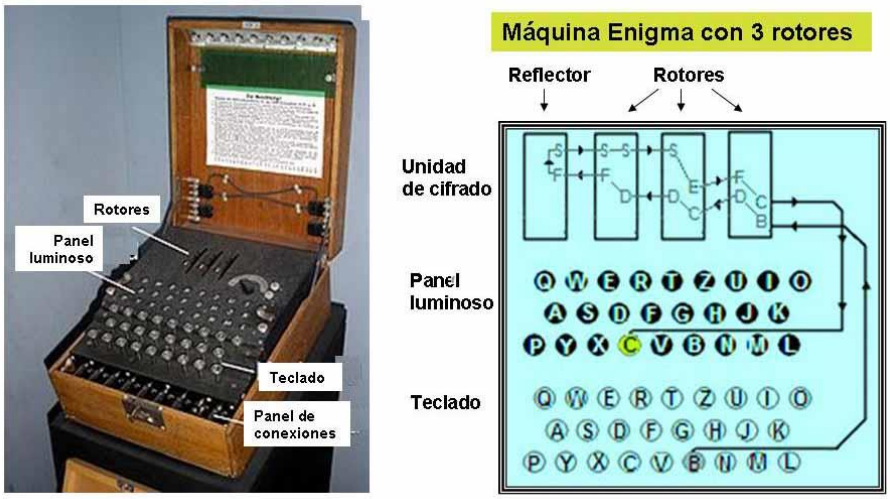

Figura 3 . Máquina Enigma

Finalmente, criptólogos polacos en colaboración con los británicos consiguieron romper el cifrado de la máquina y, desde ese entonces, se empezó a dar gran relevancia al estudio teórico de la criptografía.

2.2.2 Criptografía moderna

Lo que se conoce por criptografía moderna surgió tras la Segunda Mundial cuando, en 1949, el ingeniero y matemático estadounidense Claude Shannon publicó un trabajo sobre la teoría de la información al que luego siguieron otros tantos. En dicho documento, profundizó en los aspectos teóricos de la criptografía dándole, por fin, un marco científico. A esto, hay que sumarle el nacimiento de la computación y el hecho de que, a partir de este momento, la información clasificada estaba representada por unos y ceros (bits) en lugar de por caracteres. Además, por primera vez, surge la necesidad de hacer llegar estas técnicas a la población civil, no solo a los organismos gubernamentales o al ejército. Esto es debido al cada vez más generalizado uso de la tecnología informática y el constante aumento en el volumen del flujo de información.

Los sistemas criptográficos modernos se dividen en tres grandes grupos:

- Criptosistemas de clave privada o simétricos
- Criptosistemas de clave pública o asimétricos
- Criptosistemas híbridos

2.3 Conceptos básicos

Este apartado pretende explicar de forma medianamente resumida los conceptos teóricos que están estrechamente ligados al proyecto. Al inicio del capítulo ya se introdujo la definición de criptografía, ahora procederemos a ver en qué tipos se clasifica y en qué consiste el criptoanálisis.

2.3.1 Tipos de criptografía

La criptografía puede clasificarse según diferentes criterios. Atendiendo a su relación con la historia en criptografía clásica y moderna, como hemos destacado antes y, dentro de la criptografía moderna, según el tipo de clave utilizada para el cifrado en criptografía simétrica, asimétrica o híbrida. En este apartado van a desarrollarse estas últimas.

2.3.1.1 Criptografía simétrica

El principio básico de la criptografía simétrica es que la clave con la que se cifra el mensaje es la misma que se utiliza para su descifrado. Por tanto, dicha clave ha de ser acordada de antemano entre emisor y receptor y justo ahí reside el mayor problema de este tipo de sistemas.

En los sistemas de clave simétrica los algoritmos de cifrado son de dominio público o código abierto y, por tanto, el nivel de seguridad de la comunicación reside sola y exclusivamente en la complejidad de la clave. Ésta debe ser lo más elaborada posible ya que, hoy en día, las computadoras son capaces de adivinar claves mediante fuerza bruta con mucha facilidad. La clave ha de ser compleja y, cuanto más extensa, mejor.

Por tanto, las mayores preocupaciones de este sistema de cifrado son la complejidad de la clave, la transmisión de la misma entre emisor y receptor de forma segura y el hecho de tener que almacenar y proteger gran cantidad de ellas [6]. En la mayoría de los casos, para asegurar que la clave no cae en manos de ningún atacante malicioso, se suele compartir protegida mediante un sistema de cifrado asimétrico.

Ejemplos de este tipo de cifrado son el DES, el RC5 y el algoritmo objetivo de estudio en este trabajo, el AES.

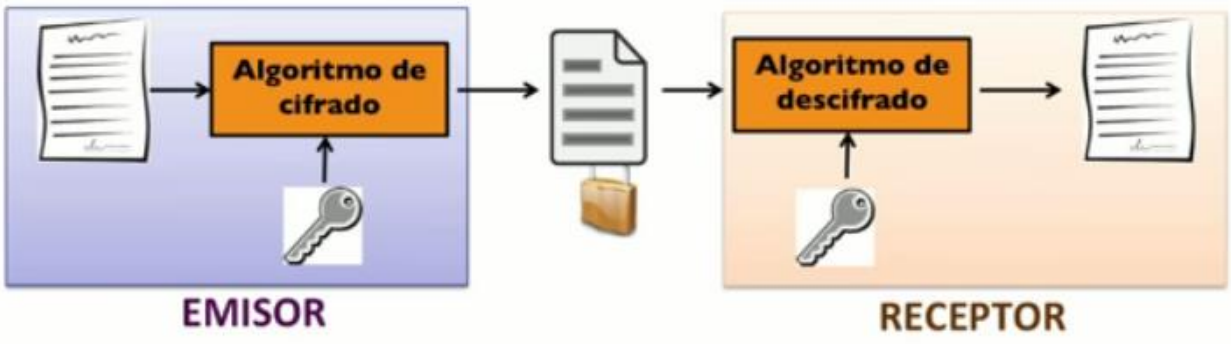

Figura 4. Esquema de comunicación mediante criptología simétrica. Fuente: [6]

Además, dentro de este tipo de criptografía y, atendiendo al tratamiento de la información a cifrar, encontramos otra clasificación: el cifrado en bloque y el cifrado en flujo.

- Cifradores de bloque

En este caso, el texto a cifrar se divide en bloques de un determinado número de bits (n). Según el tamaño del mensaje completo tenemos dos casos:

- El texto a cifrar tiene un número de bits mayor que n. En este caso se utilizan los llamados modos de operación: Electronic Code-Book (ECB), Cipher feedback (CBC), Propagating Cipher-Block Chanining (PCBC), Output FeedBack (OFB). Éstos se encargan de dividir el texto en diferentes bloques de tamaño n, cada uno de los cuales se cifrará de forma independiente. Si el tamaño del texto a encriptar no es múltiplo de n se utilizarán esquemas de relleno
- El texto a cifrar tiene un tamaño menor que n. En este caso solo se utilizarán los esquemas de relleno.

- Cifradores de flujo

En este caso el mensaje se cifra bit a bit. Se suele emplear en situaciones en las que el fragmento de mensaje a cifrar en tiempo real es muy pequeño, del orden de 1 bit.

2.3.1.2 Criptografía asimétrica

La criptografía asimétrica o de clave pública surgió como solución al problema del intercambio de claves requerido en la criptografía simétrica. El principio básico de este tipo de criptografía es el uso de dos claves diferentes para el cifrado y descifrado, una de carácter público y otra privada.

Cuando se va a establecer una comunicación entre emisor y receptor mediante este tipo de cifrado es el receptor el que debe establecer la clave privada. Una vez establecida, genera a partir de ésta una clave pública que envía al emisor sin necesidad de ser protegida, de hecho, existen servidores en los que se almacenan estas claves públicas y a los que cualquiera tiene acceso. Una vez compartida la clave pública con el emisor, éste ha de utilizarla aplicando el algoritmo para cifrar el mensaje y enviarlo al receptor. Dicho mensaje solo puede ser desencriptado con la clave privada por lo que, ni siquiera el emisor del mensaje, sería capaz de leerlo una vez cifrado.

En un sistema de cifrado asimétrico basado en la factorización de números primos [7], el proceso de generación de claves es el siguiente: la clave pública se compone de un número enorme que es el resultado de la multiplicación de dos números primos. Con los ordenadores de hoy en día, es fácil multiplicar dos números grandes, pero, es extremadamente difícil realizar la operación inversa, es decir, a partir de un número compuesto, encontrar los dos factores que han dado lugar a él. Aquí reside la seguridad del proceso ya que, para descifrar el mensaje, es necesario conocer estos factores que solo quedan recogidos en la clave privada y que, por tanto, solo conoce el receptor del mensaje.

Se ha de destacar también el hecho de que el funcionamiento de este sistema es bidireccional, es decir, como hemos dicho, lo que es cifrado por la clave pública solo puede ser descifrado por la privada, obteniendo confidencialidad absoluta, pero, además, estos sistemas tienen la particularidad de que también se puede cifrar con la clave privada y descifrar con la pública. Este último proceso no asegura confidencialidad, pero si autenticidad, es decir que si podemos descifrar el mensaje con la clave pública tenemos la seguridad de que ese mensaje ha sido generado por un emisor concreto, el poseedor de la clave privada. A priori esto puede parecer simple, pero, en realidad, requiere de la aplicación de importantes fundamentos y propiedades matemáticas por lo que, la generación de este par de claves no es en absoluto trivial.

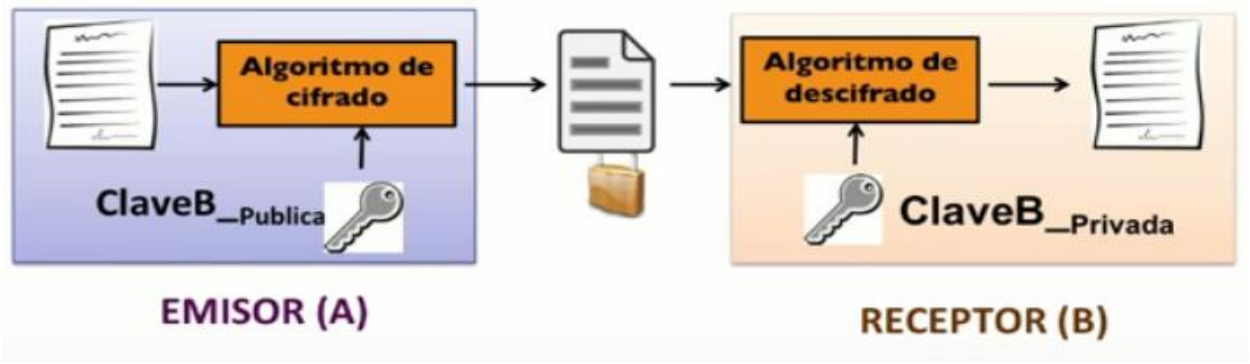

Figura 5. Esquema típico de comunicación mediante criptología asimétrica. Fuente: [6]

A continuación, se recogen las ventajas y los inconvenientes de este método [7].

- Ventajas:
  - No se necesita de canales seguros para enviar la clave pues, la que se comparte, es pública y puede ser conocida por cualquiera
  - No hay desbordamiento en el tratamiento de claves y canales
- Inconvenientes:
  - Son poco eficientes. Así como 128 bits de clave se consideran suficientes para garantizar seguridad en los sistemas simétricos, para este tipo de encriptación la clave ha de ser de, al menos, 1024 bits y, por tanto, se tarda mucho en aplicarlas
  - Hay que proteger la clave privada. Las claves privadas se guardan todas en un archivo denominado keyring que a su vez está protegido con cifrado simétrico. Esto quiere decir que para tener acceso a ella hay que introducir una clave que descifre el archivo keyring y esto implica tener una copia de seguridad del mismo.
  - Hay que transportar la clave privada. Lo que supone que hay que transportar el llavero (keyring) con el peligro que esto supone. El método más seguro para su transporte son las tarjetas inteligentes o Smart Card[1].

2.3.1.3 Criptografía híbrida

La criptografía híbrida es, como su propio nombre indica, una simbiosis entre las otras dos. Consiste en utilizar cifrado asimétrico para compartir la clave del cifrado simétrico. No nos extenderemos más en su desarrollo porque no es objeto de este trabajo.

2.3.2 Criptoanálisis

El criptoanálisis es la ciencia opuesta o, mejor dicho, complementaria a la criptografía. Como se ha descrito en el resto de este capítulo, la criptografía se encarga de crear y analizar criptosistemas seguros mientras que, el criptoanálisis, tiene como objetivo romper estos sistemas, es decir, dejar a relucir sus vulnerabilidades consiguiendo descifrar los criptogramas creados por los mismos [8].

En el análisis para determinar los posibles puntos débiles de un sistema de cifrado se han de asumir las

[1] Tarjeta de plástico y de tamaño de bolsillo que incorpora circuitos integrados que permiten la ejecución de cierta lógica programada

llamadas condiciones del peor caso:

- El criptoanalista o atacante tiene acceso al algoritmo completo de encriptación
- El analista tiene una cantidad notable de texto cifrado
- El criptoanalista conoce el texto plano de entrada a partir del cual se ha obtenido el cifrado

Se debe asumir también el Principio de Kerckhoffs que proclama que la seguridad de un criptosistema ha de residir exclusivamente en el secreto de la clave y no en el algoritmo de cifrado.

A la hora de validar la robustez de un sistema de cifrado, normalmente, se han de asumir todas estas condiciones, sin embargo, existen ataques más específicos en los que no se cumplen todas ellas. Por ejemplo, si el ataque consiste en probar todas las claves posibles de un espacio de claves hasta dar con la correcta, el ataque se denomina de fuerza bruta o exhaustivo; si el atacante tiene acceso únicamente al algoritmo y al criptograma final el ataque se denomina ataque solo al criptograma; si el ataque cumple todas las condiciones enumeradas del peor caso, se le conoce como ataque de texto en claro conocido y si, además, el atacante tiene la posibilidad de cifrar una cantidad indeterminada de texto en claro se dice que es de texto en claro escogido. Este último es el caso habitual cuando se pretende analizar la robustez y seguridad de un criptosistema.

Para que un algoritmo de cifrado sea considerado seguro, debe resistir todos estos ataques y algunos no citados, aunque, también hay que destacar que, en la criptografía, como en cualquier aspecto de la seguridad, el punto más débil siembre son las personas, que pueden caer ante un soborno, un chantaje, una extorsión, etc.

2.4 Revisión histórica del algoritmo

El AES surgió como respuesta a la convocatoria de un concurso público organizado por el NIST (Instituto Nacional de Estándares y Tecnología) de Estados Unidos, para encontrar un sustituto al primer estándar de cifrado para comunicaciones, el Data Encryption Standard, DES, que ya se encontraba rozando la obsolescencia [9]. En 1997 dicha entidad lanzó a la comunidad científica el reto de crear un nuevo algoritmo de cifrado que fuera capaz de proteger información confidencial en el siglo XXI. Este algoritmo recibió el nombre de Advanced Encryption Standard (AES) y se le impusieron los siguientes requisitos:

- Debía ser un algoritmo de cifrado simétrico y capaz de encriptar bloques de, al menos, 128 bits.
- Debía poderse implementar en software y en hardware.
- Debía soportar claves de cifrado de 128, 192 o 256 bits.
- Debía de ser de dominio público, es decir, accesible a cualquiera.

Un año después fueron presentadas las primeras propuestas y, tras dos rondas de selección se anunciaron cinco algoritmos finalistas:

- MARS [IBM. Nevenki Zunic]
- RC6 [RSA Laboratories. Rivest, M. Robshaw, Sidney, Yin]
- RIJNDAEL [Joan Daemen, Vincent Rijmen]
- SERPENT [R. Anderson, E. Biham, L. Knudsen]
- TWOFISH [B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. May, N. Ferguson]

Todo el proceso de selección fue público y, para proceder a la elección del ganador, se animó a la comunidad criptológica a que intentara vulnerar la seguridad de los algoritmos finalistas y, así, descartar aquellos que no fueran lo suficientemente robustos.

En octubre del año 2000 tuvo lugar la conferencia final para elegir al que se convertiría en el Advanced Encryption Standard y, tras una votación y debido a su equilibrio seguridad-velocidad-eficiencia, el Rijndael fue anunciado vencedor. Las principales alegaciones de sus defensores se basaron en que el algoritmo poseía una estructura sencilla, que era eficaz tanto en hardware como en software, que su proceso de cifrado y de permutación de clave eran rápidos y que sus requerimientos de memoria eran muy bajos y, por tanto, se hacía ideal para entornos con espacio restringido.

Sus creadores, dos ingenieros electrónicos belgas, consiguieron imponerse a sus competidores entre los que se contaban grandes multinacionales y equipos de criptólogos de fama mundial con lo que, su algoritmo, se convirtió en el nuevo estándar de comunicaciones de EE. UU. y, en consecuencia, de todo el mundo. Hoy en día, el AES es el algoritmo más popular usado en criptología simétrica.

3 REVISIÓN DEL ALGORITMO

En primer lugar, debemos hacer una distinción entre el Rijndael y el AES. Actualmente son llamados de manera indistinta, pero se ha de decir que, a diferencia del AES que solo cifra bloques de entrada de 128 bits y con una longitud de clave de 128, 192 o 256 bits, el Rijndael tiene una longitud de bloque de entrada y de clave variables. Este admite cualquier longitud que sea múltiplo de 32, con un mínimo de 128 y un máximo de 256 bits [10]. Dicho esto, de ahora en adelante solo hablaremos del Rijndael como AES, es decir, con las especificaciones de este último.

El AES es un algoritmo simétrico de cifrado por bloques lo que significa, que se sirve de la misma clave para el cifrado que para el descifrado. Como se ha dicho antes, soporta bloques de información de 128 bits y una longitud de clave de 128, 192 o 256 bits. Antes de entrar en el análisis del proceso de cifrado se han de introducir algunos conceptos matemáticos de los que se sirve el algoritmo.

3.1 Conceptos matemáticos. Cuerpos finitos. GF(28)

Es menester introducir el concepto de cuerpo finito porque, en este algoritmo, todos los bytes se consideran elementos de uno de estos cuerpos, en concreto, se consideran Campos de Galois.

Los Campos de Galois son un cuerpo definido por un conjunto finito de elementos en el que, el número de elementos debe ser primo o potencia de un número primo:

$$Número\ de\ elementos = (Número\ primo)^{Número\ Natural}$$

El Campo de Galois se representa como GF($primo^m$) y sus elementos lo hacen como polinomios de grado menor que m, es decir:

$$GF(primo^m) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_{m-1} x^{m-1}$$

En los Campos de Galois cada elemento de GF(p^m) es un resto módulo "p", por tanto, si extrapolamos este concepto al AES, lo que interesa es que cada byte sea considerado como un campo GF(28) , así, cada uno de sus elementos serán restos de módulo 2, es decir, 0 y 1, lo que permite que se representen en binario. En consecuencia, si consideramos que un byte está compuesto por los bits b0, b1, b2, b3, b4, b5, b6, y b7 el polinomio que obtendremos será el siguiente [10]:

$$GF(2^8) = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^4 + b_5 x^5 + b_6 x^6 + b_7 x^7$$

Para que sea más visual, si el valor de un byte en hexadecimal es "87", en binario es "10000111" y, por tanto, su polinomio correspondiente será:

$$x^7 + x^2 + x + 1$$

3.1.1 Suma en GF(2^8)

En el caso particular en el que estamos, es decir, que el número primo que define el campo sea 2, la suma de dos polinomios se traduce en una operación Or-Exclusiva de cada uno de los coeficientes dos a dos [11]. Para facilitar su comprensión veamos la tabla de verdad de dicha operación lógica junto con un ejemplo:

Tabla 1. Tabla de verdad Or-Exclusiva

A	B	$A \oplus B$
1	1	0
1	0	1
0	1	1
0	0	0

$$A = 25_{hex} = 00100101_{bin} = x^5 + x^2 + 1$$
$$B = 63_{hex} = 01100011_{bin} = x^6 + x^4 + x^3$$

$$A \oplus B = 01000110 = x^6 + x^2 + x = 46_{hex}$$

3.1.2 Multiplicación en GF(2^8)

La multiplicación en este campo es más compleja puesto que al multiplicar polinomios, lo más probable es que se obtenga un polinomio resultado de grado mayor que 8 y, por tanto, fuera del Campo de Galois en el que nos encontramos. Esto se soluciona reduciendo el polinomio resultado aplicándole el módulo con un polinomio irreducible de grado 8. En el AES en concreto, este polinomio es el siguiente:

$$x^8 + x^4 + x^3 + x + 1$$

Por tanto, los pasos a seguir serían: multiplicar los polinomios de la forma habitual y realizar la operación módulo como se mostrará en el ejemplo, hasta obtener un resultado comprendido en el campo GF(2^8) [11]. A continuación, incluyo un ejemplo para ilustrar esta explicación:

$$A = 25_{hex} = 00100101_{bin} = x^5 + x^2 + 1$$
$$B = 63_{hex} = 01100011_{bin} = x^6 + x^4 + x^3$$
$$A \cdot B = (x^{11} + x^8 + x2^6) + (x^9 + x^6 + x^4) + (x^8 + x^5 + x^3) = x^{11} + x^9 + x^5 + x^4 + x^3$$

Nótese que los elementos que se encontraban repetidos y, por tanto, quedaban multiplicados por 2, como x^8, desaparecen al estar trabajando en GF(2^8).

Una vez obtenido el polinomio resultado pasamos a su reducción para que esté contenido en el campo GF(2^8) de la siguiente manera:

$$Resultado = x^{11} + x^9 + x^5 + x^4 + x^3 = 101000111000$$
$$Irreducible = x^8 + x^4 + x^3 + x + 1 = 100011011$$

$$\mathbf{1}01000111000 \bmod (100011011) = 1000101=$$
$$= x^6 + x^2 + 1 = 45_{hex}$$

Como se puede observar, tras la reducción con el módulo hemos conseguido un polinomio contenido en GF(2^8)

Con estos conceptos será suficiente para entender las operaciones que se llevarán a cabo en el proceso de cifrado del algoritmo.

3.2 Revisión del algoritmo

AES se basa en un diseño conocido como red se sustitución y permutación y, a diferencia de su predecesor, el DES, no utiliza una red de Feistel. El texto de entrada, llamado texto en claro o texto plano y siempre de 128 bits, se organiza, durante las transformaciones, como una matriz de cuatro por cuatro en la que cada posición es ocupada por un byte, a esta matriz se la denomina Estado. Sus dieciséis celdas o bytes van cambiando de valor de acuerdo con los procesos que ejecuta el algoritmo. Para ello, se usan técnicas de permutación, sustitución y operaciones polinómicas en el campo GF(2^8). Todas las transformaciones de cifrado de la matriz Estado se realizan sobre bytes, agrupados en palabras de 32 bits que se colocan de arriba abajo y de izquierda a derecha como se muestra a continuación [12]:

- Texto plano:

~ M E L L A M O P A O L A ~
7E 4D 45 20 4C 4C 41 4D 4F 20 50 41 4F 4C 41 7E (Hex)

- Cuatro palabras 32 bits:

7E4D4520 4C4C414D 4F205041 4F4C417E

Tabla 2. Matriz Estado

7E	4C	4F	4F
4D	4C	20	4C
45	41	50	41
20	4D	41	7E

En el algoritmo AES es la longitud de la clave la que determina el número de rondas de transformación que van a convertir el texto en claro de entrada en el texto final ya cifrado. El número de iteraciones sería, pues, el siguiente: 10 rondas para claves de 128 bits, 12 rondas para claves de 192 bits y 14 rondas para claves de 256 bits. A cada una de estas rondas le corresponde una subclave que desciende de la clave original y que también se configura como un Estado de tamaño igual al del texto plano (4x4); sin importar la longitud de clave con la que estemos trabajando. Los diagramas de flujo del proceso de cifrado según la longitud de clave son los siguientes:

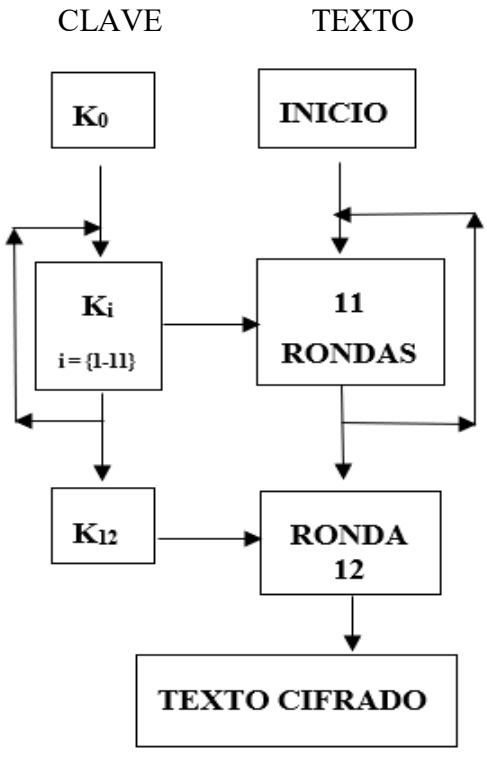

Figura 7. Estructura AES-128

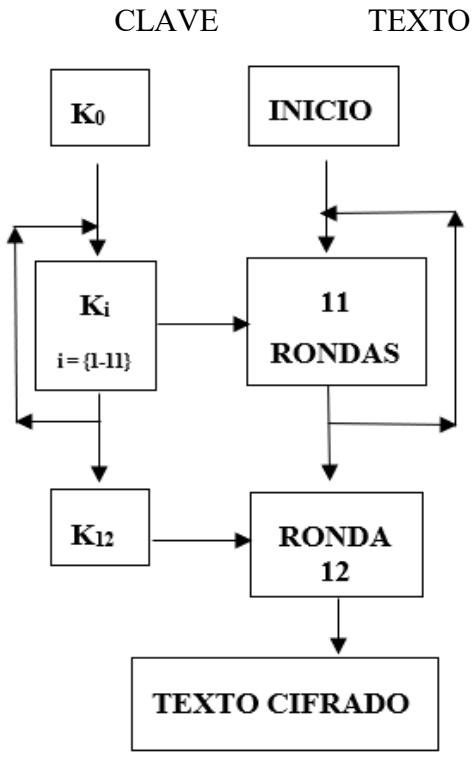

Figura 6. Estructura AES-192

Esta arquitectura sería la misma para una clave de 256 bits salvo que con 14 rondas.

Presentada la arquitectura básica vamos a estudiar en qué consisten las diferentes transformaciones

que sufre el texto en cada ronda y, más adelante, veremos el procedimiento de generación de subclaves de ronda.

Para facilitar la comprensión de cada una de las operaciones que se van a detallar, llamaremos al Estado del texto T y a los bytes que lo componen los denotaremos por t_i (i=1,…,16). Lo mismo haremos con el Estado de la clave al que denominaremos K con bytes k_i.

Tabla 3. Matriz Estado del texto en claro

t_1	t_2	t_3	t_4
t_5	t_6	t_7	t_8
t_9	t_{10}	t_{11}	t_{12}
t_{13}	t_{14}	t_{15}	t_{16}

Tabla 4. Matriz Estado de la clave para AES-128

k_1	k_2	k_3	k_4
k_5	k_6	k_7	k_8
k_9	k_{10}	k_{11}	k_{12}
k_{13}	k_{14}	k_{15}	k_{16}

3.2.1 Transformaciones aplicadas al texto plano durante cada ronda

En primer lugar, hay que recalcar que todas las operaciones a realizar se llevan a cabo sobre la matriz que hemos definido como Estado tanto del texto en claro, como de la subclave de ronda. Además, se ha de saber que todas las funciones que se van a definir a continuación son invertibles lo que facilita enormemente el descifrado. Entendido esto, pasamos a explicar cada una de estas transformaciones y, una vez detalladas, explicaremos cómo y en qué orden se implementan durante cada ronda de cifrado.

1. AddRoundKey

Esta es la operación en la que se combinan los Estados del texto y de la subclave generada para la ronda en cuestión [13]. La subclave es integrada mediante la suma Or-Exclusiva de cada uno de los 16 bytes del Estado del texto con su byte correspondiente de la subclave.

En la siguiente figura se muestra cómo sería el Estado intermedio resultado de esta transformación.

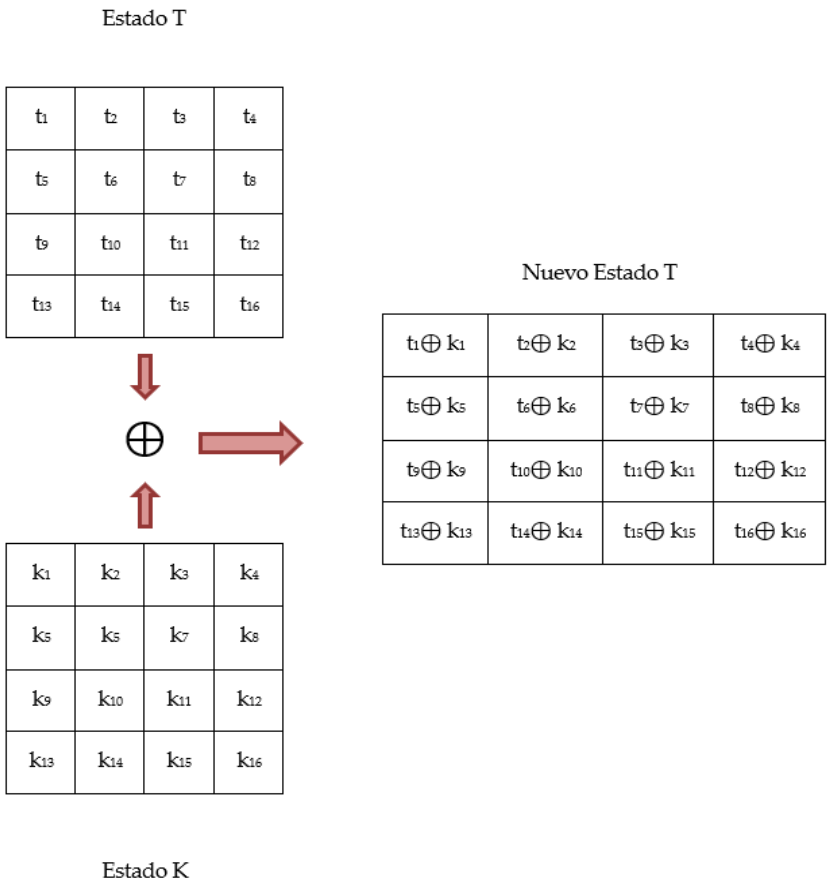

Figura 8. Transformación AddRoundKey

2. SubBytes

Esta operación consiste en la sustitución de cada uno de los bytes del Estado, de forma no lineal e independiente, por su valor correspondiente en una tabla es, por tanto, la función que incorpora la no-linealidad al proceso. En este caso, el byte del Estado es sustituido por uno concreto de la tabla según su valor numérico, no según su posición en el Estado.

Tabla 5. Caja de sustitución para SubBytes. Fuente [28]

A^1	A^0															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Si nos fijamos en la tabla, la fila se selecciona según el valor de A^1 y las columnas según el de A^0 que en un byte son los siguiente:

$$t_i = 11011010 = DA_{hex} = A^1A^0$$

Luego el nuevo valor para $t_i = DA_{hex}$ según la tabla, será: 57. Así sustituímos todos y cada uno de los bytes del Estado.

3. ShiftRows

Esta operación consiste en una permutación de las filas del Estado según un determinado patrón. En concreto, la primera fila se mantiene inalterada, la segunda rota un byte hacia la izquierda, la tercera lo hace dos bytes y la cuarta, tres bytes. La finalidad principal de esta operación es evitar que las columnas se cifren de forma independiente y se pueda dividir el Estado en cuatro bloques independientes porque, si esto ocurriera, el algoritmo sería infinitamente más vulnerable que formando un único bloque. El Estado tras la aplicación de ShiftRows quedaría así:

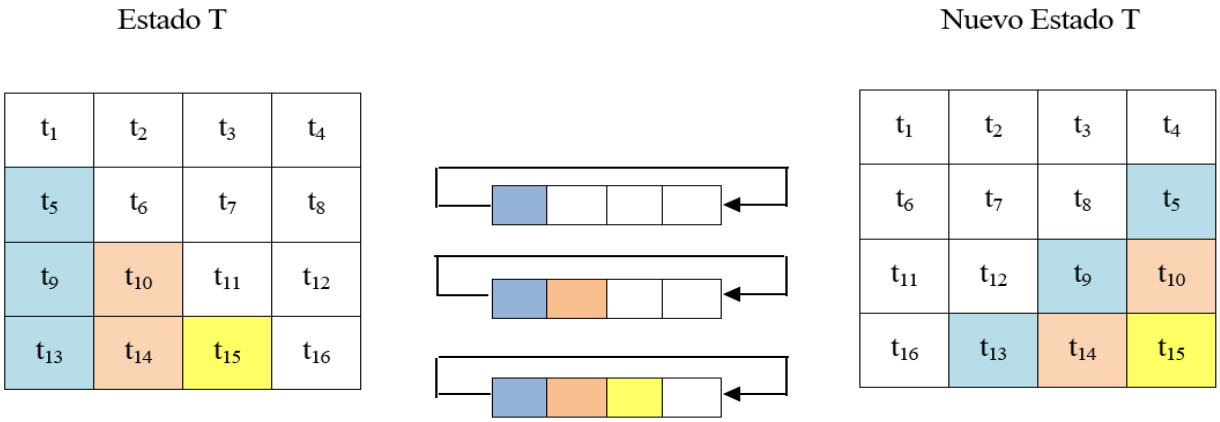

Figura 9. Transformación ShiftRows

1. MixColumns

Esta operación combina los cuatro bytes de cada columna del Estado mediante una transformación lineal. La función MixColumns toma cuatro bytes de entrada y genera otros cuatro a su salida de forma que, cada byte de entrada afecta a los cuatro de salida. Junto con la transformación ShiftRows, MixColumns es la que proporciona difusión en el proceso de cifrado. Es una operación en el campo de GF(2^8) ya que cada columna es considerada un polinomio en dicho campo y se multiplica, módulo $(x^4 + 1)$ con un polinomio fijo [14]:

$$P(x) = 03_{hex}x^3 + 01_{hex}x^2 + 01_{hex}x + 02_{hex}$$

En forma algebraica, la función MixColumns quedaría:

$$C(x) = P(x) \bullet T(x)$$
$$Tnueva(x) = C(x) mod (x^4 + 1)$$

Si la desarrollamos, obtendremos lo siguiente:

$$C(x) = (03_{hex}x^3 + 01_{hex}x^2 + 01_{hex}x + 02_{hex}) \bullet (t_1x^3 + t_2x^2 + t_3x + t_4)$$
$$= c_7x^6 + c_6x^5 + c_5x^4 + c_4x^3 + c_3x^2 + c_2x + c_1$$

Donde los coeficientes se calculan así:

$c_1 = 02_{hex} \bullet t_1$
$c_2 = 01_{hex} \bullet t_1 \oplus 02_{hex} \bullet t_2$
$c_3 = 01_{hex} \bullet t_1 \oplus 01_{hex} \bullet t_2 \oplus 02_{hex} \bullet t_3$
$c_4 = 03_{hex} \bullet t_1 \oplus 01_{hex} \bullet t_2 \oplus 01_{hex} \bullet t_3 \oplus 02_{hex} \bullet t_4$
$c_5 = 03_{hex} \bullet t_2 \oplus 01_{hex} \bullet t_3 \oplus 01_{hex} \bullet t_4$
$c_6 = 03_{hex} \bullet t_3 \oplus 01_{hex} \bullet t_4$
$c_7 = 03_{hex} \bullet t_4$

Una vez hallado el polinomio resultado se reduce módulo $(x^4 + 1)$ quedando:

$$t_{nueva.1} = c_1 \oplus c_4 = 02_{hex} \bullet t_1 \oplus 03_{hex} \bullet t_2 \oplus 01_{hex} \bullet t_3 \oplus 01_{hex} \bullet t_4$$
$$t_{nueva.2} = c_2 \oplus c_6 = 01_{hex} \bullet t_1 \oplus 02_{hex} \bullet t_2 \oplus 03_{hex} \bullet t_3 \oplus 01_{hex} \bullet t_4$$
$$t_{nueva.3} = c_3 \oplus c_7 = 01_{hex} \bullet t_1 \oplus 01_{hex} \bullet t_2 \oplus 02_{hex} \bullet t_3 \oplus 03_{hex} \bullet t_4$$
$$t_{nueva.4} = c_4 = 03_{hex} \bullet t_1 \oplus 01_{hex} \bullet t_2 \oplus 01_{hex} \bullet t_3 \oplus 02_{hex} \bullet t_4$$

Para hacer la operación más visual podemos expresarla de forma matricial:

$$\begin{bmatrix} t_{nueva.1} \\ t_{nueva.2} \\ t_{nueva.3} \\ t_{nueva.4} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix}$$

Con esta operación quedan expuestas todas las funciones que transforman el texto en claro por lo que, ya podemos pasar a ver todas aquellas implementadas durante el proceso de generación de claves.

3.2.2 Algoritmo de generación de subclaves de ronda.

En este apartado vamos a seguir las transformaciones que sufre la clave original para dar lugar a una subclave diferente para cada ronda de cifrado. El algoritmo encargado de esta labor se divide en dos funciones: Expansión de clave y Selección de clave. Antes de comenzar con su explicación se ha de saber que:

- La clave original de cifrado va a derivar en una clave expandida (array lineal)
- El número de bits totales de la clave expandida depende de la longitud del bloque de entrada y del número de rondas (longitud de la clave original) de la siguiente manera:

$$L_{clave_{expandida}} = L_{entrada}(Bits) * (N_{rondas} + 1)$$

Para AES-128→ $L_{clave_{expandida}} = 128 * (10 + 1) = 1408\ bits$

- Las subclaves de cada ronda se extraen de la clave expandida en función de la longitud del texto de entrada, en el caso del AES siempre es 128 bits, es decir, 4 palabras. Por tanto, las primeras 4 palabras serán la clave original, las siguientes 4 la subclave de la primera ronda, las siguientes de la segunda y así, sucesivamente.

3.2.2.1 Expansión de clave

Esta función tiene como objetivo expandir la clave inicial tanto como sea necesario según la longitud de esta [9]. Su salida es una matriz de 4 filas con un número de columnas diferente según sea AES-128, AES-192 o AES-256.

Para calcular el número de columnas o palabras a generar nos servimos de la siguiente fórmula:

$$N_{columnas_{clave}} = \left(N_{rondas_{AES}} + 1\right) * N_{columnas_{texto}}$$

Para AES-128→ $N_{columnas_{clave}} = (10 + 1) * 4 = 44\ columnas\ o\ palabras$

Conocido el número de columnas necesario, pasamos a ver cómo se genera cada una de ellas. En primer lugar, hay que recordar que el número de columnas del Estado de la clave original (N_k) es diferente según sea la longitud de la misma. Sabiendo esto y asignando a la primera columna de la clave expandida el índice cero, todas las columnas que se encuentren en posiciones múltiplo de N_k dentro de la clave expandida se generarán de una forma diferente al resto.

- Generación de columnas múltiplo de N_k

Las columnas en cuestión que, en AES-128 serán la 4, la 8, la 12…etc, se generan a partir de aplicar las siguientes transformaciones a la palabra que se encuentra en la posición anterior a la suya.

1. RotWord

Esta operación consiste en rotar los bytes de la palabra de forma que el primero pase a ser el último y, en consecuencia, los demás suban de posición.

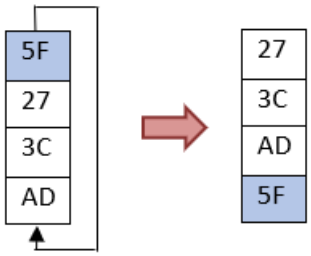

Figura 10. Operación RotWord

1. SubBytes

En esta operación se sustituyen, por sus correspondientes en la caja de sustitución, todos los bytes de esta palabra dando lugar a una nueva.

Figura 11. Operación SubBytes sobre clave

2. Or-Exclusiva (i-3)

Esta operación consiste en realizar la suma Or-Exclusiva entre la palabra en cuestión (ocupando la posición "i") y la que se encuentra tres posiciones más atrás en el Estado.

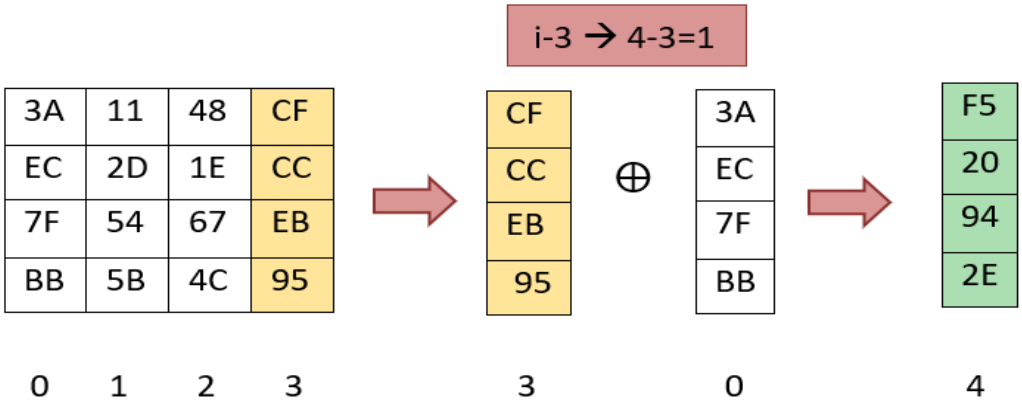

Figura 12. Operación Or-Exclusiva (i-3)

3. Or-Exclusivo Rcon

Esta operación consiste en realizar la suma Or-Exclusiva con un vector denominado Rcon y que es diferente para cada ronda. Dicho vector se genera teniendo en cuenta que Rcon(i)=[Rcon(i), 0_{hex}, 0_{hex}, 0_{hex}] donde cada Rcon(i) es un elemento de GF(2^8) de valor x^{i-1}, con i>0, por ejemplo:

$$Rcon(3) = x^{(3-1)} = x^2 = 00000100 = 04_{hex}$$

El vector Rcon para las primeras 10 rondas sería, por tanto:

Tabla 6. Vectores Rcon

01	02	04	08	10	20	40	80	1B	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
R(1)	R(2)	R(3)	R(4)	R(5)	R(6)	R(7)	R(8)	R(9)	R(10)

Tras todas las operaciones descritas, quedaría generada la palabra en cuestión. Siguiendo los ejemplos, la cuarta palabra de la clave expandida, a la que he denominado 4' quedaría, tras esta última operación:

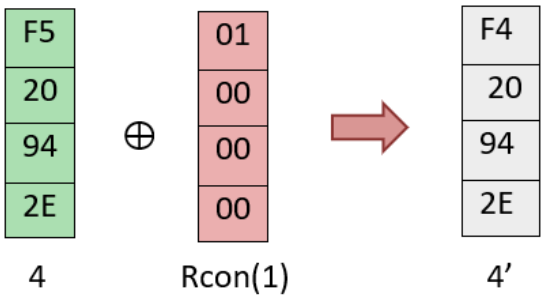

Figura 13. Operación Or-Exclusiva Rcon

- Generación del resto de columnas

El resto de las columnas se generan a partir de una sola operación que se rige por la siguiente fórmula:

$$Columna\ (i \neq N_k) = Columna(i - N_k) \oplus Columna(i - 1)$$

La única excepción tiene lugar en AES-256 en el que las columnas que se encuentran en una posición que dividida por N_k, en este caso 8, da de resto 4, se calculan con la fórmula:

$$Columna(i) = Columna(i - N_k) \oplus SubBytes[Columna(i - 1)]$$
$$Con:\ i\ mod\ N_k = 4$$

3.2.2.2 Selección de clave

La función selección de clave únicamente se dedica a escoger palabras consecutivas del array de la clave expandida y asignarlas a cada subclave de ronda. Estas subclaves siempre han de tener el mismo tamaño y configuración que el texto de entrada, es decir, 4 palabras de 4 bytes distribuidas en una matriz de 4x4.

3.2.3 Proceso de cifrado

Finalmente vamos a ver cómo y en qué orden se implementan las operaciones explicadas a lo largo del proceso de cifrado. En la siguiente figura, se muestra el itinerario seguido por un texto plano de entrada hasta su salida como criptograma según la estructura del AES-128. Para las otras dos longitudes de clave, el proceso es exactamente el mismo con las operaciones en el mismo orden salvo que, la etapa central, se repetirá durante 11 rondas para el AES-192 y durante 13 para el AES-256.

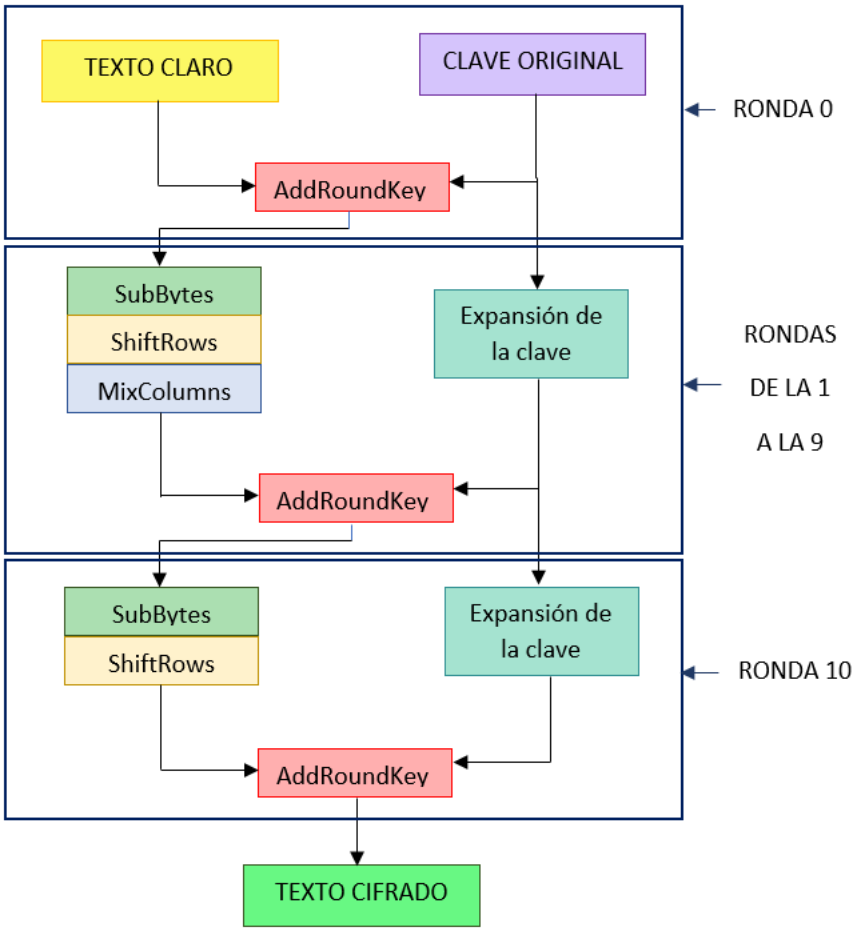

Figura 14. Proceso de cifrado AES-128

4 ATAQUES DE FALLOS SOBRE AES

Este capítulo recoge algunos de los ataques más efectivos conocidos hasta la fecha sobre el AES. Incluye una breve exposición de cada uno de ellos y unas conclusiones.

En primer lugar, se expondrán brevemente los conceptos teóricos necesarios para la comprensión del capítulo.

4.1 Conceptos teóricos

Todos los algoritmos criptográficos son, en mayor o menor grado, vulnerables a determinadas técnicas que buscan cómo obtener información acerca de la clave y, así, descifrar el contenido del mensaje secreto. A continuación, se expondrán algunas de las técnicas más utilizadas hoy en día para atacar los algoritmos y obtener información de relevancia.

4.1.1 Ataque de canal lateral (Side Channel Analysis)

Este ataque tiene su base en obtener información útil a partir del dispositivo físico que implementa el algoritmo en lugar de buscar debilidades en el proceso de cifrado. Esta información útil puede obtenerse a partir de la temporización, del consumo de energía del dispositivo, de fugas electromagnéticas o, incluso, a partir de sonidos. Algunos de estos ataques requieren conocimientos técnicos sobre el funcionamiento del sistema, pero otros como el análisis de diferencias de consumo energético, resultan eficaces como ataques de caja negra [15].

Un ejemplo de este tipo de ataque implementado sobre el AES es el de Adi Shamir y otros dos investigadores en 2005 [16] que consiguieron deducir la clave completa a partir del análisis de los tiempos de caché en tan solo 65 milisegundos. Si es cierto que, para realizar este tipo de ataques, el atacante ha de tener acceso y poder ejecutar programas en el sistema que implementa el algoritmo.

No vamos a entrar en los diferentes tipos de ataques de canal lateral porque su aplicación sobre AES no es objeto de estudio de este trabajo.

4.1.2 Ataque de fallos (Fault Analysis)

Este tipo de ataque es efectivo para los cifradores de bloques. Consiste en introducir un error computacional que modifique el resultado a la salida, es decir, una alteración de los elementos de memoria internos del sistema. Si consideramos una implementación en hardware, este ataque implica un cambio en los valores de los registros del circuito atacado. Este tipo de fallos pueden ser inducidos mediante una proyección láser sobre el chip de silicio [17], aunque, también se puede conseguir sirviéndose de radiación de iones [18], pero, este método implica un mayor presupuesto. A continuación, vamos a ver varios tipos de ataques de fallos hasta explicar, en último lugar, el que es objeto de estudio de este trabajo.

4.1.2.1 Collision Fault Analisys (CFA)

Este tipo de ataque se basa en obtener información a partir de la colisión entre dos textos intermedios idénticos, uno de ellos cifrado sin error y otro con error. Para ello, primero se realiza el cifrado erróneo y, después, se busca un texto plano de entrada que, durante el cifrado coincida con el erróneo. A priori esto puede parecer muy difícil de conseguir, pero la labor se facilita si el error se induce en las primeras fases del proceso evitando, así, el efecto de avalancha entre las encriptaciones de ambos textos de entrada. Para que todo esto suceda, el modelo de fallos implementado asume que un fallo introducido durante una operación aritmética o lógica, como una suma Or-Exclusiva entre dos bytes, produce un resultado igual a algún valor constante (normalmente se considera que cero), valga lo que valga la entrada. Cuando esto se lleva a cabo en las primeras fases del cifrado, el valor intermedio corrompido solo dependerá de la entrada en un bit o byte por lo que, el atacante tan solo tiene que cambiar ese bit o byte hasta que ocurra una colisión con el texto erróneo.

4.1.2.2 Ineffective Fault Analysis (IFA)

Este método, al igual que el anterior, se basa en la obtención de dos salidas idénticas habiendo inducido fallos en una de ellas durante el proceso de encriptado [19]. A diferencia del CFA este tipo de ataque no se centra en encontrar un texto plano de entrada que produzca la misma salida que el erróneo, sino que, usando el mismo texto de entrada para ambos casos, se dé con un tipo de fallo que no tenga efecto ninguno sobre los textos intermedios y, en consecuencia, la salida. Mientras que en el CFA solo se implementaba un fallo, en este ataque se han de probar multitud de ellos hasta dar con el que no tiene efecto ninguno sobre los valores intermedios. De ahí que su nombre sea ataque de fallos inefectivo.

4.1.2.3 Differential Fault Analysis (DFA)

Este ataque es el objetivo de estudio del proyecto pues, se ha demostrado, que algoritmos de encriptación segura como el AES, son vulnerables a él.

El ataque consiste en introducir un fallo en un determinado punto del proceso de manera que se propague hasta la salida y se obtengan dos textos cifrados diferentes, a partir de un mismo texto de entrada, uno de ellos erróneo y, el otro, sin error. De este modo y sabiendo que el circuito ha funcionado correctamente en ambos casos antes del fallo, se pueden comparar ambas salidas y, haciendo una pequeña conjetura sobre la clave, conseguir inferir información de las últimas transformaciones llevadas a cabo. El objetivo principal es conseguir información de la clave utilizada en la última y, puede que también, en la penúltima ronda de forma que se pueda implementar una búsqueda exhaustiva de la clave original.

4.2 Collision/Ineffective Fault Analysis (CFA/IFA) sobre AES

El primer y más popular ataque de este tipo sobre el AES fue propuesto por Blömer et al. [20]. Ellos 9supusieron un modelo de fallos en el que un atacante puede forzar a cualquier bit, elegido de un resultado intermedio, a valer cero. El principio de su ataque es bastante simple. El atacante, en primer lugar, cifra un texto plano compuesto únicamente por ceros hasta obtener el texto cifrado correspondiente. Para continuar, fuerzan a cero el valor de un bit a su salida del primer AddRoundKey y prosiguen con el cifrado, si el texto encriptado final es igual al texto plano de

entrada, se puede asumir que el bit correspondiente a ese en la clave es también cero. Si, por el contrario, ambos textos difieren, se puede asumir que el bit de la clave es uno. Así, llevando a cabo el mismo proceso con todos los bits a la salida del primer AddRoundKey (128 inyecciones de fallo) se puede recuperar la clave completa [19].

Debido a que el modelo de fallos orientado a bits, este ataque se puede considerar como una forma degenerada del CFA o como un IFA.

Este ataque es además extensible a cualquier algoritmo de cifrado en bloque cuya primera operación y, única que involucra al texto en claro y a la clave, sea una suma Or-Exclusiva entre ambos. Nótese que, además, no es necesario conocer el resto de transformaciones que siguen al XOR con la clave. Es, por tanto, aplicable a cualquier algoritmo de cifrado de este tipo.

4.3 Differential Fault Analysis (DFA) sobre el AES

Este apartado consiste en una recopilación de los DFAs más efectivos que se han ejecutado sobre el AES, en su implementación hardware, desde que se convirtió en el estándar de cifrado por excelencia.

Los primeros ataques se expondrán de forma breve hasta llegar al ataque que ha servido de modelo para la realización de este proyecto. Éste se explicará con algo más de detalle a fin de ayudar a la comprensión de la parte práctica del trabajo.

4.3.1 El primer ataque de relevancia sobre AES

En 2003, Gilles Piret y Jean-Jacques Quisquater describieron en un artículo [21] uno de los ataques más efectivos sobre AES. Necesitaba de pocos textos claros de entrada y, además, tenía un modelo de fallos simple, por ello, este ataque se convirtió en una referencia para todo los que le siguieron [22].

El ataque consiste en inducir un fallo en un byte a la entrada de la última operación de MixColumns, es decir, en la penúltima ronda. Lo primero que hay que hacer, es crear una lista, a la que llaman D, que contenga todas las posibles salidas de MixColumns asumiendo una modificación de un byte a su entrada. Esta tarea es un poco tediosa pero solo hay que llevarla a cabo una vez y puede servir para ataques futuros. A continuación, se obtienen los dos textos cifrados con (C*) y sin error (C). A continuación, se plantean las siguientes ecuaciones para los 4 bytes afectados por el fallo:

$$\triangle_i = SB^{-1}(C_i \oplus K_i) \oplus SB^{-1}(C^*{}_i \oplus K_i)$$
$$\triangle_j = SB^{-1}(C_j \oplus K_j) \oplus SB^{-1}(C^*{}_j \oplus K_j)$$
$$\triangle_k = SB^{-1}(C_k \oplus K_k) \oplus SB^{-1}(C^*{}_k \oplus K_k)$$
$$\triangle_l = SB^{-1}(C_l \oplus K_l) \oplus SB^{-1}(C^*{}_l \oplus K_l)$$

Siendo SB^{-1}la función inversa de SubBytes. Los resultados se van comparando con los elementos de la lista D y, los que encuentran una coincidencia son almacenados en otra lista, L. Se repite la operación con otro texto plano de entrada tantas veces como sea necesario para reducir la lista de

candidatos. Más adelante, implementaron mejoras como la de inducir el fallo a la entrada de MixColumns de la ronda anterior. De esta forma consiguieron que con una sola pareja de textos con y sin error, la lista de candidatos se redujese a 2^{40} posibilidades asumibles para una búsqueda exhaustiva o por fuerza bruta.

Este ataque también es extensible al AES-192 y al AES-256.

4.3.2 Atacando diagonales

En 2009, Saha et al. publicaron un artículo [23] que mejoraba notablemente la efectividad del propuesto por Piret y Quisquater.

Este ataque surge a partir de la observación de que, un fallo que afecte a una de las diagonales del Estado a su entrada a la antepenúltima vuelta se expande a la totalidad del Estado a su salida de la penúltima ronda. Además, se observó que los 4 bytes de cada columna responsables de la expansión del fallo tras el último MixColumns, tienen cierta relación que puede utilizarse para obtener cierta información acerca de la clave. Este ataque es capaz de discernir la clave con total exactitud con el uso de, tan solo, cuatro parejas de textos planos cifrados con y sin error.

No nos vamos a detener más en este ataque porque es la inspiración del que ha servido de modelo para este trabajo y, para evitar redundancias, pasaremos directamente a la exposición del mismo.

4.3.3 Ataque mejorado sobre AES-128

Este ataque presentado por Michael Tunstall, Debdeep Mukhopadhyay y Subidh Ali en su artículo [24], es el ataque de referencia para la realización de este proyecto.

Este ataque está destinado a una implementación iterativa del algoritmo y, asume al igual que muchos otros, que el fallo consistirá en modificar el valor de un byte a un valor aleatorio.

El ataque en concreto consiste en modificar el valor de un byte del Estado a su entrada a la antepenúltima ronda. Como se ve en la Figura 10 tras la primera operación de MixColumns el fallo se extiende a la primera columna, tras la operación ShiftRows de la siguiente ronda los bytes afectados se distribuyen hasta aparecer en todas las columnas y, por último, el siguiente MixColumns expande el error a los 12 bytes restantes.

La Figura 10 muestra en concreto la propagación de un fallo inyectado en un byte del Estado a su entrada de la octava ronda sobre el Estado de las diferencias, es decir, sobre el resultado de la operación Or-Exclusiva entre el texto libre de fallos (C) y el erróneo (C*).

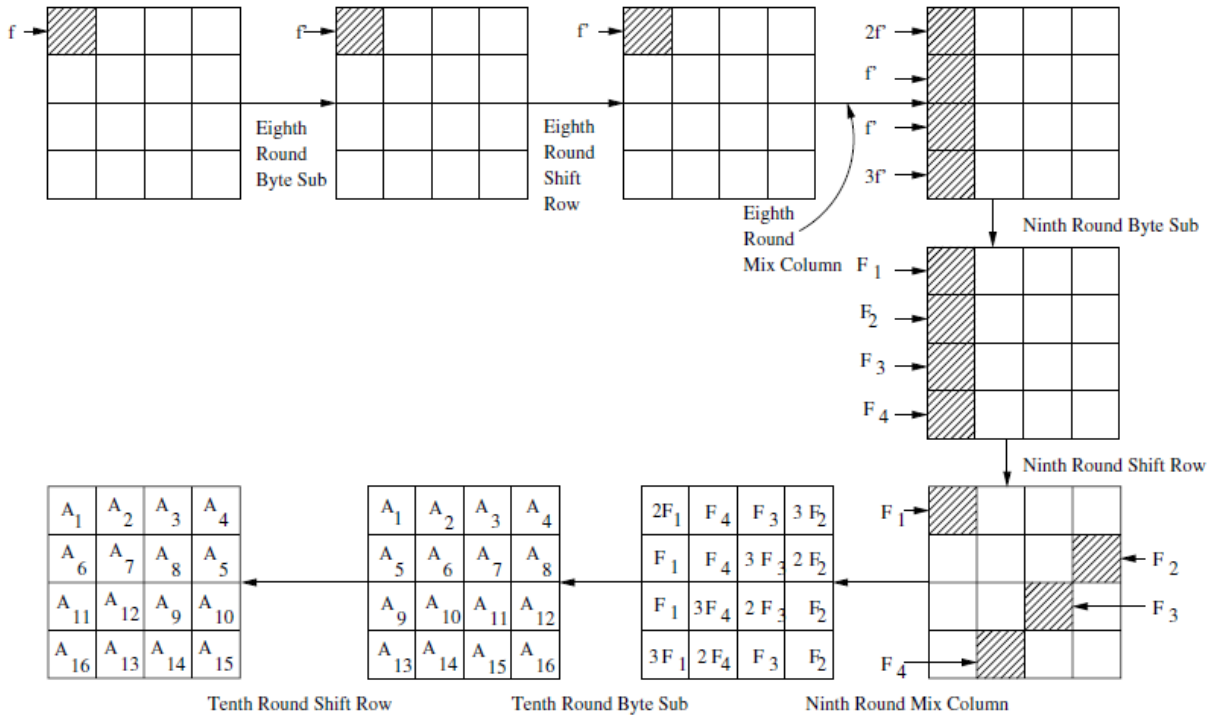

Figura 15. Propagación de fallo en un byte a la entrada de la octava ronda

4.3.3.1 Primer paso

Con este tipo de fallo inducido al inicio de la octava ronda, si consideramos el Estado de las diferencias tras la transformación ShiftRows de la novena ronda, podemos obtener el siguiente sistema de ecuaciones que explota las interrelaciones entre los bytes afectados:

$$2\delta_1 = SB^{-1}(x_1 \oplus k_1) \oplus SB^{-1}(x'_1 \oplus k_1)$$
$$\delta_1 = SB^{-1}(x_{14} \oplus k_{14}) \oplus SB^{-1}(x'_{14} \oplus k_{14})$$
$$\delta_1 = SB^{-1}(x_{11} \oplus k_{11}) \oplus SB^{-1}(x'_{11} \oplus k_{11})$$
$$3\delta_1 = SB^{-1}(x_8 \oplus k_8) \oplus SB^{-1}(x'_8 \oplus k_8)$$

Donde SB^{-1} es la operación inversa de SubBytes, δ es el diferencial de error, x_i es el byte correspondiente a la posición i del Estado del criptograma sin error, x'_i es el del criptograma erróneo y k_i es el byte correspondiente de la subclave de la última ronda. Los valores de δ y de los bytes de la subclave son desconocidos, pero ∈ {0, …, 255}.

Este sistema nos permite reducir las posibilidades para los 32 bits de la subclave representados de la siguiente manera: el atacante da valores a δ y obtiene 0, 2 ó 4 valores de cada k. Si para una δ no se satisface alguna de las cuatro ecuaciones, es decir, no se obtienen valores para alguno de los bytes de la clave, esa δ y los valores de las k asociados se descartan. Además, también se descartarán las soluciones para δ igual a cero ya que, eso significaría que el texto sin fallos y el erróneo coinciden y, en consecuencia, que no se ha inyectado fallo esperado.

Una vez resuelto el sistema siguiendo estas consideraciones el atacante puede esperar que las posibles combinaciones de bytes para el valor del cuarteto k_1, k_8, k_{11}, k_{14}, queden reducidas a 2^8.

Esta misma técnica puede utilizarse para obtener información acerca de los demás bytes de la clave. En concreto, los sistemas de ecuaciones que definen sus interrelaciones son los siguientes:

Para los bytes 2, 5, 12 y 15:

$$3\delta_2 = SB^{-1}(x_5 \oplus k_5) \oplus SB^{-1}(x'_5 \oplus k_5)$$
$$2\delta_2 = SB^{-1}(x_2 \oplus k_2) \oplus SB^{-1}(x'_2 \oplus k_2)$$
$$\delta_2 = SB^{-1}(x_{15} \oplus k_{15}) \oplus SB^{-1}(x'_{15} \oplus k_{15})$$
$$\delta_2 = SB^{-1}(x_{12} \oplus k_{12}) \oplus SB^{-1}(x'_{12} \oplus k_{12})$$

Para los bytes 3, 6, 9 y 16:

$$\delta_3 = SB^{-1}(x_9 \oplus k_9) \oplus SB^{-1}(x'_9 \oplus k_9)$$
$$3\delta_3 = SB^{-1}(x_6 \oplus k_6) \oplus SB^{-1}(x'_6 \oplus k_6)$$
$$2\delta_3 = SB^{-1}(x_3 \oplus k_3) \oplus SB^{-1}(x'_3 \oplus k_3)$$
$$\delta_3 = SB^{-1}(x_{16} \oplus k_{16}) \oplus SB^{-1}(x'_{16} \oplus k_{16})$$

Y, por último, los bytes 4, 7, 10 y 13:

$$\delta_4 = SB^{-1}(x_{13} \oplus k_{13}) \oplus SB^{-1}(x'_{13} \oplus k_{13})$$
$$\delta_4 = SB^{-1}(x_{10} \oplus k_{10}) \oplus SB^{-1}(x'_{10} \oplus k_{10})$$
$$3\delta_4 = SB^{-1}(x_7 \oplus k_7) \oplus SB^{-1}(x'_7 \oplus k_7)$$
$$2\delta_4 = SB^{-1}(x_4 \oplus k_4) \oplus SB^{-1}(x'_4 \oplus k_4)$$

Como se puede apreciar, todas las ecuaciones tienen la misma estructura y, por tanto, la naturaleza de sus soluciones también será la misma. Si se repite el procedimiento aplicado al primer sistema con todos los demás, cada uno aportará 2^8 posibilidades para los bytes implicados en el mismo y, por tanto, se tendrán un total de 2^{32} hipótesis para la clave secreta de la ronda 10.

4.3.3.2 Segundo paso

Implementando este segundo paso se pretende reducir el número de hipótesis para la clave. Para ello, sabiendo que el proceso de generación de claves es invertible, se explotan las relaciones entre los bytes de la clave de la décima ronda y los de la clave de la ronda anterior.

En primer lugar, se establece el proceso inverso de generación de clave con el fin de tener la clave de la novena ronda expresada en función de la de la décima. En este sentido se obtiene el siguiente Estado de la clave de la novena ronda:

Tabla 7. Estado de la subclave de la novena ronda expresada en términos de la de la décima ronda

$k_1 \oplus SB(k_{14} \oplus k_{10}) \oplus R_{10}$	$k_5 \oplus k_1$	$k_9 \oplus k_5$	$k_{13} \oplus k_9$
$k_2 \oplus SB(k_{15} \oplus k_{11})$	$k_6 \oplus k_2$	$k_{10} \oplus k_6$	$k_{14} \oplus k_{10}$
$k_3 \oplus SB(k_{16} \oplus k_{12})$	$k_7 \oplus k_3$	$k_{11} \oplus k_7$	$k_{15} \oplus k_{11}$
$k_4 \oplus SB(k_{13} \oplus k_9)$	$k_8 \oplus k_4$	$k_{12} \oplus k_8$	$k_{16} \oplus k_{12}$

La demostración de estas relaciones y su desarrollo se llevará a cabo en el siguiente capítulo, por el momento basta con saber que son esas y que R_{10} es una constante derivada del vector Rcon.

Observando la Figura 10 de nuevo, puede verse cómo el diferencial de fallo a la salida de la octava ronda se denota por 2*f'*, *f'*, *f'* y 3*f'*, siendo *f'*, al igual que δ, cualquier valor ∈ {0, …, 255}. Pues, aplicando la operación inversa a MixColumns y las relaciones establecidas para la clave, obtenemos otro set de cuatro ecuaciones de las cuales sólo voy a escribir la segunda a modo de ejemplo:

$$
\begin{aligned}
f' = \; & SB^{-1}\{\, 9\,[SB^{-1}(x_{13} \oplus k_{13}) \;\oplus\; (k_{13} \;\oplus k_9)] \;\oplus\; 14\,[\,SB^{-1}(x_{10} \oplus k_{10}) \;\oplus (k_{10} \oplus k_6)] \\
& \oplus\; 11[\,SB^{-1}(x_7 \oplus k_7) \;\oplus (\,k_{15} \oplus k_{11})] \oplus\; 13\,[\,SB^{-1}(x_4 \oplus k_4) \;\oplus(\,k_{15} \oplus k_{11})]\} \\
& \oplus\; SB^{-1}\{\, 9\,[SB^{-1}(x'_{13} \oplus k_{13}) \;\oplus\; (k_{13} \;\oplus k_9)] \\
& \oplus\; 14\,[\,SB^{-1}(x'_{10} \oplus k_{10}) \;\oplus (k_{10} \oplus k_6)] \\
& \oplus\; 11[\,SB^{-1}(x'_7 \oplus k_7) \;\oplus(\,k_{15} \oplus k_{11})] \\
& \oplus\; 13\,[\,SB^{-1}(x'_4 \oplus k_4) \;\oplus\; (k_{16} \oplus k_{12})]\}
\end{aligned}
$$

Dicha ecuación se puede reescribir como: $f' = A \oplus B$ donde A sería el primer miembro que se encuentra entre corchetes y B el segundo o, mejor dicho, donde A es el que recoge las operaciones que implican al texto sin error y B el que comprende a las del texto fallido.

Si consideramos A y B cualquier valor comprendido en GF(2^8), la probabilidad de que la ecuación sea válida es de $\frac{1}{2^8}$ y, por tanto, de que lo sean las cuatro ecuaciones $(\frac{1}{2^8})^4 = \frac{1}{2^{32}}$.

Luego, cualquiera de las hipótesis hechas sobre la clave en el primer paso, será cierta con una probabilidad de $2^8 * \frac{1}{2^{32}} = \frac{1}{2^{24}}$. Recordando que se tenían 2^{32} hipótesis tras el primer paso, uno puede esperar que, tras este paso, queden reducidas a 2^8 posibilidades en total.

4.4 Conclusiones

Hoy en día, los ataques de fallos en todas sus variantes han probado ser los más efectivos contra el AES en su implementación hardware y, entre ellos, destaca el DFA por ser el más reproducido.

En este trabajo se ha elegido como referencia el ataque expuesto en ultimo lugar porque es capaz de reducir mucho el número de claves candidatas con tan solo una inyección de fallo.

5 TRABAJO REALIZADO

En el capítulo anterior se han visto algunos de los ataques de fallos que, inyectados en determinadas rondas, permiten inferir información acerca de la clave.

Este capítulo recoge la simulación del último de los ataques expuestos sirviéndose de la herramienta ISIM de Xilinx, en la que el código del ataque será implementado en VHDL. Es crucial para la realización de un ataque sobre hardware el haber comprobado primero la viabilidad del mismo mediante simulación, para ello, se inyectará un error y se procesarán los textos cifrados con y sin error hasta conseguir inferir información de la clave.

5.1 Pasos previos a la implementación del ataque

Antes de pasar a explicar cómo se ha implementado el ataque en sí, se van a describir los pasos previos que han sido necesarios para llevarlo a cabo.

En primer lugar, ha de conocerse el simulador ISE de Xilinx y las herramientas que ofrece para la realización de este trabajo. Xilinx-ISE es una herramienta de diseño de circuitos que opera con lenguajes HDL, en concreto, con VHDL y Verilog. Permite realizar diseños hardware sintetizables y, además, es capaz de generar el esquemático de los mismos tras compilar el código. Permite estimular las entradas de los diseños y simularlos gracias a la herramienta ISIM. Sin embargo, la razón principal de su uso para este proyecto es que ofrece la posibilidad de forzar señales a los valores que uno desee, ya sean o no sean registros, y observar la propagación del cambio. Este forzado de señales servirá para simular la inyección de un error en el proceso de cifrado en un momento determinado ya que, con el registro de contador de ronda, se puede determinar el punto exacto en el que inyectar el error.

5.1.1 Implementación del AES en VHDL

El primer paso para la reproducción de este ataque es encontrar una implementación del AES en VHDL que se ajuste a los requerimientos del trabajo [25]. En este caso, el código elegido implementa el AES y viene acompañado por un fichero de estímulos para poder realizar las primeras pruebas y alcanzar la comprensión del proceso de cifrado implementado. El código cuenta, por tanto, con:

- Módulo principal. Lleva a cabo el proceso completo de cifrado
- Módulo de generación de claves. Implementa las operaciones necesarias para la generación de las subclaves de ronda
- Módulo de pruebas (Test-bench). Fichero que contiene los estímulos de las entradas para poder simular el código.

En la Figura se muestra la simulación de dicho código. En la imagen se puede apreciar cómo el texto plano de entrada se va transformando según va pasando cada ronda hasta obtener un criptograma final completamente diferente.

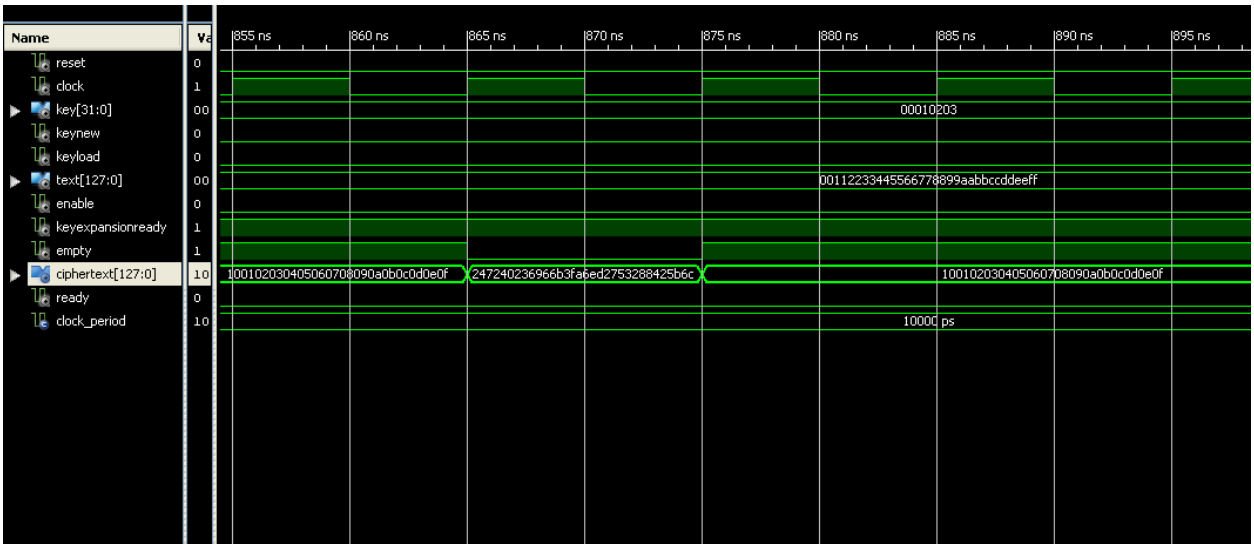

Figura 16. Simulación del código que implementa el AES

5.1.2 Forzado de señales

Para poder llevar a cabo el forzado de señales es necesario que el código se haya compilado y simulado correctamente con el fichero es estímulos mencionado. Una vez en la ventana de simulación, se puede forzar el contenido de cualquier señal durante el tiempo que se considere oportuno haciendo clic derecho sobre su nombre y seleccionando la opción "Force Constant". Al seleccionarla, se despliega una ventana con las diferentes opciones (nombre, base, valor, tiempo)

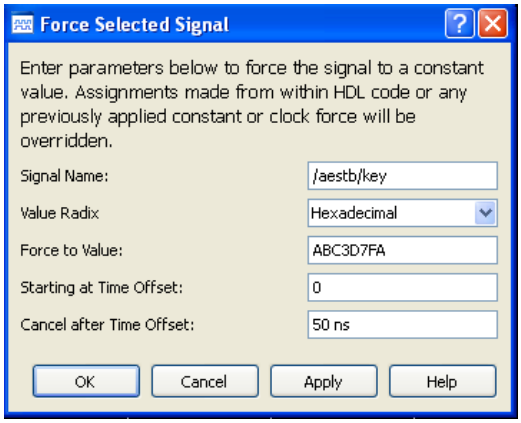

Figura 17. Panel de forzado de señales en ISIM

A modo de ejemplo se va a forzar la salida del algoritmo, es decir, el criptograma final del valor 0x100102030405060708090a0b0c0d0e0f al valor 0x00112233445566778899aabbccddeeff para que coincida con el texto de entrada, En la Figura 18 puede verse dicho cambio de valor durante el tiempo que se le ha especificado (4 ns).

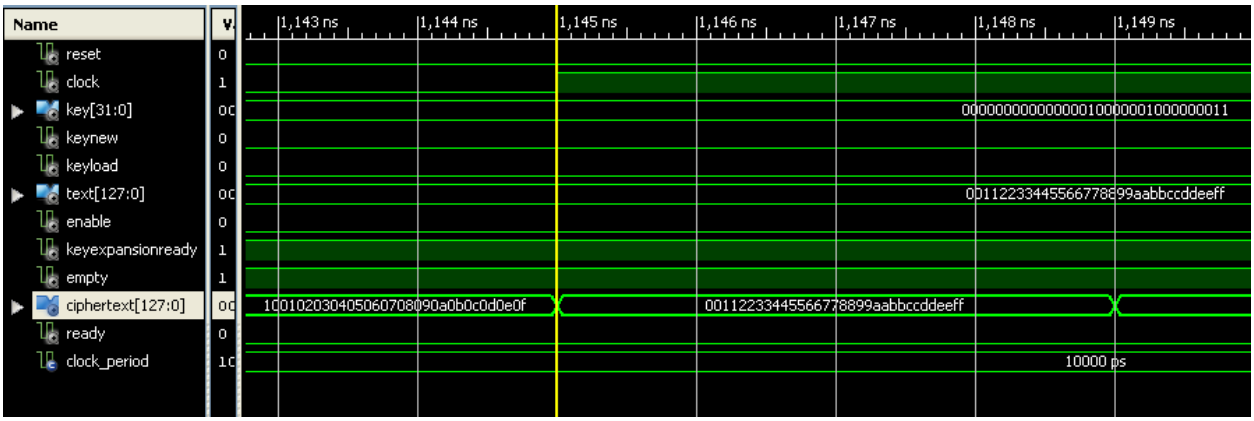

Figura 18. Señal forzada en ISIM

5.2 Reproducción del ataque de fallos en simulación

Durante este apartado se van a detallar los pasos que se han seguido para la reproducción del ataque, con el código implementado en VHDL, y la consecuente recuperación de la clave.

El fallo se va a reproducir sobre una recreación del AES-128, es decir, con una clave de 128 bits. Para ello, contamos con un código que implementa el sistema de cifrado y al que se le va a incorporar un módulo de recuperación de clave.

Como se dijo en el capítulo anterior, el ataque que va a servir referencia es el que se expuso en último lugar, el realizado por Michael Tunstall, Debdeep Mukhopadhyay y Subidh Ali. Ese ataque consta de dos pasos o fases dedicadas a reducir las 2^{128} posibilidades de clave que se tienen inicialmente en el AES-128, llevando a cabo una sola inyección de fallo, es decir, utilizando solo una pareja de textos cifrados con y sin error. Una vez reducidas, se realiza una búsqueda exhaustiva entre las candidatas restantes hasta dar con la clave.

5.2.1 Implementación del ataque en VHDL

El objetivo de este trabajo es implementar la primera fase del ataque escogido que, si se lleva a cabo correctamente, reduce a 2^{32} las posibilidades de clave, un número que, con las computadoras de hoy en día, es asumible para su recuperación por fuerza bruta.

Si se recuerda, el ataque proponía unos sistemas de ecuaciones que explotaban la relación entre los bytes del texto cifrado sin error, el erróneo y la clave de la décima ronda. El código diseñado para su reproducción no hace otra cosa que implementar dichos sistemas en VHDL y resolverlos para obtener los posibles candidatos a clave.

El código tiene como entradas al texto encriptado correctamente y al texto al que se le ha inferido un error de un byte a la entrada de la octava ronda pues ambos intervienen en las ecuaciones que se van a plantear.

Todas las ecuaciones que componen los sistemas tienen dos incógnitas, el diferencial de error y un determinado byte de la clave de la décima ronda. En consecuencia, para su resolución, es necesario dar valores a una de ellas y despejar para obtener la otra. En el caso particular del código que se ha implementado en este proyecto, damos valores a los bytes de la clave para obtener el diferencial de error.

Como se explicó en la exposición del ataque, para cada sistema, el diferencial de error o δ, puede tomar valores desde 1 a 255, ya que tiene un tamaño de 8 bits y un valor de 0 implicaría que no se introducido fallo, sin embargo, sólo debemos quedarnos con aquellos valores que tengan asociados un valor para cada uno de los cuatro bytes de la clave implicados. El proceso que se ha seguido es el siguiente:

Cada uno de los conjuntos de ecuaciones es de este tipo:

$$2\delta_1 = SB^{-1}(x_1 \oplus k_1) \ \oplus \ SB^{-1}(x'_1 \oplus k_1)$$
$$\delta_1 = SB^{-1}(x_{14} \oplus k_{14}) \ \oplus \ SB^{-1}(x'_{14} \oplus k_{14})$$
$$\delta_1 = SB^{-1}(x_{11} \oplus k_{11}) \ \oplus \ SB^{-1}(x'_{11} \oplus k_{11})$$
$$3\delta_1 = SB^{-1}(x_8 \oplus k_8) \ \oplus \ SB^{-1}(x'_8 \oplus k_8)$$

En el código, se resuelve la primera ecuación de forma independiente dando valores entre 0 y 255 a k_1 y obteniendo las δ_1 correspondientes. Se almacenan los valores de k_1 que han dado lugar a deltas diferentes de cero y, en otra matriz de memoria se guardan las δ, también distintas de cero, que se han obtenido.

A continuación, se pasa a la segunda ecuación para la cual sólo se almacenan los valores de k_{14} que han dado lugar a deltas distintas de cero y que, además, hayan sido guardadas como resultado de la primera ecuación porque, recordemos, que según se establece en el ataque, solo son válidos aquellos valores de delta que tienen asociados los cuatro valores de los bytes de la clave.

Con las otras dos ecuaciones se procede de la misma forma consiguiendo reducir los valores de esos 4 bytes de la clave a tan solo 2^8 posibilidades.

Nótese que en varias ecuaciones delta viene multiplicada por una constante lo que podría ser un problema porque estamos trabajando en un campo GF(2^8) y una multiplicación podría implicar salirnos del mismo. Para evitar esto, tras realizar la multiplicación el resultado se reduce con una operación módulo el polinomio:

$$x^8 + x^4 + x^3 + x + 1$$

Se prosigue de la misma manera con los 12 bytes restantes de la clave, es decir, se implementan los otros tres conjuntos de ecuaciones que se definieron en el capítulo anterior.

La resolución de cada conjunto de ecuaciones concluye con las mismas 2^8 posibilidades de clave que se comentaron antes por lo que, tras buscar las soluciones de los cuatro conjuntos, nos quedan:

$$4 * 2^8 = 2^{32} \ posibilidades \ de \ clave$$

Sin embargo, estas posibilidades se tienen en función de la clave de la última ronda, es decir, la décima por lo que, antes de comenzar el ataque por fuerza bruta, se han de deshacer las operaciones de generación de clave hasta llegar a la original.

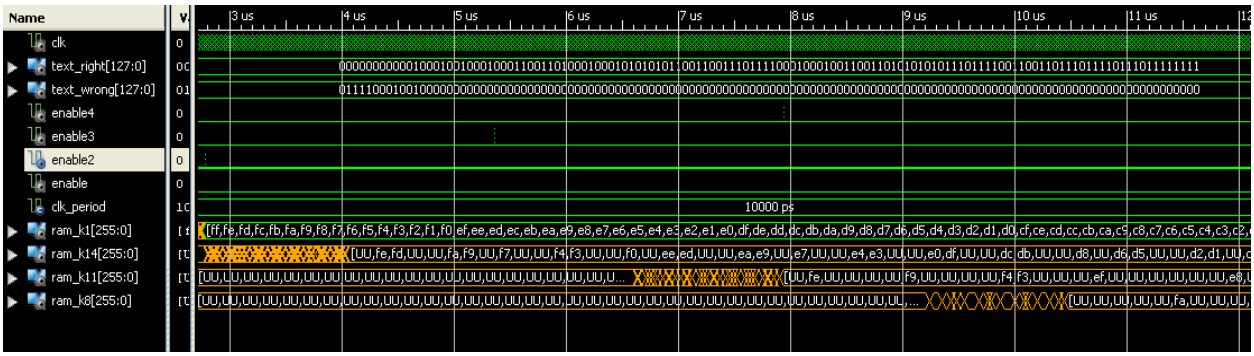

Figura 19. Candidatas a clave en la décima ronda obtenidas del primer conjunto de ecuaciones

En la Figura 19 se pueden ver los resultados del primer conjunto de ecuaciones resueltas con la pareja de textos con (text_wrong) y sin error (text_right). Los valores que aparecen indeterminados son aquellos que no cumplen las condiciones expuestas anteriormente, es decir, son los correspondientes a deltas que no satisfacen las cuatro ecuaciones. Por tanto, los valores recogidos en esas matrices de memoria darán lugar, como se ha dicho, a 2^8 posibles combinaciones.

5.2.2 Recuperación de la clave

Para realizar la búsqueda exhaustiva de la clave entre esas 2^{32} posibilidades obtenidas tras el ataque, hemos de probar a cifrar el texto con todas las combinaciones posibles hasta dar con la que produzca el mismo criptograma que el cifrado con la clave original. Tras la implementación del ataque, tenemos esos candidatos a clave, pero expresados en la subclave de la última ronda por lo que, para hacer la búsqueda por fuerza bruta, hemos de deshacer las transformaciones sufridas por la clave original en el proceso de generación de claves de manera inversa. Al principio, esto puede parecer extremadamente tedioso, pero, sin embargo, se pueden definir unas ecuaciones que desharían los cambios sin demasiada complicación.

Si consideramos el siguiente ejemplo de matriz de clave expandida:

$$\begin{pmatrix} K_1 & K_2 & K_3 & K_4 & \dots\dots & k_1^{10} & k_2^{10} & k_3^{10} & k_4^{10} \\ K_5 & K_6 & K_7 & K_8 & \dots\dots & k_5^{10} & k_6^{10} & k_7^{10} & k_8^{10} \\ K_9 & K_{10} & K_{11} & K_{12} & \dots\dots & k_9^{10} & k_{10}^{10} & k_{11}^{10} & k_{12}^{10} \\ K_{13} & K_{14} & K_{15} & K_{16} & \dots\dots & k_{13}^{10} & k_{14}^{10} & k_{15}^{10} & k_{16}^{10} \end{pmatrix}$$

Donde K_i es un byte de la clave original y k_i^{10} es un byte de la clave de la última ronda.

En el apartado de expansión de clave, vimos que las palabras se generaban de forma diferente según su posición en la matriz expandida, en concreto, se generaban de forma diferente todas aquellas columnas o palabras que ocupaban una posición múltiplo de 4 (empezando a contar desde la posición cero) en la matriz expandida. Lo mismo ocurre a la hora de realizar la operación inversa.

Los cambios repiten siempre esta estructura:

- Matriz clave de la novena ronda expresada en términos de la décima:

$$\begin{pmatrix} k_1^{10} \oplus \mathrm{SB}(k_{14}^{10} \oplus k_{10}^{10}) \oplus 36_{hex} & k_5^{10} \oplus k_1^{10} & k_9^{10} \oplus k_5^{10} & k_{13}^{10} \oplus k_9^{10} \\ k_2^{10} \oplus \mathrm{SB}(k_{15}^{10} \oplus k_{11}^{10}) & k_6^{10} \oplus k_2^{10} & k_{10}^{10} \oplus k_6^{10} & k_{14}^{10} \oplus k_{10}^{10} \\ k_3^{10} \oplus \mathrm{SB}(k_{16}^{10} \oplus k_{12}^{10}) & k_7^{10} \oplus k_3^{10} & k_{11}^{10} \oplus k_7^{10} & k_{15}^{10} \oplus k_{11}^{10} \\ k_4^{10} \oplus \mathrm{SB}(k_{13}^{10} \oplus k_9^{10}) & k_8^{10} \oplus k_4^{10} & k_{12}^{10} \oplus k_8^{10} & k_{16}^{10} \oplus k_{12}^{10} \end{pmatrix}$$

- Matriz clave de la octava ronda expresada en función de la novena:

$$\begin{pmatrix} k_1^{9} \oplus \mathrm{SB}(k_{14}^{9} \oplus k_{10}^{9}) \oplus 1B_{hex} & k_5^{9} \oplus k_1^{9} & k_9^{9} \oplus k_5^{9} & k_{13}^{9} \oplus k_9^{9} \\ k_2^{9} \oplus \mathrm{SB}(k_{15}^{9} \oplus k_{11}^{9}) & k_6^{9} \oplus k_2^{9} & k_{10}^{9} \oplus k_6^{9} & k_{14}^{9} \oplus k_{10}^{9} \\ k_3^{9} \oplus \mathrm{SB}(k_{16}^{9} \oplus k_{12}^{9}) & k_7^{9} \oplus k_3^{9} & k_{11}^{9} \oplus k_7^{9} & k_{15}^{9} \oplus k_{11}^{9} \\ k_4^{9} \oplus \mathrm{SB}(k_{13}^{9} \oplus k_9^{9}) & k_8^{9} \oplus k_4^{9} & k_{12}^{9} \oplus k_8^{9} & k_{16}^{9} \oplus k_{12}^{9} \end{pmatrix}$$

Y así sucesivamente con todas las subclaves de ronda hasta llegar a la clave original. Nótese que la primera columna de cada matriz (la que se encuentra en una posición múltiplo de cuatro en la matriz de clave expandida) lleva sumando a la columna correspondiente a la ronda de la tabla Rcon. Además, lleva incorporada la operación inversa a RotWord y la sustitución SubBytes.

Aplicando estas operaciones obtenemos las 2^{32} combinaciones posibles para la clave original. De aquí en adelante el ataque consiste en probar todas las posibilidades hasta que, como se ha dicho, se dé una coincidencia entre el criptograma obtenido con la clave original y el obtenido con una de las candidatas.

Cabe destacar que el ataque que ha servido como modelo constaba de un segundo paso, antes de deshacer la generación de subclaves, en el que las candidatas quedan reducidas a 2^8 posibilidades totales pero su implementación quedará pendiente para trabajos futuros ya que 2^{32} posibilidades siguen siendo asumibles computacionalmente.

5.3 Resultados

La realización de este criptoanálisis permite reconocer que hasta un algoritmo seguro y tan utilizado a nivel global puede ser vulnerable a este tipo de ataques. El hecho de que el algoritmo sea de dominio público lo expone a que cualquier persona pueda estudiarlo y analizar su comportamiento frente a diversas situaciones consiguiendo, así, sacar a la luz sus puntos débiles.

Esta simulación ha permitido comprobar la viabilidad del ataque para que, en un futuro, pueda implementarse con garantías en un sistema real.

6 MEDIDAS DE PROTECCIÓN

El objetivo principal de este trabajo era, tras reproducir el ataque, encontrar medidas de protección del algoritmo que pudieran prevenirlo de ataques del mismo tipo. En el capítulo anterior se ha demostrado que, con una sola inyección de error, se puede recuperar la clave completa del AES-128. Esto es la señal de alarma de que para no comprometer la seguridad es esencial dotar de protecciones frente a ataques de fallos al sistema. En este capítulo van a recogerse algunas de las medidas de seguridad más comunes, aunque, a diferencia de los dedicados a ataques, no hay demasiados artículos publicados a este respecto [22].

En el caso de los cifradores de bloque y, en concreto, el AES, estas son algunas de las medidas de seguridad más aplicadas [26]:

6.1 Redundancia modular (Modular Redundancy)

Una da las formas más directas de proteger un sistema es repetir la ejecución del algoritmo o de alguna de sus partes, varias veces para comprobar que las salidas son las mismas y, en consecuencia, que no se ha inyectado error.

La ejecución puede realizarse en paralelo o de forma secuencial. Implementarlo en paralelo, también conocido como redundancia espacial, consiste en clonar alguna función o el algoritmo completo de forma que se tengan dos circuitos independientes. A lo largo del proceso se irán comparando los Estados intermedios en algún punto o, el criptograma final para asegurar que coinciden. El principal problema de hacerlo de esta manera es que requiere un alto coste de hardware.

Si la implementación elegida es la secuencial, también conocida como redundancia temporal, no trae incorporado ningún coste de hardware, pero si se sacrifica el tiempo que tarda en computarse.

Este tipo de medidas de protección es fácilmente evadible si el mismo ataque se repite en varias ocasiones consecutivas pues se convertiría en indetectable, al menos, para la redundancia temporal. En el caso de la especial, el atacante debe conseguir inyectar el mismo fallo en ambas implementaciones y, si se modifica el layout de una a otra esta tarea puede complicarse extremadamente.

6.1.1 Redundancia modular dual (Dual Modular Redundancy – DMR)

En este caso, el Sistema a proteger se duplica o clona. Su principal problema es que, por lo general, incrementa el área de la implementación hardware, aunque, en el caso de los cifradores de bloque, puede evitarse en determinados casos. Algunos de los métodos más comunes de implementar esta medida son:

- Usar el inverso

Los cifradores de bloques y, en concreto, el AES, ofrecen la posibilidad de implementar el proceso de descifrado. De esta forma, la inyección del fallo es detectada descifrando el criptograma final y comparándolo con el texto plano de entrada.

- Cifradores involutivos

En este caso, se utilizan las operaciones involutivas que son aquellas que, al repetir la misma operación sobre la salida se vuelve a obtener la entrada. Un ejemplo es la operación Or-Exclusiva utilizada por ejemplo en la función AddRoundKey.

$$A \oplus B = C \quad \rightarrow \quad C \oplus B = A$$

6.2 Feedback Modes

Esta medida de protección se basa en los modos de operación que, como se explicó en capítulos anteriores, son los encargados de dividir las grandes cantidades de datos en bloques admisibles por los algoritmos de cifrado, en el caso del AES, bloques de 128 bits. La medida consiste en crear algún tipo de dependencia entre los diferentes bloques a cifrar.

Si se utiliza el modo de operación más simple, el Electronic Code Book (ECB), cada bloque es cifrado independientemente por lo que, entradas iguales producirán salidas iguales. Esto facilita la labor al atacante que sabe que la única diferencia a la salida de dos bloques de entrada idénticos corresponde al error inyectado. La solución a este problema es el uso de Feedback Modes como, por ejemplo, el Cipher Block Changing (CBC) que implementa una operación Or-Exclusiva entre el criptograma obtenido del bloque anterior y el texto del nuevo bloque.

Los Feedback Modes aportan seguridad, pero, por otro lado, imposibilitan la implementación en paralelo de cada bloque.

6.3 Protección de los bucles

Este tipo de medida de seguridad protege el sistema frente a un intento de reducción de rondas. Consiste en proteger el contador que controla bucle principal para que el atacante no pueda modificarlo y no consiga reducir el número de rondas y obtener la clave. Su implementación radica en programar otro contador de ronda independiente que, de alguna forma se comunique con el original y comprueben que todo está correcto.

6.4 Comprobaciones cíclicas de redundancia (CRC)

Si un atacante es capaz de modificar un registro crítico como el de la clave, sería un desastre porque, cambiando determinados valores, puede asumir que, si el bit correspondiente al valor cambiado en el criptograma final no ha sufrido modificaciones, se ha acertado ese bit de la clave en concreto. Para evitar este tipo de ataques surge esta medida que consiste en proteger a toda costa este tipo de registros incorporando, por ejemplo, un registro basado en una suma de comprobación.

Estas propuestas de mejora son aplicables a la mayoría de cifradores de bloque. Si nos fijamos en el caso concreto del AES, pienso que, puesto que el grado de seguridad de un algoritmo depende del secretismo de la clave, además de las protecciones descritas, podría mejorarse su robustez complicando algo más el proceso de generación de las claves de manera que sea prácticamente imposible recuperar la original a partir de la de la décima ronda.

7 CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo se ha estudiado en profundidad el algoritmo criptográfico simétrico más utilizado en la actualidad, el Advanced Encryption Standard. Se ha hecho con el fin de poder demostrar sus vulnerabilidades y así, conseguir romper su proceso de cifrado para obtener la clave secreta. Con la realización del trabajo se ha demostrado que, a pesar de lo extendido de su uso, el AES también es susceptible a ataques de fallos, en concreto, al Differential Fault Analysis.

El desarrollo de este trabajo permite llegar a la conclusión de que el futuro del AES reside principalmente en ampliar los tamaños de clave permitidos para el cifrado. Esta conclusión viene motivada por el hecho de que existe una gran diferencia entre atacar AES-128 y atacar AES-256, para el que hay muchas menos publicaciones, por lo que, si fuera AES-512, el trabajo a realizar por los atacantes sería muchísimo mayor. Además de la ampliación de la longitud de la clave y la complicación de su proceso de generación de subclaves, el AES tiene, algún otro problema como la simplicidad de su estructura algebraica o que cada bloque se encripta exactamente de la misma manera.

Sin embargo, a pesar de tener algunos aspectos mejorables, se puede apostar por su permanencia como algoritmo criptográfico estándar porque, no existe hoy en día, un competidor claro del mismo.

Este trabajo queda muy abierto a ser continuado en trabajos futuros.

En primer lugar, está la posibilidad de mejorar el ataque reproducido. Se llevaría a cabo implementando el segundo paso del ataque, que era capaz de reducir las posibilidades clave a, tan solo, 2^8. Esta segunda parte va acompañada de una mayor carga computacional por lo que quizá habría que buscar alternativas a su instrumentalización en VHDL.

Otro de los caminos más interesantes por los que puede ser continuado este trabajo es la implementación del ataque sobre un sistema real como la FPGA. Una vez comprobada la viabilidad del ataque mediante su reproducción en simulación se puede dar con certeza y garantías el salto a un sistema real. En concreto, sería realmente interesante realizar esta implementación física sobre FPGA ya que la Escuela y, en concreto el Departamento de Electrónica, cuenta con la herramienta FT-UNSHADES (Fault Tolerance University of Sevilla Hardware Debugging System) [27]. Es una plataforma de inyección de fallos que puede invertir el valor de un biestable que es precisamente en lo que se basa el ataque de fallos expuesto.

REFERENCIAS

[1] «Wikipedia - Criptografía,» 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Criptograf%C3%ADa. [Último acceso: Enero 2018].

[2] «CriptoBlog,» 2012. [En línea]. Available: https://sites.google.com/site/criptohuelin/historia-de-la-criptografia. [Último acceso: Enero 2018].

[3] M. Luna, «Blogspot - Criptografía y los diferentes tipos de cifrado,» 2012. [En línea]. [Último acceso: Mayo 2018].

[4] INTECO, «E-GOV,» 2013. [En línea]. Available: http://www.egov.ufsc.br/portal/conteudo/la-criptograf%C3%ADa-desde-la-antigua-grecia-hasta-la-m%C3%A1quina-enigma. [Último acceso: Mayo 2018].

[5] «Wikipedia - Historia de la criptografía,» 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Historia_de_la_criptograf%C3%ADa. [Último acceso: MAyo 2018].

[6] F. Pereñíguez, «Youtubbe,» UCAM Universidad Católica de Murcia, 2015. [En línea]. Available: https://www.youtube.com/watch?v=eZzCuGzwpdg&t=242s. [Último acceso: 2018].

[7] G. Ribera, «Infosegur,» 2013. [En línea]. Available: https://infosegur.wordpress.com/unidad-4/criptografia-simetrica-y-asimetrica/. [Último acceso: Mayo 2018].

[8] C. d. Ibiblio, «Ibiblio,» 2003. [En línea]. Available: https://www.ibiblio.org/pub/linux/docs/LuCaS/Manuales-LuCAS/doc-unixsec/unixsec-html/node310.html. [Último acceso: Enero 2018].

[9] A. Muñoz Muñoz, «TierradeLazaro.com,» Septiembre 2004. [En línea]. Available: http://www.tierradelazaro.com/wp-content/uploads/2016/04/AES.pdf. [Último acceso: Abril 2018].

[10] J. Daemen y V. Rijmen, The Design of Rijndael. AES - The Advanced Encryption Standard, Springer, 2001.

[11] «Wikipedia - Finite Field Arithmetic,» Abril 2018. [En línea]. Available: https://en.wikipedia.org/wiki/Finite_field_arithmetic. [Último acceso: Mayo 2018].

[12] J. Ramió, «Youtube,» Criptored, Noviembre 2015. [En línea]. Available: https://www.youtube.com/watch?v=tzj1RoqRnv0. [Último acceso: 2018].

[13] Ç. Aydoğan, «Blogspot - AES Cryptography,» 2012. [En línea]. Available: http://aescryptography.blogspot.com/. [Último acceso: Abril 2018].

[14] «Wikipedia - Rijndael MixColumns,» 2018. [En línea]. Available: https://en.wikipedia.org/wiki/Rijndael_MixColumns. [Último acceso: 2018].

[15] «Wikipedia - Ataque de canal lateral,» 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Ataque_de_canal_lateral. [Último acceso: 2018].

[16] D. Osvik, A. Shamir y E. Tromer, «Cache Attacks and Countermeasures: The Case of AES,» de *Pointcheval D. (eds) Topics in Cryptology*, Rehovot, 2006.

[17] H. Li y S. Moore, «Security evaluation at design time against optical fault injection attacks,» *Information Security, IEE Proceedings,* vol. 153, nº 1, pp. 3,11, 2006.

[18] J. Tombs, M. Aguirre, R. Palomo, J. Mogollon, H. Guzmán, J. Nápoles, A. Rodríguez-Perez, J. Rodríguez, A. Vega-Leal, Y. Morillas y J. García, «The experience of starting-up a radiation test at the 18MeV cyclotron in the Spanish National Acelerators Center,» de *RADECS 2007. 9th European Conference*, Deauville, 2007.

[19] C. Clavier, «Attacking Block Ciphers,» de *Fault Analysis in Cryptography*, Springer, 2012.

[20] J. Bömer y J. Seifert, «Fault Based Cryptanalysis of the Advanced Encryption Standard (AES),» de *Financial Cryptography. FC 2003. Lecture Notes in Computer Science*, Guadaloupe, 2003.

[21] G. Piret y J. Quisquater, «A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad,» de *Cryptographic Hardware and Embedded Systems. CHES 2003. Lecture Notes in Computer Science*, Cologne, 2003.

[22] C. Giraud, «Differential Fault Analysis of the Advanced Encryption Standard,» de *Fault Analysis in Criptography*, Springer, 2012.

[23] D. Saha, D. Mukhopadhyay y D. RoyChowdhury, *A Diagonal Fault Attack on the Advanced,* Kharagpur, 2009.

[24] M. Tunstall, D. Mukhopadhyay y S. Ali, «Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault,» de *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication. WISTP 2011. Lecture Notes in Computer Science*, Heraclión, 2011.

[25] B. Fekete, «Opencores,» 2017. [En línea]. Available: https://opencores.org/project/aes_all_keylength. [Último acceso: Julio 2017].

[26] J.-M. Schmidt y M. Medwed, «Countermeasures for Symmetric Key Ciphers,» de *Fault Analysis in Cryptography*, Springer, 2012.

[27] J. Mogollon, H. Guzmán-Miranda, J. Nápoles, J. Barrientos y M. Aguirre, «FTUNSHADES2: A novel platform for early evaluation of robustness against,» de *Radiation and Its Effects on Components and Systems (RADECS), 2011 12th European Conference*, 2011.

[28] V. Trayno, «UPCommons,» 2016. [En línea]. Available: https://upcommons.upc.edu/bitstream/handle/2117/99275/PFC_entrega.pdf?sequence=2&isAllowed=y. [Último acceso: 2018].