Departament d'Enginyeria Electrònica

UNIVERSITAT POLITÈCNICA DE CATALUNYA

“Generación de falsas claves criptográficas como medida de protección frente a ataques por canal lateral”

Tesis doctoral presentada para la obtención del título de doctor.

Rubén Lumbiarres López

Director: *Mariano López García*

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola de Doctorat

Acta de qualificació de tesi doctoral

Curs acadèmic: **2014-2015**

Nom i cognoms: **Rubén Lumbiarres López**

Programa de doctorat: **Enginyeria Electrònica**

Unitat estructural responsable del programa: **Departament d'Enginyeria Electrònica**

Resolució del Tribunal

Reunit el Tribunal designat a l'efecte, el doctorand / la doctoranda exposa el tema de la seva tesi doctoral titulada **"Generación de falsas claves criptográficas como medida de protección frente a ataques por canal lateral"**

Acabada la lectura i després de donar resposta a les qüestions formulades pels membres titulars del tribunal, aquest atorga la qualificació:

☐ NO APTE ☐ APROVAT ☐ NOTABLE ☐ EXCEL·LENT

(Nom, cognoms i signatura) President/a	(Nom, cognoms i signatura) Secretari/ària	
(Nom, cognoms i signatura) Vocal	(Nom, cognoms i signatura) Vocal	(Nom, cognoms i signatura) Vocal

_____, _____ d'/de _____ de _____

El resultat de l'escrutini dels vots emesos pels membres titulars del tribunal, efectuat per l'Escola de Doctorat, a instància de la Comissió de Doctorat de la UPC, atorga la MENCIÓ CUM LAUDE:

☐ SÍ ☐ NO

(Nom, cognoms i signatura) President de la Comissió Permanent de l'Escola de Doctorat	(Nom, cognoms i signatura) Secretari de la Comissió Permanent de l'Escola de Doctorat

Barcelona, _____ d'/de _____ de _____

11-E6-4E-61-1A-17-4E-98-D1-A4-8B-FD-74-65-2D-E6-12-F6-70-0E-0F-78
93-95-B1-E3-7B-66-78-AB-D0-1D-DB-ED-F8-7B-69-C1-79-24-0F-4C-79-2E
91-8B-F7-5B-25-97-14-55-B3-9C-5C-D7-41-52-23-F5-BD-D9-45-37-B0-31
C0-31-D4-72-39-7C-18-45-E8-B9-7C-89-0E-2A

30-31-32-33-34-35-36-37-38-39-41-42-43-44-45-46

La verdad absoluta no existe y esto
es absolutamente cierto.

Les Luthiers.

Agradecimientos

Quiero mostrar mi agradecimiento al director de esta tesis el Dr. Mariano López García, una persona sin la cual esta tesis no existiría. Son varias las razones que me llevan a reconocer públicamente la influencia que ha tenido en mi carrera profesional, en primer lugar fue quien me introdujo en el mundo del hardware reconfigurable, posteriormente dirigió mi proyecto final de carrera cuando cursé los estudios de segundo ciclo de ingeniería, y finalmente me propuso el tema de estudio de esta tesis. Su visión de los objetivos, su conocimiento sobre el hardware reconfigurable y su experiencia en el campo de la investigación han sido fundamentales para completar el arduo trabajo que representa la escritura de una tesis doctoral.

También quiero agradecer al Dr. Enrique Cantó Navarro que haya compartido conmigo alguno de los "secretos" de las FPGA que tan bien conoce.

Un agradecimiento muy especial va dirigido a mi esposa Yolanda y a mi hijo Noel, primero por dejarme llenar la casa de trastos y segundo, por permitirme pasar horas ante el ordenador mientras ellos preparaban numerosos exámenes de 3º, 4º y 5º de primaria.

No puedo olvidar a mis padres, Jacinto y Noeli, y a mi hermana Dámaris, ellos sufrieron mi afición por desmontar cosas que raramente podían volver a ser montadas. Mis padres supieron respetar mi decisión cuando tomé el camino de la Formación Profesional, en contra de la opinión de algunos maestros de la escuela de primaria. Incluso me compraron mi primer ordenador (Comodore 64), con el que aprendí a programar en un rudimentario ensamblador.

Finalmente quiero agradecer al Profesor Félix Prado Pueo que pusiera en mí el gusanillo de la ingeniería y de la electrónica.

Índice general

Abstract

In the late 90s, Paul C. Kocher introduced the concept of differential attack focused on the power consumption of a cryptographic device. In this type of analysis the plain text sent to the device is known, and all possible hypotheses of a subset of the key, related to a specific point of the cryptographic algorithm, are tested. If the key value at that point depends only on 1 byte, it is possible to predict the input current based on a theoretical model of power consumption. Thus, using statistic procedures, it is easy to compare the consumption measured during the processing of each plain text and the intermediate values related to all the hypothesis of the key. The one with the highest level of similarity will correspond to the actual key.

So far the countermeasures proposed to prevent the success of the attack can be classified into two groups: Masking and Hiding. Masking tries to decouple the processed data and the power consumption by adding a random mask which is unknown by the attacker. Therefore, it is impossible to make a hypothesis that allows the theoretical and the real power consumption of the device to be related. Although is a valid method, the key could be revealed by performing a second-order attack that analyzes several points of the

current trace. Hiding aims at making constant the consumption of a device in each clock cycle and independent of the processed data. In order to achieve this objective, the data is processed in double line, in such a way that the datum and its complementary are processed together, so that the same number of transitions always occurs on every clock cycle. The weakness of such a method lies on the impossibility of building identical CMOS cells, which causes a minimum difference of consumption between the two lines that can be used successfully to discover the key.

This thesis proposes a countermeasure based on a differentiated protection strategy with respect to the proposals made in other specific studies. It is intended to modify the algorithm in order to force a very high correlation with a different hypothesis to the one of the true key (Faking). Thus, the actual key is hidden behind the strong correlation, which is impossible to differentiate from the rest of false assumptions and remains protected.

To verify its performance a trial bank has been designed to launch consumption analysis attacks. We have implemented the algorithm AES due to its simplicity and strength. Two types of attacks have been carried out. In the first one, the analysis was performed using both the correlation and the mean difference analysis without including any countermeasure. In the second attack, the proposed countermeasure has been added and the attack was repeated to check its effectiveness.

We have evaluated three different situations. First of all, the algorithm and the countermeasure are solved by software on a 32-bit processor. Secondly, the algorithm is executed in software and the implementation of the countermeasure has been performed with a specific hardware coprocessor. Finally, a full hardware implementation including both the algorithm and the countermeasure has been chosen. All of them have been implemented on a Virtex 5 FPGA Xilinx.

Several conclusions are obtained from the comparison between each of the AES implementations without countermeasures and their respective solution with the added countermeasure. The obtained results are also compared to other which uses "masking" and "hiding" techniques.

The results demonstrate that the proposal is valid. In all three cases, the protected system behaves like the unprotected system but returning the false key after the attacks. It should be noted that the amount of resources needed to carry out the "Faking" is less than the "Masking" or "Hiding" and the time needed to process the plain text is not particularly affected.

Resumen

A finales de los 90 Paul C. Kocher introdujo por primera vez el concepto de ataque diferencial sobre el consumo de corriente de un dispositivo criptográfico. En este tipo de análisis se conoce el texto plano que se envía al dispositivo y se plantean todas las posibles hipótesis de la clave para un punto concreto del algoritmo. Si el valor en dicho punto depende únicamente de 1 byte de la clave, es posible calcular todos los valores que se producirán. Finalmente, se compara, por métodos estadísticos, el consumo medido durante el procesado de cada texto plano y los valores intermedios relacionados con todas las hipótesis de la clave. Aquella que mayor nivel de similitud tenga corresponderá con la clave real.

Las contramedidas propuestas hasta la fecha, para evitar el éxito del ataque, pueden separarse en dos grupos: enmascaramiento (Masking) y ocultación (Hiding). El enmascaramiento trata de desvincular el dato procesado del consumo eléctrico mediante la adición de una máscara aleatoria y desconocida por el atacante. En consecuencia, resulta imposible realizar una hipótesis que permita relacionar el consumo teórico y real del dispositivo. Si bien este es un método inicial-

mente válido, puede descubrirse la clave realizando un ataque de segundo orden que analiza varios puntos de la traza. La ocultación persigue que el consumo de un dispositivo sea el mismo en cada ciclo de reloj e independiente del dato procesado. Para ello, se procesa el dato en doble línea, por un lado el dato propiamente dicho y por el otro su complementario, de forma que siempre se produzcan la misma cantidad de transiciones en cada ciclo de reloj. La debilidad de este método radica en la práctica imposibilidad de construir celdas CMOS idénticas, lo que provoca una diferencia mínima de consumo entre las dos líneas que puede usarse con éxito para descubrir la clave.

En esta tesis se propone una contramedida basada en una estrategia de protección claramente diferenciada con respecto a las propuestas realizadas en la bibliografía específica. Se pretende modificar el algoritmo con el objetivo de forzar una correlación muy alta en una hipótesis diferente a la de la clave (Faking). De este modo, la clave real se oculta tras la fuerte correlación aparecida y resulta imposible diferenciarla del resto de hipótesis falsas.

Para verificar su funcionamiento se ha diseñado un banco de pruebas para realizar ataques por análisis de consumo. Se ha implementado el algoritmo AES debido a su simplicidad y robustez. Se han realizado dos tipos de ataques: en el primero se han practicado análisis de correlación y diferencia de medias sin contramedida alguna; en el segundo, se ha añadido la contramedida propuesta y se han repetido los ataques para comprobar su eficacia.

Se han evaluado tres escenarios diferentes. Primeramente el algoritmo y la contramedida se resuelven mediante software en un procesador de 32 bits. En segundo lugar, el algoritmo se resuelve mediante software y la implementación de la contramedida se ha realizado en un coprocesador hardware específico. Finalmente, se ha elegido una implementación totalmente hardware para resolver tanto el algoritmo como la contramedida. Todos ellos se han implementado sobre una FPGA Virtex 5 de Xilinx.

Las conclusiones se obtienen de la comparación entre cada una de las implementaciones del AES sin contramedidas y su respectiva solución con la contramedida añadida. También se comparan los resultados obtenidos con otros que utilizan las técnicas “Masking” y “Hiding”

Los resultados demuestran que la propuesta es válida. En los tres casos el sistema protegido se comporta igual que el sistema sin proteger, pero retornando la clave falsa ante los ataques realizados. Se ha de destacar que, la cantidad de recursos necesarios para llevar a cabo el “Faking” es menor que con el “Masking” o el “Hiding” y el tiempo necesario para procesar el texto plano no se ve particularmente afectado por su inclusión.

Índice de figuras

Índice de tablas

1. Introducción

El origen de la Historia se establece con la aparición de la escritura (Sumer 3000 AC). El hombre empezó a poner por escrito aquellas ideas o pensamientos que consideraba que no debieran perderse en el olvido; este hecho es descrito por James H. Breasted[1] como el inicio de la civilización.

El uso de la escritura permitió, no sólo el registro de hechos históricos, sino la comunicación entre personas mediante el uso de misivas que se hacían llegar de un punto a otro por diversos medios. Está claro que, al plasmar un mensaje en un soporte físico, para ser transmitido, cualquier persona que tenga acceso a él es capaz de conocer su contenido. Por tanto, la protección de la información queda supeditada a la fiabilidad del canal de comunicación utilizado.

La necesidad de proteger las comunicaciones, especialmente en el ámbito político-militar, y desvincular su seguridad del canal de comunicación hace aparecer el concepto de criptografía[2]. Si no es posible evitar que un mensaje caiga en manos enemigas hay que evitar que éstos sean capaces

[1]Arqueólogo, egiptólogo e historiador estadounidense. Rockford, Illinois, 27 de agosto de 1865 - Nueva York, 2 de diciembre de 1935

[2]Arte de escribir con clave secreta o de un modo enigmático (RAE). Del griego "kriptos" oculto, y "graphos" escritura.

de acceder a su contenido. La primera forma de codificación de mensajes de la que se tiene constancia es la "Escítala", que fue utilizada por los éforos espartanos para enviar mensajes secretos. Para la codificación se enrollaba un papiro o una cinta de cuero sobre un cilindro de madera y se escribía el mensaje. El destinatario que recibía la cinta de cuero únicamente debía conocer el diámetro del cilindro original para recuperar la información.

Desde entonces hasta la actualidad, la complejidad de los métodos criptográficos ha ido en aumento. Una serie de sistemas mecánicos, electro-mecánicos, así como modernos métodos informáticos se han diseñado para aumentar la protección de los mensajes cifrados. El uso de claves cada vez más largas y de algoritmos cada vez más complejos [Sta'2005] han dado como resultado comunicaciones más fiables.

Llegados a este punto hay que destacar que, paralelamente al hecho de que alguien pretenda ocultar una información, aparece quien pretende acceder a ella de forma fraudulenta. Las largas claves criptográficas utilizadas hoy en día (de 128 bits en adelante) dificultan notablemente, o incluso impiden, la obtención de la clave por simple fuerza bruta. Por esta razón, el atacante, en lugar de probar todos los posibles valores de la clave, ha de analizar el sistema y utilizar todas las fuentes de información que éste le aporte. Aparece el concepto de criptoanálisis[3].

La idea del criptoanálisis no es nueva. En el siglo IX, el matemático árabe Abū Yūsuf Ya'qūb ibn Isḥāq al-Kindī[4] ya describió un método para obtener claves criptográficas [San'2012]. Los métodos matemáticos tratan de analizar el criptograma[5] y hacer hipótesis en función de algunos datos estadísticos relacionados con el idioma en que se redactó el mensaje. Por ejemplo, una simple codificación por sustitución de caracteres de un texto en castellano puede iniciarse su descifrado sabiendo que las letras más comunes son la 'E' (14%) y la 'A' (12.2%)[6] o que existen las letras dobles

[3] Arte de descifrar criptogramas. (RAE)

[4] Sabio árabe que estudió matemáticas y lógica entre otras muchas ciencias y artes. (Kufa, actual Irak, 801 - Bagdad, 873)

[5] Documento cifrado. (RAE)

[6] En el texto de "Don Quijote de la Mancha".

("rr" "ll"). Buscando estas relaciones en el texto encriptado y tratando de completar palabras puede llegarse a su descifrado.

Ya en la era moderna, el criptoanálisis vive su momento álgido cuando un equipo de matemáticos y científicos del ejército aliado, entre los que se encontraba Alan Turing[7], lograron descifrar el funcionamiento de la máquina "Enigma" que el ejército alemán usaba para codificar sus comunicaciones. Basándose en la información obtenida por matemáticos polacos, Turing ideó, junto a Gordon Welchman[8], la máquina que denominaron "The Bombe" y que permitía discriminar si la clave utilizada en el descifrado era correcta o no. Según algunos historiadores, el trabajo de criptoanálisis realizado por este equipo supuso adelantar en dos años el fin de la segunda guerra mundial. Los trabajos se mantuvieron en secreto hasta l970.

1.1. Criptografía moderna

Según [Gra'2006] la aparición del algoritmo DES[9] [NIST'1977] es uno de los puntos que marca la aparición de la criptografía moderna. Por otro lado, [Dif'1976] mostraba los caminos que podía tomar la criptografía debidas a la aparición de los métodos electrónicos de proceso frente a los métodos mecánicos.

En la criptografía moderna se pueden distinguir dos métodos diferenciados con los que proceder al cifrado de la información:

a) **Sistemas simétricos**. Son aquellos en los que la información se codifica y se decodifica utilizando la misma clave. El emisor de la información deberá hacer llegar la clave al receptor por medio de algún canal seguro. Por tanto, un emisor deberá tener una clave privada por cada posible interlocutor y la seguridad de las comuni-

[7] Matemático, lógico, científico de la computación y criptógrafo inglés. Fue el primero en proponer la idea de una máquina programable (Paddington, Londres, 23 de junio de 1912 - Wilmslow, Cheshire, 7 de junio de 1954)

[8] Matemático y profesor universitario ingles. (June 15, 1906, Bristol, England – October 8, 1985, Newburyport, Massachusetts, USA)

[9] De las siglas en inglés "Data Encryption Standard".

caciones dependerá de la seguridad del canal de transmisión de la clave.

b) **Sistemas asimétricos**. En este caso se utiliza una clave para la codificación y otra clave para la decodificación de la información. Es decir, un usuario dispone de una clave pública conocida por todos los usuarios del sistema, de modo que cualquier mensaje dirigido a él debe ser codificado con su clave pública. En cambio, el descifrado del mensaje únicamente puede hacerse con la clave privada y que sólo el usuario conoce. Al no existir la necesidad de enviar la clave privada a través de ningún canal de comunicación, la seguridad de la comunicación no se ve comprometida.

Al mismo tiempo los sistemas simétricos se pueden dividir en:

a) **Sistemas de cifrado por bloques**, aquellos en los que cada operación de cifrado se realiza sobre un bloque de información, generalmente mediante algoritmos iterativos que realizan su función en base a la repetición de operaciones básicas.

b) **Sistemas de cifrado en flujo,** a partir de una clave inicial y un proceso determinista de expansión, se obtienen sucesivas claves diferentes para los sucesivos textos planos. La información se codifica a nivel de bit mediante una operación lógica o una sustitución en la que intervienen el texto plano y la clave.

El algoritmo AES[10] [NIST'2001] es el estándar más conocido de algoritmo de clave privada y de encriptación por bloques. La norma establece que este algoritmo trabaja con bloques de información de 128 bits y con longitudes de clave de 128, 192 o 256 bits, en función del nivel de seguridad que se pretenda obtener. En esta tesis doctoral la investigación se centra en la versión de 128 bits del algoritmo AES.

1.2. Criptoprocesadores.

Los criptoprocesadores son sistemas electrónicos digitales que tienen por misión aplicar algoritmos matemáticos a cadenas de información para

[10] De las siglas en inglés “Advanced Encryption Standard”

ocultar o revelar su contenido. Estos dispositivos pueden estar construidos en base a microprocesadores que ejecutan un programa, integrados reconfigurables tipo FPGA[11] cuya funcionalidad se describe mediante algún lenguaje HDL[12], o integrados ASIC[13] diseñados específicamente para la realización de tareas criptográficas. Las necesidades de cada aplicación práctica determinará la tecnología apropiada para realizar la implementación del sistema.

Actualmente los criptoprocesadores se encuentran embebidos en sistemas a diferentes escalas. Se encuentran en pequeñas tarjetas inteligentes[14], en las que un microprocesador (de 8, 16 o 32 bits) ejecuta algoritmos para proteger la información que se transmite hacia o desde el lector correspondiente. En la misma línea, las tarjetas criptográficas incorporan, a parte del microprocesador, hardware específico para realizar las tareas de codificado o decodificado de la información. En el otro extremo, existen las tarjetas criptográficas para ordenadores corporativos que incorporan memoria, procesador, coprocesadores criptográficos y sistema operativo propio, tienen por misión proteger las comunicaciones y la información del equipo en el que está instalada.

1.3. Criptoanálisis.

La aparición de los criptoprocesadores permite procesar grandes cantidades de información y, a su vez, codificarla con extensas claves. El uso de dispositivos, con cada vez más potencia de cálculo, facilita esta tarea reduciendo el tiempo necesario para realizarla. En estas condiciones, el criptoanálisis basado únicamente en el análisis matemático de los criptogramas encuentra numerosas dificultades para desvelar las claves. Por consiguiente, un atacante necesita información suplementaria que le ayude en su propósito.

A finales del siglo XX se introdujo el concepto de "Ataque por Canal Lateral". En este caso, a parte del texto plano o criptograma, el atacante

[11] De las siglas en inglés "Field Programmable Gate Array"
[12] De las siglas en inglés "Hardware Description Language"
[13] De las siglas en inglés "Application-Specific Integrated Circuit"
[14] También denominadas "Smart Cards"

utiliza información recogida de parámetros físicos relacionados con el funcionamiento del dispositivo criptográfico. Entre otras magnitudes puede extraerse información del consumo eléctrico, la radiación electromagnética, el ruido, la temperatura, el tiempo de procesado, etc.

Un ataque por canal lateral implica: tener acceso al dispositivo atacado y tener acceso al bus de comunicación para poder enviar datos al criptoprocesador; disponer de una sonda de medida adecuada a la magnitud que pretende monitorizarse; y contar con un sistema de captura y almacenamiento de los valores medidos. El proceso se completa mediante el análisis de los valores medidos relacionándolos con los datos enviados.

1.4. Objetivos de la tesis

En esta introducción el criptoanálisis se ha definido como el arte de descifrar criptogramas. Parece obvio que tratar de descifrar una clave cuyo creador no quiere que sea descifrada resulte una actividad ilícita, si no ilegal. Por el contrario, el criptoanálisis puede ser utilizado para valorar la seguridad de los sistemas, localizar sus puntos débiles y emprender aquellas modificaciones dirigidas a eliminarlos. Las acciones dirigidas a eliminar los puntos débiles se conocen bajo en nombre de "Contramedidas".

El objetivo principal de esta tesis es proponer una contramedida en la que, si un atacante intenta descubrir una clave mediante un análisis por canal lateral, el proceso que realiza lo conduzca a una resultado distinto al de la clave real. La técnica propuesta para proteger el sistema mediante la obtención de una clave falsa se ha denominado "Faking[15]"

Los objetivos específicos de la tesis son:

a) **Enunciar** los fundamentos matemáticos y estadísticos en los que se basa la contramedida propuesta.

b) **Implementar el algoritmo AES** [NIST'2001] sin contramedidas para ser utilizado como referencia de comparación. Se implementará un sistema totalmente software, en el que todo el proceso se

[15] Siguiendo la nomenclatura utilizada en la literatura, de "Mask" se deriva "Masking", de "Hide" "Hiding" y de "Fake" "Faking".

resuelve en un microprocesador. Por otro lado, se implementará un sistema totalmente hardware en el que la contramedida se resuelve mediante un circuito digital descrito para FPGA.

c) **Desarrollar un banco de trabajo** que permita obtener las claves del sistema criptográfico mediante análisis por canal lateral. El sistema será determinante a la hora de extraer las conclusiones del trabajo, pues permitirá saber si la contramedida propuesta cumple su propósito.

d) **Aplicar la contramedida** a los sistemas de referencia para verificar su funcionamiento y la forma en que afecta a las prestaciones y/o a los recursos necesarios.

e) **Comparar** los resultados obtenidos con otras contramedidas propuestas hasta la fecha por la comunidad científica.

1.5. Estructura de la tesis

Esta tesis se ha estructurado en 5 capítulos según la siguiente descripción:

Capítulo 2- Estado del arte de los SCA: Se describe lo que son los análisis por canal lateral basados en el análisis de consumo, los tipos más comunes y los modelos que suelen utilizarse. También se exponen las tendencias generales que se usan para la protección de los dispositivos ante este tipo de ataques.

Capítulo 3-Contramedida basada en "faking": En esta parte se exponen las vulnerabilidades más explotadas del algoritmo AES para la realización de ataques por canal lateral. Se enuncian los fundamentos de la contramedida. Se determinan las características fundamentales que debe tener cada una de las implementaciones sobre las que se va a verificar el trabajo.

Capítulo 4-Sistema de desarrollo, medidas y procesado: Se realiza una descripción detallada de la plataforma de trabajo sobre la que se implementarán cada uno de los sistemas. Se detallan las características de cada uno de los elementos que conforman el banco de pruebas. Se detalla la forma en que se realizará la toma de datos

y posterior procesado de los mismos y se determina la forma en que se presentarán los resultados.

Capítulo 5-Resultados experimentales: Se muestra la implementación de los sistemas de referencia software y hardware, así como el resultado de realizar ataques por análisis de consumo en diversos puntos del algoritmo. Se detalla la forma en que se implementa la contramedida en los sistemas software, software/hardware y totalmente hardware, se realizan los mismos ataques que a los sistemas de referencia. Finalmente se comparan los resultados obtenidos con otras contramedidas basadas en las técnicas hasta ahora publicadas.

Capítulo 6-Conclusiones y futuras líneas de trabajo: En el capítulo final, se muestran las conclusiones de este trabajo y se proponen futuras líneas de investigación sobre la implementación de la contramedida presentada en esta tesis.

2. Estado del arte de los SCA

2.1. Introducción

Fue en 1999 cuando Kocher y otros autores [Koc'1999] revelaron que uno de los puntos débiles de los algoritmos criptográficos aparece por el simple hecho de ejecutarse sobre dispositivos físicos. El funcionamiento de dichos dispositivos provoca variaciones en algunas magnitudes físicas, que son mesurables externamente, tales como el consumo de corriente o las emisiones de radiaciones electromagnéticas [Mar'2013]. La observación de estas magnitudes puede conducir a descubrir la clave criptográfica, dado que su variación está directamente relacionada con el dato que en cada instante procesa el sistema. Estos intentos de obtención fraudulenta de claves criptográficas se denominaron ataques por canal lateral o SCA[16].

Esta nueva realidad provocó que investigadores de diversos campos científicos dedicaran sus esfuerzos a descubrir las vulnerabilidades relacionadas con el dispositivo físico y a tratar de eliminarlas con el objeto de crear sistemas más robustos en cuanto a la protección de los datos encriptados se refiere.

[16] De las siglas en inglés "Side Channel Attack"

En este capítulo se hace una introducción a los ataques por canal lateral basados en el consumo eléctrico del dispositivo. Posteriormente se describe los ataques por análisis de correlación estadística y por diferencia de medias, y se detallan las contramedidas desarrolladas para evitarlos.

2.2. Ataques por canal lateral

En principio podría pensarse en la posibilidad de atacar un dispositivo criptográfico por fuerza bruta, es decir, intentando todas las claves posibles hasta dar con la correcta. El aumento de la potencia de cálculo de los sistemas informáticos ha provocado que algoritmos de encriptado con claves cortas puedan ser atacados en un corto periodo de tiempo y a un coste económico realmente bajo. Como ejemplo el algoritmo DES [NIST'1977] con clave de 56 bits puede ser descifrado en media hora utilizando una FPGA de bajo coste [Qui'2005]. Esta es la razón por la que el NIST[17] dejó de considerarlo seguro.

Para evitar un ataque por fuerza bruta es necesario aumentar el número de bits de la clave; con cada bit añadido se dobla el número de claves disponibles y se divide a la mitad la probabilidad de encontrarla.

En 2001 se publicó el algoritmo AES con longitudes de clave de 128, 192 y 256 bits según la versión que se implemente [NIST'2001]. El ataque por fuerza bruta aplicado a este algoritmo se considera inviable, dado que el tiempo necesario para obtener la clave, si la longitud es de 128 bits, se estima en 2^{16} años utilizando el dispositivo antes citado.

Ahora bien, si consideramos que el algoritmo debe ser implementado en algún sistema informático, podemos tratar de encontrar vulnerabilidades físicas no relacionadas con la propia concepción del algoritmo. Por contra el atacante debe conocer datos característicos del dispositivo tales como el tipo de implementación (software/hardware), el algoritmo utilizado, la velocidad del reloj del sistema o los buses de comunicación utilizados (RS232, USB, I2C,…); también ha de tener acceso físico al dispositivo para poder monitorizar parámetros físicos como por ejemplo la corriente consu-

[17] Actualmente "National Institute of Standards and Technology", anteriormente NBS "National Bureau of Standards".

mida, la radiación electromagnética emitida o los tiempos de ejecución del algoritmo [Koe'2005].

Los ataques físicos pueden clasificarse en dos grupos [Man'2007]:

a) **Invasivo – Semi invasivo – No invasivo:** El invasivo requiere el des-encapsulado del circuito integrado para acceder a los componentes internos mediante estaciones de medida o prueba, con la intención de conocer el estado de algunas señales, buses o registros internos. En el caso del semi invasivo, se retira la capsula del circuito integrado pero no se establece contacto eléctrico con el interior para conocer los datos necesarios [Sam'2002]. El no invasivo explota la información externamente disponible sin alterar el dispositivo en lo más mínimo.

b) **Activo – Pasivo:** El activo trata de inducir externamente un error en el dispositivo a través de sus entradas o del entorno para estudiar la reacción del mismo. El pasivo observa el funcionamiento dentro de sus especificaciones técnicas sin tratar de alterarlo.

El ataque por canal lateral se trata de un ataque pasivo, no invasivo, que no deja evidencia alguna de que el sistema haya sido atacado. Este tipo de ataques pueden dividirse en tres grupos en función de la información que explotan:

a) **Ataques por análisis del tiempo:** en los que la información que se explota está relacionada con el tiempo de ejecución de las diferentes partes del algoritmo [Koc'1996].

b) **Ataques por análisis de la radiación electromagnética:** en los que se analizan las perturbaciones que, en el espectro electromagnético próximo al sistema, provoca el funcionamiento del dispositivo [Mar'2013].

c) **Ataques por análisis del consumo:** se analiza el consumo eléctrico del dispositivo [Lum'2013].

El estudio realizado en esta tesis doctoral se basa en los ataques por análisis de consumo.

2.3. Ataques por análisis de consumo.

A juzgar por la cantidad de referencias bibliográficas se infiere que el análisis por consumo de corriente es el más popular dentro de la comunidad científica. A ello ha contribuido el hecho de que la corriente consumida por un circuito integrado sea una magnitud fácil de observar y a un coste económico bajo [Song'2008] [Jev'2011].

Los procesadores criptográficos consumen energía durante su funcionamiento y esta energía depende del dato que se está procesando en cada momento. En la Figura 2.1 puede apreciarse la diferencia de consumo que se produce en un dispositivo PIC de 8 bits cuando se mueve un dato desde un registro a la memoria. Si el dato movido corresponde al valor hexadecimal "FF" (línea de trazos) el consumo es mayor que si el dato movido corresponde al valor hexadecimal "00" (línea continua) [Lum'2013].

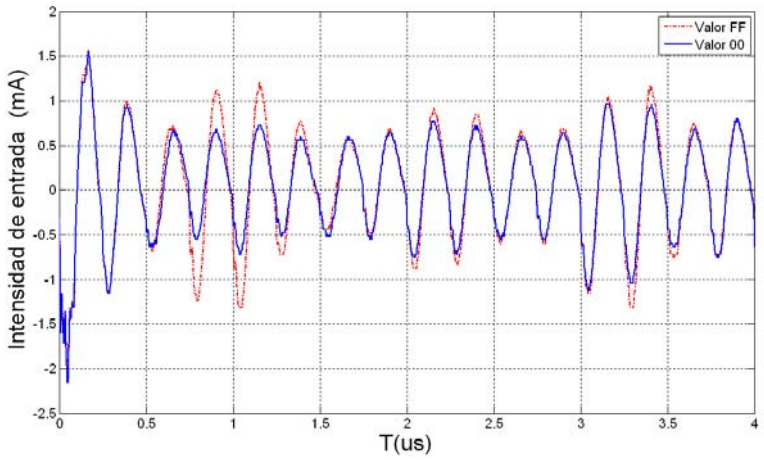

Figura 2.1: Comparación del consumo en un dispositivo PIC de 8 bits en función del dato procesado.

El análisis de consumo persigue descubrir la relación del par (energía, dato) con el objetivo de descubrir el segundo de ellos observando el primero. Para esto es necesario conocer un modelo de consumo de forma que estudiando y comparando los pares (energía, modelo) y (energía, dato) podamos llegar a deducir el dato procesado.

Cada dispositivo dispone de un modelo de consumo diferente y característico que depende de su estructura interna [Deg'2005] [Hok'2008]

[Dar'2014]. La elección del modelo adecuado es determinante a la hora de establecer la comparación entre éste y el consumo real del dispositivo.

El atacante suele desconocer la estructura interna del dispositivo, por tanto el modelo de comparación no puede ser ajustado con precisión. Ante este escenario se opta por el uso de modelos aproximados que en múltiples publicaciones muestran su validez [Pee'2007] [Cor'2000] [Man'2007]. Estos modelos son:

a) **Peso de Hamming[18] (HW[19]):** corresponde con el número de bits de un registro que están en valor lógico alto.

b) **Distancia de Hamming (HD[20]):** corresponde con el número de bits que cambian entre el estado actual y anterior de un registro.

La mayoría de los dispositivos integrados que se usan en la actualidad están creados a partir de puertas lógicas que tienen como base celdas CMOS, por tanto es necesario estudiar el consumo en estas celdas para poder caracterizarlo.

2.3.1. El consumo en dispositivos CMOS

El consumo total de un dispositivo CMOS es la suma del consumo de todas sus celdas lógicas. Así pues, el consumo depende de la cantidad de celdas lógicas y de las conexiones entre ellas [Man'2007].

Los elementos a considerar a la hora de modelar el consumo son dos: en primer lugar la celda CMOS es alimentada por una corriente continua V_{DD} proveniente de una fuente de alimentación externa; y en segundo lugar se modela, mediante la inclusión de un condensador virtual, las capacidades parásitas debidas a la conexión entre una celda CMOS y la siguiente. A su vez la celda es controlada por una señal de entrada que provoca transiciones en la salida (Figura 2.2).

[18] Richard Wesley Hamming (1915–1998) matemático estadounidense que trabajó en temas relacionados con la informática y las telecomunicaciones.

[19] De las siglas en inglés "Hamming Weight".

[20] De las siglas en inglés "Hamming Distance".

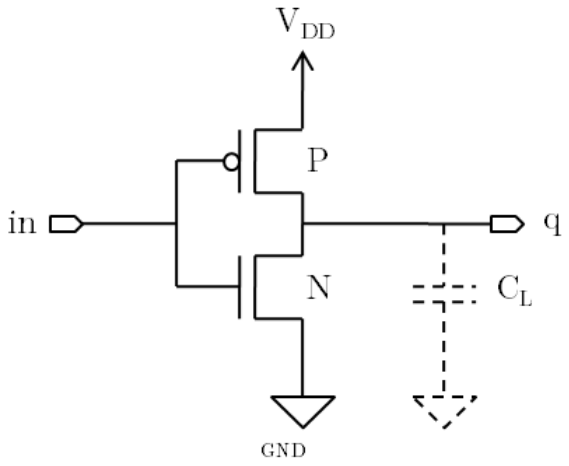

Figura 2.2: Celda CMOS básica

Según este modelo el consumo puede divirse en una parte estática y otra parte dinámica. La parte estática es debida a la corriente de fuga de los transistores (valores típicos del orden de 1 pA). La parte dinámica tiene su origen en la conmutación de la salida de las celdas CMOS y está formada por dos componentes:

a) La carga o descarga de las capacidades parásitas en las líneas de conexión entre una celda y la siguiente. Estas capacidades se cargan desde la fuente cuando la transición es ascendente (L-H) y se descargan a masa cuando la transición es descendente (H-L) (Figura 2.3).

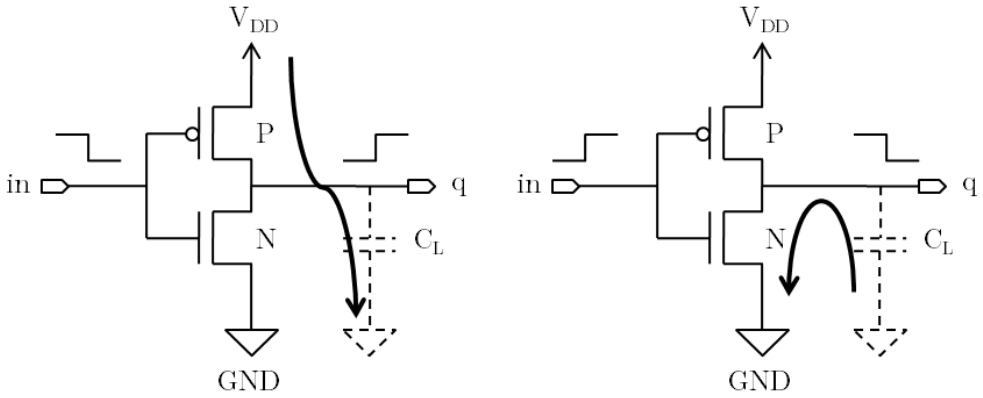

Figura 2.3: Corrientes en las celdas CMOS: a) Transición L-H b) Transición H-L

b) Las corrientes de cortocircuito que se producen en la transición debido a que el tiempo de conmutación de los transistores no es cero y durante un tiempo mínimo ambos están conduciendo.

La Tabla 2.1 muestra las componentes del consumo, medido desde la fuente de alimentación externa, en función de la transición que se produzca.

Transición a la salida	**Tipo de consumo**
L→L	Estático
L→H	Estático + carga del condensador + cortocircuito
H→L	Estático + cortocircuito
H→H	Estático

Tabla 2.1: Composición del consumo de corriente en las diferentes transiciones.

Como puede apreciarse, el consumo en una transición ascendente (L→H) es superior al consumo de una transición descendente (H→L). Es posible deducir que, el consumo transitorio de un grupo de celdas lógicas dependerá del dato que en cada momento se esté procesando. Para un chip genérico [Cor'2000] propone que:

$$P_{TOT} = P_{EST} + P_{CC} + P_{CL}$$
$$P_{EST} < 0.05 \cdot P_{TOT}; \quad P_{CC} \cong 0.15 \cdot P_{TOT}; \quad P_{CL} > 0.80 \cdot P_{TOT} \tag{2.1}$$

P_{TOT}: Consumo total.

P_{EST}: Consumo estático.

P_{CC}: Consumo debido al cortocircuito.

P_{CL}: Consumo debido a la carga del condensador.

Ante esta realidad se puede concluir que el consumo estático es insignificante comparado con el consumo dinámico y que será este último el que caracterizará el consumo total del dispositivo.

El efecto se hace más presente cuando la salida de la celda ataca a un bus local del procesador. Los buses suelen ser largas líneas de distribución de datos con múltiples dispositivos de entrada y salida, comunicaciones,

contadores, temporizadores o memorias conectadas a ellos. El gran tamaño de éstos, comparados con el resto de componentes del sistema, provoca que la capacidad parásita presente en la salida sea mayor y el efecto que esta provoca sobre el consumo gane importancia.

A título de ejemplo se adjunta la estructura interna de un micro controlador PIC de Microchip[21], en donde puede apreciarse como el bus de datos distribuye información a múltiples dispositivos (Figura 2.4). Cualquier transición que se produzca en los datos del bus reflejará un importante transitorio en el consumo del procesador.

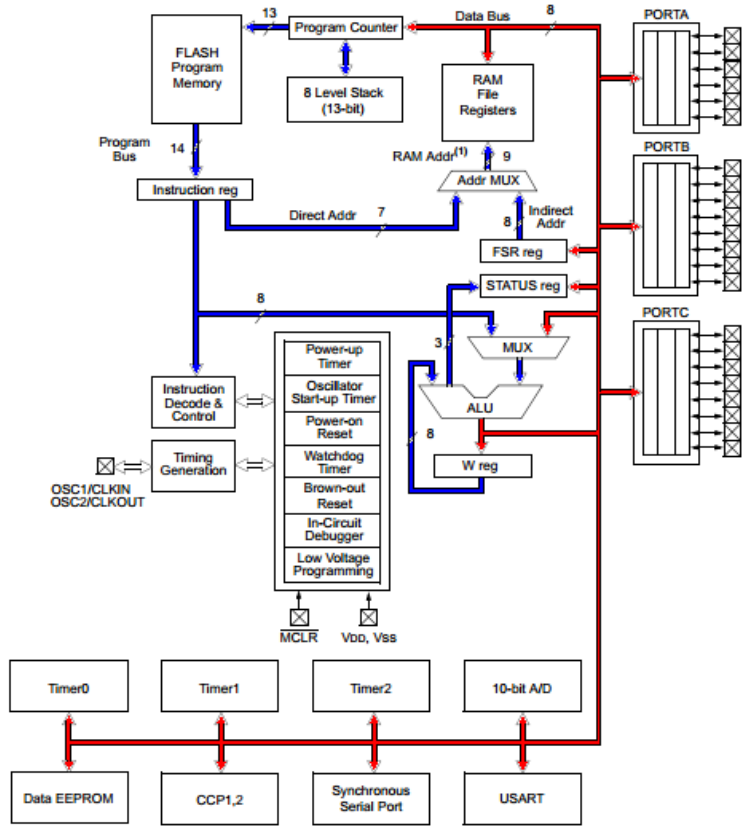

Figura 2.4: Estructura interna de un procesador PIC de Microchip.

2.3.2. Altura o peso de Hamming (HW)

Como se ha introducido anteriormente, la altura de Haming corresponde con la cantidad de bits en estado lógico alto que se almacena en un conjunto de celdas CMOS. Este modelo representa una aproximación respecto del funcionamiento real, pues presupone que las transiciones L→H y

[21] Microchip Technology Inc. - http://www.microchip.com/

H→H concentran todo el consumo de la celda lógica y no se toman en consideración las transiciones restantes, L→L y H→L. Esta aproximación supone que no hay dependencia entre el valor que almacena un conjunto de celdas lógicas y el valor que anteriormente almacenaba, y se fundamenta en el hecho aproximado de que un valor con un alto HW tendrá una huella de consumo diferente a un valor con un nivel HW (Véase Figura 2.1).

La aproximación planteada en este modelo, aun siendo una aproximación muy lejana de la realidad, ha sido utilizada con éxito para realizar ataques por canal lateral [Ren'2009] [Yua'2010] [Hiu'2011]. El modelo ha dado buenos resultados en aquellos dispositivos que efectúan una precarga lógica fija antes de la cargar del valor definitivo, o aquellos que disponen de un bus local, compartido por diversos dispositivos. Este comportamiento puede asemejarse a una precarga lógica a valor lógico bajo [Lum'2013] [XIL'2001].

2.3.3. Distancia de Hamming (HD)

En este caso la aproximación efectuada es considerar que las transiciones L→H y H→L provocan el mismo consumo y se obvian las transiciones L→L y H→H, que por otro lado únicamente afectan al consumo estático (véase Tabla 2.1). En este modelo es necesario conocer el valor actual y el valor anterior almacenado en un conjunto de celdas CMOS para calcular la cantidad de transiciones, tanto ascendentes como descendentes, que se han producido y poder estimar el consumo producido [Bri'2004] [Mey'2010].

Este modelo es apropiado para realizar ataques en implementaciones hardware [Man'2007] sobre dispositivos ASIC o FPGA en los que se conozcan los valores consecutivos que se almacenan en los registros.

Una versión más aproximada de la distancia de Hamming se puede obtener si el atacante dispone de información suficiente para diferenciar el consumo de las transiciones L→H del de las transiciones H→L. Es posible de este modo crear un modelo específico para una determinada implementación [Pee'2007] [Li'2010].

2.4. Análisis simple del consumo (SPA[22])

El análisis simple del consumo es una técnica en la que se debe interpretar las trazas de consumo de un dispositivo mientras efectúa operaciones criptográficas [Koc'1999]. Según [Man'2007] es posible conseguir la clave mediante la obtención de pocas trazas de corriente, incluso en un caso extremo, se puede obtener con una única traza siempre y cuando la clave tenga un impacto significativo en el consumo de energía y la cantidad de ruido electrónico sea pequeña [Man'2003].

En un primer análisis de la traza puede observarse la diferencia de consumo si el procesador ejecuta una parte del programa con poca carga computacional, como por ejemplo las comunicaciones RS232, o bien si realiza un proceso de gran carga computacional, como es el proceso de encriptado (Figura 2.5). Pero para poder extraer información relevante el atacante debe conocer muy detalladamente características del sistema tales como:

a) **Implementación del sistema** (software/hardware): si se trata de una implementación software, deben conocerse las instrucciones a nivel de máquina que se ejecutan en cada instante y el número de ciclos que cada una de ellas ocupa. Si se trata de implementación hardware, deben conocerse qué registros y qué lógica combinacional se han definido en su descripción. Tanto en un caso como en otro, este nivel de detalle permite saber si hay partes que se ejecutan o no en función del valor de un determinado bit para tratar de detecta la huella de consumo en cada caso.

b) **Instante de ejecución** de las diferentes partes del algoritmo: es necesario conocer el instante en que se ejecuta cada una de ellas al punto de poder determinar el ciclo de reloj exacto en que se mueve un dato o se ejecuta una operación que dependa de la clave.

[22] De las siglas en inglés "Simple Power Análisis".

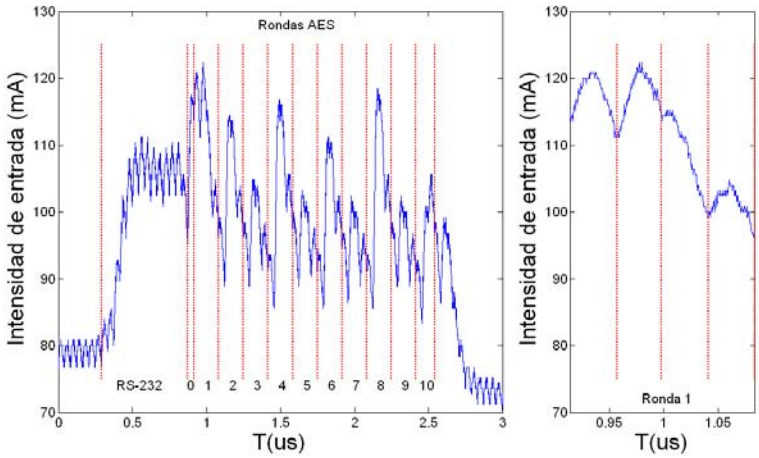

Figura 2.5: Traza de consumo de algoritmo AES sobre FPGA:
a) ciclo completo b) ampliación de la ronda 1

En este escenario es posible realizar un ataque por análisis simple del consumo para obtener la clave o, en la mayoría de los casos, reducir las combinaciones que debieran aplicarse a un ataque por fuerza bruta. Por ejemplo, si se puede determinar que la altura de Hamming de un byte de la clave es 2, las combinaciones a probar se reducen de 2^8=256 a las 28 que cumplen dicha condición.

En este escenario se han realizado ataques simples del consumo con éxito como muestran [Man'2003], en donde se ataca el bloque "*Key Expansion*" del algoritmo AES-128 [NIST'2001], o [Kad'2011] y [Li'2011] que exponen un ataque a un procesador criptográfico implementado, el primero de ellos, sobre una FPGA.

2.5. Análisis diferencial del consumo (DPA[23])

El análisis DPA es el más popular de los ataques por análisis de consumo. En este caso el atacante no necesita conocer el dispositivo de forma detallada. Por el contrario, deberá tener acceso al texto que se envía al dispositivo y a las trazas de consumo que con cada uno de ellos se produzcan.

[23] De las siglas en inglés de "Differential Power Análisis".

En este tipo de ataque se pretende analizar el consumo en un punto determinado del algoritmo y encontrar la relación con el dato que está siendo procesado en ese instante.

El objetivo es obtener la clave criptográfica basando su cálculo en la toma de una gran cantidad de medidas de corriente mientras se procesan otros tantos textos planos. Para ello el atacante necesita conocer el algoritmo de encriptación, pero no necesita información detallada de cómo éste ha sido implementado.

[Man'2007] propone la siguiente estrategia de trabajo formada por 5 puntos:

a) **Elección de un resultado intermedio en el algoritmo**. Es necesario analizar el algoritmo para encontrar el punto de ataque óptimo. Éste corresponderá a aquel punto en que el dato procesado dependa de una pequeña parte de la clave, y si es posible, de un solo byte de ella. A parte de esto, también debe ocurrir que el consumo sea función de un dato conocido, habitualmente el texto plano o el texto encriptado.

b) **Medir el consumo**, mientras se efectúan D ciclos de encriptado o des-encriptado con diferentes datos conocidos. Para cada uno de estos datos es necesario medir el consumo de los D valores intermedios que se producirán en el algoritmo cuando sean procesados. Las D trazas del consumo han de ser almacenadas en un sistema informático para poder proceder a su análisis con posterioridad.

c) **Cálculo del valor hipotético en el punto intermedio.** Debe calcularse este valor hipotético para cada una de los posibles valores que pueda tener la clave, si se ha elegido un punto en el que el dato dependa de un byte de la clave deberán calcularse los valores correspondientes a los 256 valores que la clave puede tomar. Este cálculo debe hacerse para los D valores conocidos que se han enviado en el apartado anterior, por tanto la cantidad de valores hipotéticos será de D x 256.

d) **Construcción del modelo,** a partir de los valores hipotéticos calculados anteriormente se determina el modelo de consumo para ca-

da uno de ellos. Puede usarse un modelo orientado a un bit aunque los modelos de altura de Hamming (Apartado 2.3.2) y peso de Hamming (Apartado 2.3.3), orientados a un byte, son los más usados en este punto.

e) **Comparación modelo hipotético con las trazas medidas**. En este punto se procede a discernir cuál de los 256 valores de la clave que se han usado en el punto c), de los que se han hecho 256 hipótesis, se aproxima en mayor medida a las trazas de consumo capturadas en el punto b). Aquella hipótesis que mayor semejanza tenga con la realidad corresponderá al valor real de la porción de clave que se persigue obtener.

El análisis diferencial ha demostrado ser una herramienta muy útil para la obtención de claves criptográficas [Han'2008] [Vel'2008] [Lum'2013] o para la comprobación de la fortaleza de un sistema ante eventuales ataques [Ito'2012].

Este tipo de ataques es el que se ha utilizado para el desarrollo de esta tesis doctoral y por esta razón se detallarán pormenorizadamente sus características.

2.6. Ruido en las trazas de corriente

Tanto en un SPA como en un DPA es necesario el estudio de las trazas de corriente. El éxito del ataque se ve directamente afectado por la presencia más o menos importante de ruido.

A parte de la caracterización que se ha hecho en el apartado 2.3.1 el consumo puede definirse como [Man'2007]:

$$P_{TOT} = P_{EXP} + P_{RUIDO} + P_{CNT} \tag{2.2}$$

P_{TOT}: Consumo total.

P_{EXP}: Consumo explotable por al atacante.

P_{RUIDO}: Consumo debido al ruido.

P_{CNT}: Consumo constante.

El consumo explotable es aquel del que el atacante puede obtener información para su propósito. Este consumo puede separarse en dos partes:

$$P_{EXP} = P_{OP} + P_{DATO} \tag{2.3}$$

P_{OP}: Consumo debido a la operación

P_{DATO}: Consumo debido al dato procesado

El consumo debido al ruido puede descomponerse, a su vez, en los siguientes elementos:

$$P_{RUIDO} = P_{CNM} + P_{EL} \tag{2.4}$$

P_{CNM}: Ruido procedente de la conmutación.

P_{EL}: Ruido electrónico

Los elementos que aparecen en (2.4) se definen como: $P_{CNM,}$ ruido procedente de los elementos del sistema que están conmutando y que no están relacionados con el dato procesado; P_{EL} es el ruido electrónico producido por los componentes del dispositivo.

De las ecuaciones (2.2), (2.3) y (2.4) se obtiene la siguiente expresión:

$$P_{TOT} = P_{EXP} + P_{CNM} + P_{EL} + P_{CNT} \tag{2.5}$$

En (2.5) todas las componentes del consumo son independientes entre sí [Man'2007]. La componente del consumo que se encuentra en el parte explotable depende de las características del ataque a realizar. Un ataque diferencial se basa en tomar múltiples trazas de un algoritmo cuando procesa diversos datos. Por tanto, el consumo producido por las operaciones es siempre constante y formaría parte del ruido de conmutación y no de la parte explotable. En cambio, en un ataque simple es importante el consumo de las operaciones para tratar de saber qué hace el procesador en cada momento, en este caso formaría parte de la información explotable.

Un factor determinante en el proceso es la relación señal ruido de las trazas, ésta se calcula según:

$$\text{SNR} = \frac{\text{VAR (Señal)}}{\text{VAR (Ruido)}} \tag{2.6}$$

En el contexto de un ataque por análisis de corriente la relación señal ruido puede calcularse mediante:

$$\text{SNR} = \frac{\text{VAR } (P_{EXP})}{\text{VAR } (P_{CNM} + P_{EL})} \tag{2.7}$$

En esta ecuación VAR(P_{EXP}) indica cuanto varia la información explotable en las trazas de consumo y VAR(P_{CNM}+ P_{EL}) indica cuanto varia el ruido en dichas trazas. La obtención fraudulenta de la clave será más fácil si el valor de la relación señal ruido es grande.

La integridad de la señal capturada es críticamente importante para este tipo de análisis, siendo necesario asegurar esta integridad protegiendo el canal de medición. Si esto no fuese posible, es conveniente capturar múltiples trazas del consumo mientras el dispositivo procesa la misma información. De este modo podrá reducirse el ruido calculando la media aritmética de todas ellas.

2.7. Ataque estadístico de correlación.

El análisis de correlación es un ataque DPA que permite determinar la relación lineal entre series de datos diferentes. Es una herramienta potente cuando se pretende realizar un ataque por análisis de consumo [Li'2008]. El ataque consiste en calcular el coeficiente de correlación del modelo, al que llamaremos M, y la muestra de valores coincidentes en el tiempo, obtenida en la captura del consumo, a la que llamaremos C.

2.7.1. Coeficiente de correlación lineal

El coeficiente de correlación se calcula dividiendo la covarianza (entre el modelo y la muestra) y la raíz cuadrada del producto de las varianzas de cada una de ellas:

$$r(M,C) = \frac{COV(M,C)}{\sqrt{VAR(M) \cdot VAR(C)}} \tag{2.8}$$

Se puede demostrar que el coeficiente de correlación es independiente de la escala en que esté expresada cada una de las muestras. Supongamos que cada una de las muestras de valores está multiplicada por un factor de escala tal que:

$$M' = a \cdot M \qquad C' = b \cdot C \tag{2.9}$$

Se cumple que:

$$\begin{aligned} r(M',C') &= \frac{COV(M',C')}{\sqrt{VAR(M') \cdot VAR(C')}} = \frac{COV(a \cdot M, b \cdot C)}{\sqrt{VAR(a \cdot M) \cdot VAR(b \cdot C)}} \\ \frac{a \cdot b \cdot COV(M,C)}{\sqrt{a^2 \cdot VAR(M) \cdot b^2 \cdot VAR(C)}} &= \frac{COV(M,C)}{\sqrt{VAR(M)VAR(C)}} = r(M,C) \end{aligned} \tag{2.10}$$

De esta forma para el cálculo de la correlación puede usarse el modelo HW o HD, que contendrá números enteros positivos, y las trazas tomadas por un sistema de captura, que contendrá valores procedentes de un convertidor A/D, sin necesidad de adaptar las escalas de cada uno de ellos.

La obtención de una correlación de 1 indica que las dos magnitudes están directamente relacionadas, un valor de 0 indica que no están relacionadas y un valor de -1 indica que la relación es inversa (Figura 2.6).

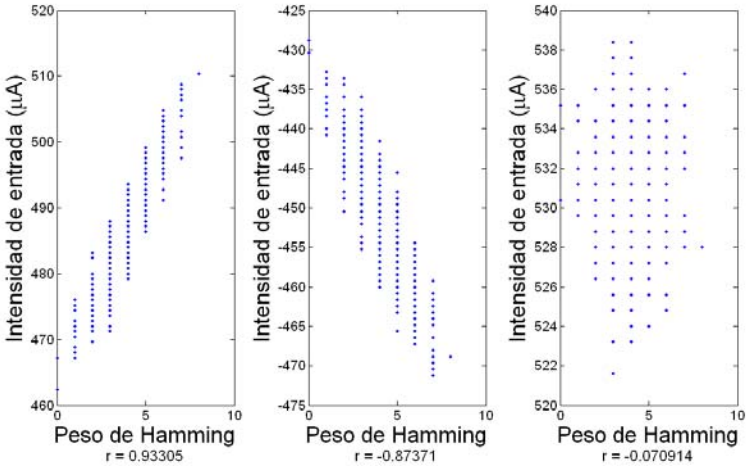

Figura 2.6: Correlación entre magnitudes a) directa b) inversa c) sin correlación

2.7.2. Influencia del ruido en la correlación lineal

Cuando se utiliza la ecuación (2.8) en el análisis de correlación lineal las magnitudes que intervienen en el proceso son: el consumo medido y el modelo de consumo calculado. Otro aspecto a considerar es que el consumo está formado por una componente explotable y una componente de ruido (Apartado 2.6). De las ecuaciones (2.8) y (2.2) se obtiene:

$$r(M,C) = \frac{COV(M,\ P_{EXP}\ +\ P_{RUIDO})}{\sqrt{VAR(M)\cdot VAR(P_{TOT})}} = \\ = \frac{COV(M,\ P_{EXP}\ ) + COV(M, P_{RUIDO})}{\sqrt{VAR(M)\cdot VAR(P_{TOT})}} \tag{2.11}$$

puesto que el ruido no está relacionado con el dato procesado su covarianza con el modelo es nula, por tanto:

$$r(M,C) = \frac{COV(M,\ P_{EXP}\ )}{\sqrt{VAR(M)\cdot VAR(P_{TOT})}} \tag{2.12}$$

Añadiendo el término $\frac{VAR(P_{EXP})}{VAR(P_{EXP})}$ al denominador de la ecuación (2.12) ésta se puede escribir como:

$$r(M,C) = \frac{COV(M,\ P_{EXP}\ )}{\sqrt{VAR(M)\cdot VAR(P_{EXP})\frac{VAR(P_{TOT})}{VAR(P_{EXP})}}} = \\ = r(M,\ P_{EXP})\sqrt{\frac{VAR(P_{EXP})}{VAR(P_{TOT})}} \tag{2.13}$$

Expresión de la que se obtiene que:

$$r(M,C) = r(M,\ P_{EXP})\sqrt{\frac{SNR}{1+SNR}} \tag{2.14}$$

De (2.14) se deduce que la correlación máxima obtenida depende de la correlación máxima que puede darnos el modelo de consumo y de la relación señal ruido de las medidas realizadas. De esta forma se cumple que con una SNR=0 no se obtendrá correlación alguna mientras que de una SNR=∞ la correlación será la máxima posible.

Hay que considerar que la SNR influye tanto en la correlación que se obtiene de la hipótesis correcta como en la correlación de todas las demás claves, de este modo, a pesar del ruido, es posible diferenciar la clave correcta de las que no lo son.

	Bits de ruido				
	0	**24**	**56**	**120**	**240**
Correlación máxima	1	0.4994	0.3401	0.2527	0.1712
Correlación media	9.1 e-05	6.7 e-05	4.7 e-05	3.4 e-05	2.2e-05
Relación	10989	7454	7263	7432	7781

Tabla 2.2: Relación entre correlaciones máximas y medias.

La Tabla 2.2 muestra cómo, a pesar del ruido de conmutación, la correlación de la clave correcta es del orden de 7000 veces mayor a la media de todas correlaciones de todas las hipótesis planteadas.

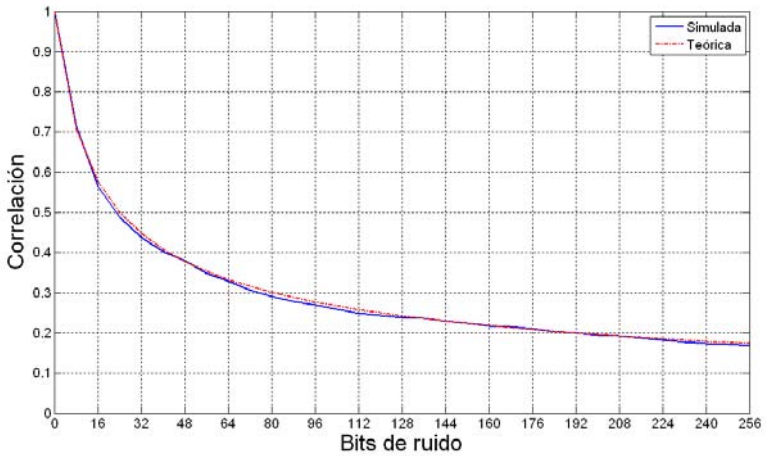

Figura 2.7: Influencia del ruido de conmutación en la correlación máxima modelo-consumo. En trazos rojos se muestra el cálculo teórico y en azul el resultado de la simulación.

La Figura 2.7 muestra la forma en que la correlación máxima se reduce a medida que aumentan los bits que conmutan en un circuito, que no están

relacionados con el dato procesado y que son ruido de conmutación. En esta figura se ha simulado la captura de 1000 trazas de corriente y se muestra la correlación obtenida para la hipótesis correcta.

2.7.3. Ataque de primer orden.

El ataque de primer orden explota la información obtenida mediante la medición del consumo cuando se ejecuta un punto determinado del algoritmo [Wan'2012]. El dato que está siendo procesado en dicho punto debe depender de una pequeña parte de la clave.

En el caso del algoritmo AES-128 [NIST'2001], uno de los puntos habituales de ataque es la salida de la función de sustitución SBOX en primera ronda. En este punto el dato procesado depende única y exclusivamente de un byte de la clave y, si la clave fuese conocida, seriamos capaces de determinar el valor de dicho dato.

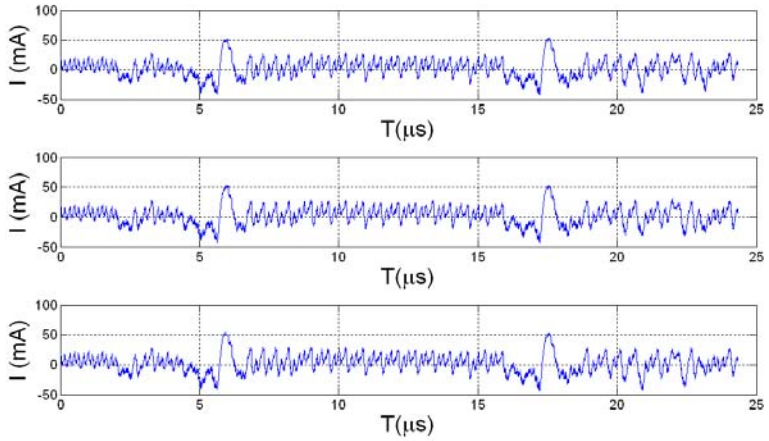

Figura 2.8: Trazas capturadas de la ejecución del bloque SBOX en tres ciclos de encriptado. AES-128 implementado en MicroBlaze[24].

Por un lado, para poder realizar este ataque deben enviarse al sistema criptográfico T textos planos, conocidos y diferentes, para medir el consumo de cada una de las T operaciones de encriptado realizadas. Cada una de las trazas estará formada por N valores adquiridos por el convertidor A/D del sistema de captura (Figura 2.8). Tras la obtención de datos se dispone de una matriz de dimensión [T_{filas} x $N_{columnas}$], donde cada fila repre-

[24] MicroBlaze™ XILINX FPGA-based soft processor.

senta la traza capturada por cada encriptado y cada columna indica el consumo en el mismo instante de cada una de las T capturas (Figura 2.9). En uno de estas N columnas se encuentra la información relativa al consumo producido en el punto de ataque elegido.

Por otro lado, debe calcularse el modelo M de consumo planteando las 256 posibles combinaciones de la clave sobre los T textos planos enviados. Se obtiene una matriz M de dimensión [T_{filas} x $256_{columnas}$] en la que cada columna corresponde con una hipótesis de la clave (Figura 2.9).

Matriz de capturas (Consumo)

$$\begin{pmatrix} c_{1,1} & c_{1,2} & c_{1,3} & .. & c_{1,n} \\ c_{2,1} & c_{2,2} & c_{2,3} & .. & c_{2,n} \\ c_{3,1} & c_{3,2} & c_{3,3} & .. & c_{3,n} \\ .. & .. & .. & .. & .. \\ c_{t,1} & c_{t,2} & c_{t,3} & .. & c_{t,n} \end{pmatrix}$$

Matriz de hipótesis (Modelo)

$$\begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} & .. & m_{1,256} \\ m_{2,1} & m_{2,2} & m_{2,3} & .. & m_{2,256} \\ m_{3,1} & m_{3,2} & m_{3,3} & .. & m_{3,256} \\ .. & .. & .. & .. & .. \\ m_{t,1} & m_{t,2} & m_{t,3} & .. & m_{t,256} \end{pmatrix}$$

Figura 2.9: Matrices de datos para calcular el coeficiente de correlación.

Finalmente, se calcula la correlación existente entre cada una de las columnas de la matriz del modelo y todas las columnas de la matriz del consumo. Será necesario realizar N x 256 cálculos de correlación de los cuales el que dé cómo resultado una mayor correlación corresponderá a la hipótesis de clave planteada en el modelo.

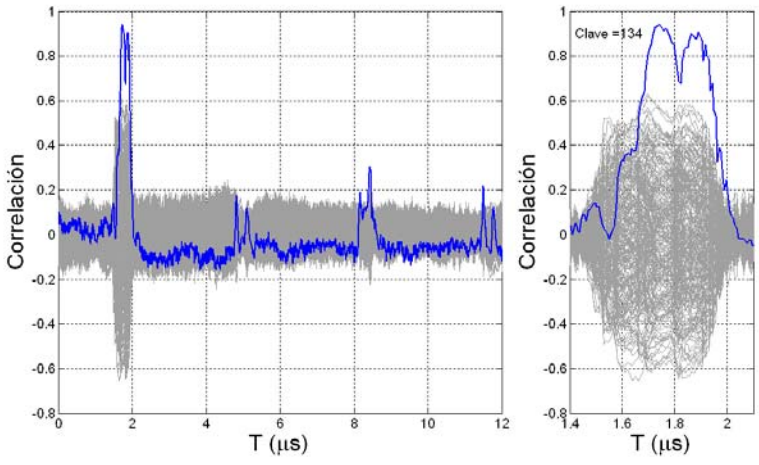

Figura 2.10: Resultado del ataque de correlación de primer orden sobre algoritmo AES

La Figura 2.10 muestra el resultado de realizar un ataque de primer orden sobre el algoritmo AES-128 en el que se obtiene el primer byte de la clave. Nótese que la correlación para la hipótesis correcta alcanza 0.94 mientras que la siguiente en valor absoluto no supera el 0.64, lo que permite diferenciar la hipótesis correcta de la que no lo es.

2.8. Ataque por diferencia de medias.

Mediante este tipo de ataque DPA se pretende observar la ocurrencia o no de un evento determinado que tenga efecto sobre el consumo del dispositivo [Man'2007]. Como puede apreciarse en la Tabla 2.1, en una celda CMOS el consumo asociado a la transición ascendente es diferente al consumo asociado a la transición descendente, por tanto, el evento a observar será el estado de un determinado bit del valor intermedio elegido para el ataque. La Figura 2.11 muestra el consumo asociado al proceso de dos datos cuyo valor difiere en 1 bit, en algunos puntos de la traza la diferencia entre ellas alcanza 1.5 mA.

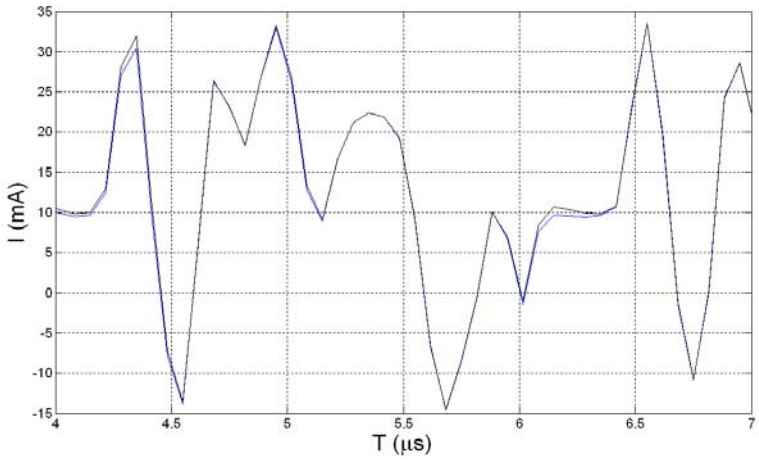

Figura 2.11: Traza de consumo de transición ascendente (azul) y descendente (negro).

2.8.1. Fundamento teórico.

El modelo que se usa en este ataque es el mismo que se usa en el ataque por correlación, no obstante el modelo sólo pretende calcular el valor que se procesará en función de la hipótesis de clave planteada. Por tanto, en la matriz de capturas (Figura 2.9) existe una columna n cuyos elementos están formados por:

$$\begin{aligned} C_{T0} &= C_{CC} \\ C_{T1} &= C_{CC} + C_{CL} \end{aligned} \tag{2.15}$$

C_{T0}: Consumo asociado a la transición descendente.

C_{T1}: Consumo asociado a la transición ascendente.

C_{CC}: Consumo debido al cortocircuito.

C_{CL}: Consumo debido a la carga del condensador.

Puesto que se calcula la media, los valores constantes del consumo no se ven afectados y, al realizar la diferencia, se anularán. Por otro lado, según la ecuación (2.15), puede decirse que:

$$\begin{aligned} C_{T0} &= C \\ C_{T1} &= C + d \end{aligned} \tag{2.16}$$

C: Consumo constante.

d: Consumo diferencial.

Consideremos que N es la cantidad de puntos de la columna de la matriz de capturas (Figura 2.9) que recoge el consumo en el punto de ataque. Estos puntos se han agrupado en dos subconjuntos de elementos N_1 y N_2 tales que:

$$N = N_1 + N_2 \tag{2.17}$$

Estos subconjuntos están formados, a su vez, por otros dos subconjuntos de número de elementos N_{1T1}, N_{2T1} y N_{1T0}, N_{2T0}; que representan el número de transiciones ascendentes y el número de transiciones descendentes respectivamente en cada uno de los subconjuntos N_1 y N_2. Estos subconjuntos verifican que:

$$\begin{aligned} N_1 &= N_{1T1} + N_{1T0} \\ N_2 &= N_{2T1} + N_{2T0} \end{aligned} \tag{2.18}$$

El valor medio M_1 y M_2 de cada uno de los subconjuntos N_1 y N_2 se calcula:

$$\begin{aligned} \mathrm{M}_1 &= \frac{N_{1T0}C + N_{1T1}(C+d)}{N_1} = \frac{(N_{1T0}+N_{1T1})\cdot C + N_{1T1}\cdot d}{N_1} = C + \frac{N_{1T1}\cdot d}{N_1} \\ \mathrm{M}_2 &= \frac{N_{2T0}C + N_{2T1}(C+d)}{N_2} = \frac{(N_{2T0}+N_{1T1})\cdot C + N_{2T1}\cdot d}{N_2} = C + \frac{N_{2T1}\cdot d}{N_2} \end{aligned} \tag{2.19}$$

Finalmente, la diferencia absoluta de estos valores medios se calcula con:

$$|\mathrm{M}_1 - \mathrm{M}_2| = \left|\frac{N_{1T1}\cdot d}{N_1} - \frac{N_{2T1}\cdot d}{N_2}\right| \tag{2.20}$$

Si se verifica que el subconjunto N_1 agrupa todas las transiciones ascendentes y, por el contrario, el subconjunto N_2 agrupa las descendentes se cumple que el resultado de la ecuación (2.20) es un valor máximo según la expresión:

$$|\mathrm{M}_1 - \mathrm{M}_2| = \left|\frac{N_1\cdot d}{N_1} - \frac{0\cdot d}{N_2}\right| = d \tag{2.21}$$

2.8.2. Realización práctica

Este tipo de ataque sigue el esquema expuesto en el apartado 2.5 usando en este caso el modelo de consumo orientado a un bit.

Tras el cálculo del valor intermedio se construye el modelo. De cada una de las columnas de la matriz de valores intermedios se obtienen 8 columnas de la matriz del modelo, éstas corresponden a los 8 bits que forman el valor intermedio calculado. Hay que remarcar que la secuencia de ceros y unos de estas columnas dependen del texto enviado y de la hipótesis de clave planteada.

Seguidamente, por cada columna se separan en un grupo aquellas trazas que, en el modelo, cuenten con un valor lógico “0” y en otro grupo las que cuenten con el valor lógico “1”. Posteriormente se calcula el valor medio de todas estas trazas y se calcula la diferencia entre ellos.

Concluido el proceso se dispone de 8 diferencias de medias correspondientes a los 8 bits de la hipótesis de la clave que se está estudiando. Para obtener un valor asociado a la hipótesis de clave estudiada se realiza la media de las diferencias.

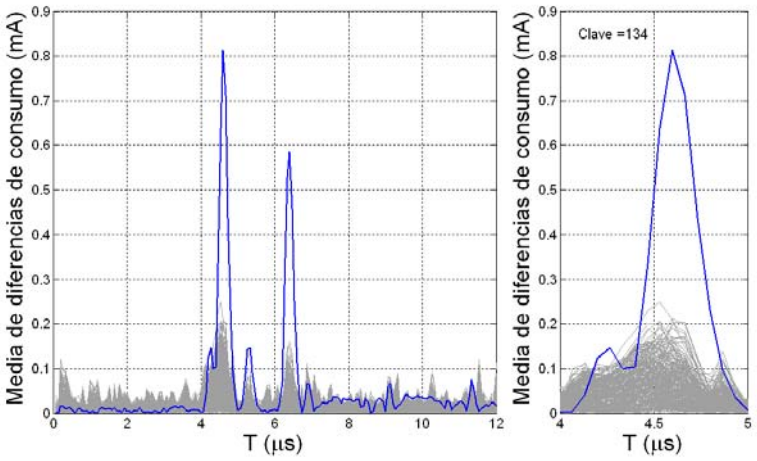

Figura 2.12: Resultado del ataque por diferencia de medias sobre algoritmo AES

La única hipótesis que satisfará la condición que desemboca en la ecuación (2.21) será la que corresponda a la clave correcta. De este modo, la media de diferencias con mayor valor absoluto dejará al descubierto dicha clave (Figura 2.12).

2.9. Medidas de protección contra DPA.

La totalidad de las medidas de protección, o contramedidas, pretenden que el consumo del dispositivo no pueda correlacionarse con el dato procesado. Pueden ser agrupadas según los siguientes enfoques:

Según la arquitectura:

a) **En algoritmo:** Se modifica el algoritmo para que el consumo sea independiente del dato. Las modificaciones se realizan durante la programación eligiendo el flujo de programa que menos información proporcione al atacante.

b) **En hardware:** Se implementa el algoritmo sobre un hardware que no permita la distinción del dato en función del consumo. Para ello, el consumo de las celdas CMOS o puertas lógicas que lo forman han de tener un consumo independiente del dato.

Según el consumo:

a) **Consumo aleatorio:** El consumo debe ser aleatorio en cada ciclo de reloj y por tanto no es posible correlacionarlo con el dato.

b) **Consumo constante:** El dispositivo siempre consume la misma cantidad de energía sea cual sea el dato procesado en cada ciclo de reloj.

Según la magnitud:

a) **Temporal:** Se altera el orden y/o el instante en que se realizan las distintas operaciones que forman el algoritmo.

b) **Amplitud:** Se altera el consumo del dispositivo para ocultar el consumo relacionado con el dato.

2.9.1. Ocultación (Hiding)

Esta contramedida se basa en ocultar al atacante la información explotable. Puede realizarse de varias maneras, tal y como se describe a continuación.

2.9.1.1. Contramedidas en el eje temporal

El ataque de correlación estadística necesita conocer el instante exacto en el que se procesa el dato modelado [Gun'2011]. Si este punto no se conoce, basta con que las trazas estén correctamente alineadas, y realizar una búsqueda exhaustiva a lo largo de toda la traza tal y como se explica en el apartado 2.7. Si se consigue desalinear las trazas el coeficiente de correlación entre el dato y el consumo disminuye y mejora la protección del sistema. Para conseguir el efecto deseado existen diferentes estrategias que pueden implementarse tanto en el algoritmo como en el hardware y consisten en lo siguiente:

a) **Inserción de operaciones sin utilidad específica:**

De forma aleatoria se introducen procesos sin una misión específica y cuya duración también es aleatoria. Se entiende que las trazas capturadas

por el atacante no estarán alineadas y la correlación del modelo con el consumo se verá dificultada.

Puede afirmarse que cuantas más operaciones de este tipo se introduzcan mejor será la protección obtenida pero más lenta será la ejecución del algoritmo.

b) **Cambio del orden de las operaciones:**

Si el algoritmo lo permite, es posible invertir de forma aleatoria aquellas operaciones cuyo orden de ejecución no altere el resultado del proceso. En el caso del AES [NIST'2001] es habitual invertir la ejecución del bloque *"SubBytes"* y del bloque *"ShiftRows"*.

También puede alterarse el orden en que se procesan los diferentes bytes del texto plano.

c) **Eliminar ciclos de reloj:**

Se implementa en hardware y consiste en filtrar la señal de reloj general del sistema para eliminar de forma aleatoria algunos ciclos de reloj, (proceso des sincronizado) [Gun'2011].

d) **Cambio aleatorio de frecuencia:**

También se implementa en hardware y se trata de modificar la frecuencia de reloj del sistema mediante un oscilador basado en números aleatorios.

e) **Múltiples dominios de reloj:**

En el hardware del dispositivo se generan diversas señales de reloj y se asignan de forma aleatoria a las diversas partes del algoritmo.

El no alineamiento provoca que en la matriz de consumo (Figura 2.9) no todos los elementos de la columna pertenezcan al mismo instante en el proceso de encriptado. Se podría decir que la columna que contiene los datos de consumo dependientes de la clave C_K tiene sus puntos agrupados en dos subconjuntos:

a) en el primero están los n elementos que sí están relacionados con el dato procesado y por tanto coinciden con el modelo y

b) en el segundo se encuentran los m elementos que no lo están, y por tanto no coinciden con el modelo.

La suma de n y m coincide con t, que es el número de puntos totales de la columna.

$$C_K = \left[[c_{k,1} \quad .. \quad c_{k,n}][c_{k,1} \quad .. \quad c_{k,m}]\right] \tag{2.22}$$

La probabilidad de que un punto del modelo coincida con el consumo es:

$$P_n = \frac{n}{t} \qquad P_m = \frac{m}{t} \tag{2.23}$$

con:

P_n: Probabilidad de que exista relación consumo-modelo.

P_m: Probabilidad de que no exista relación consumo-modelo

La covarianza entre consumo y modelo se modifica de la siguiente manera:

$$Cov(M, C_k) = P_n \cdot Cov(M, C_n) + P_m \cdot Cov(M, C_m) \tag{2.24}$$

Atendiendo a que los m puntos no correlacionados mostrarán una covarianza con el modelo prácticamente nula:

$$Cov(M, C_k) = P_n \cdot Cov(M, C_n) + 0 \tag{2.25}$$

Por extensión el coeficiente de correlación es:

$$r = \frac{P_n \cdot Cov(M, C_n)}{\sqrt{VAR(M)VAR(C_n)}} = P_n \cdot r(H, C_n) \tag{2.26}$$

El no alineamiento reduce la correlación, tanto en cuanto más variación exista en el eje temporal, pero no llega a eliminarla. En un cálculo

aproximado si de 1000 trazas tomadas 250 coinciden con el modelo la correlación se verá reducida en un 25%.

Estas contramedidas no representan un nivel de seguridad elevado ya que mediante una exploración simple de varias trazas puede deducirse que éstas no están correctamente alineadas e identificar que se está utilizando esta técnica de protección. También es posible, mediante el pre-procesado matemático de las trazas, realinearlas para realizar el ataque de correlación con posterioridad [Tia'2012].

2.9.1.2. Contramedidas en el eje de amplitud

Únicamente un dispositivo que sea capaz de consumir siempre la misma cantidad de energía en cada ciclo de reloj, o que su consumo sea aleatorio es inmune al ataque de correlación [Man'2007].

Basándose en esta premisa se plantean las siguientes estrategias de protección:

En algoritmo: Es necesario seleccionar aquellas instrucciones de programación que menos información puedan dar al atacante y seleccionar un flujo de programa que no dependa del valor de la clave. El direccionamiento de memoria también debe ser independiente de la clave, esta situación se hace relevante en búsquedas dentro de tablas, pues el bus de direcciones también está sometido a perdidas de información explotable.

Otro enfoque puede ser la generación de actividad paralela utilizando coprocesadores, contadores, temporizadores, generadores PWM o interfaces de comunicación que puedan mantener actividad de forma concurrente al núcleo lógico del sistema.

En Hardware: Es posible utilizar el filtrado del consumo para reducir el transitorio provocado por las transiciones o introducir en la medida ruido aleatorio mediante generadores conectados a redes de capacidad grande y con un fuerte impacto en el consumo total.

Otra tendencia en este ámbito es el uso de hardware cuyas celdas estén diseñadas de forma que su consumo sea independiente del dato que procesan. Se intenta que la cantidad de energía consumida sea la misma en to-

dos los ciclos de reloj. Para ello, es necesario un dispositivo que siempre presente la misma cantidad de transiciones sea cual sea el dato procesado. La base es crear dispositivos que procesen un dato y, paralelamente, su complementario (Tabla 2.3)

Dato procesado	**Dato complementario**	**HW_{total}**
10100101	01011010	8
11101111	00010000	8

Tabla 2.3: Datos de 8 bits procesados con su complementario para HW.

Si bien el modelo es inmune a la correlación con el peso de Hamming, no lo es con la distancia de Hamming pues ésta será diferente en función de cuales sean los datos procesados (Tabla 2.4)

Orden	**Dato procesado**	**Dato complementario**	**$HD_{(n,n-1)}$**
1	10100101	01011010	--
2	11101111	00010000	6
3	11111111	00000000	2

Tabla 2.4: Datos de 8 bits procesados con su complementario para HD.

La solución adoptada consiste en realizar las operaciones en dos etapas [Mok'2012] [Sok'2005]:

a) **Precarga:** el estado de todos los bits, tanto de los datos como de su complementario, se llevan a un valor constante conocido, por ejemplo todos ellos a nivel lógico bajo.

b) **Evaluación:** desde el estado de precarga se evoluciona hasta el resultado del procesado, de esta forma la distancia de Hamming siempre es la misma y por tanto el consumo no puede ser correlacionado.

Esta idea puede ser implementada, al nivel de abstracción de transistores, para crear celdas específicas que puedan ser usadas para la implementación de circuitos ASIC para criptografía [Yue'2009] [Sok'2005] [Buc'2011], lo que se conoce como DRP logic[25].

[25] Dual Rail Prechargue Logic.

En [Tir'2004] Tiri y Verbauwhede introdujeron una metodología para la creación de sistemas Dual-Rail construidas a partir de celdas CMOS y puertas lógicas estándares a las que se denomino WDDL[26]. La estructura de las celdas WDDL[27] son mucho más sencillas que las celdas DRP lo que permite reducir significativamente el tamaño de los circuitos desarrollados.

Dual Rail: Todas las señales se codifican en dos cables físicos. En uno de ellos se codifica la señal propiamente dicha y en el otro se codifica su complementaria. Los niveles lógicos se establecen según la Tabla 2.5.

Nivel lógico	**Codificación (real – complementaria)**
H	1 - 0
L	0 - 1

Tabla 2.5: Estados en codificación WDDL

De esta forma siempre que se produce una transición H→L o L→H, físicamente se están produciendo dos transiciones una en la señal real y otra, de signo contrario, en la señal complementaria.

En diseño de puertas lógicas una puerta AND con tecnología WDDL se puede implementar tal y como se muestra en la Figura 2.13:

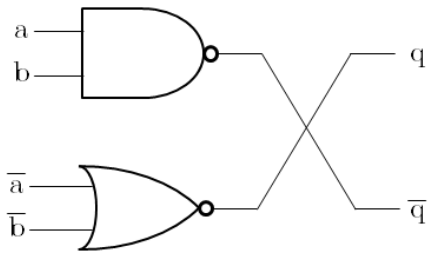

Figura 2.13: Puerta AND con tecnología WDDL

Siendo las transiciones posibles las que se muestran en la Tabla 2.6.

Como puede apreciarse el consumo no es idéntico en todos los casos pues existen dos de ellos en que no se producen transiciones mientras que en los otros dos sí se producen. El consumo continúa siendo dependiente del dato.

[26] Wave Dynamic Differential Logic.

[27] Número de patente: US 20130120024 A1 del 16 de mayo de 2013.

	Transiciones		
Transición a la salida	q	$\overline{q}$	**Total**
L→L	0→0	1→1	0
L→H	0→1	1→0	2
H→L	1→0	0→1	2
H→H	1→1	0→0	0

Tabla 2.6: Transiciones en Dual-Rail Logic

Dual-Rail con precarga: de igual manera todas las señales se codifican mediante dos cables físicos, pero en este caso el dispositivo funciona en dos etapas (Figura 2.14). En una primera los valores de todas las señales se lleva a un valor fijo y conocido y en la segunda etapa las señales se evalúan y se llevan al valor lógico resultante de la operación lógica que está realizando. Una señal sincronizada con el reloj del sistema es quien determina si se está en la etapa de precarga o en la de evaluación.

La transición a la salida evoluciona del estado de precarga al estado de evolución según la siguiente tabla [Mok'2012]:

Fase precarga (Pch→L)		**Fase evaluación** (Pch→H)		**Tran.**
q	$\overline{q}$	q	$\overline{q}$	**Total**
0	0	0	1	1
0	0	1	0	1

Tabla 2.7: Transiciones en Dual-Rail Precharge Logic

De este modo siempre se produce una única transición sea cual sea la evolución de los valores.

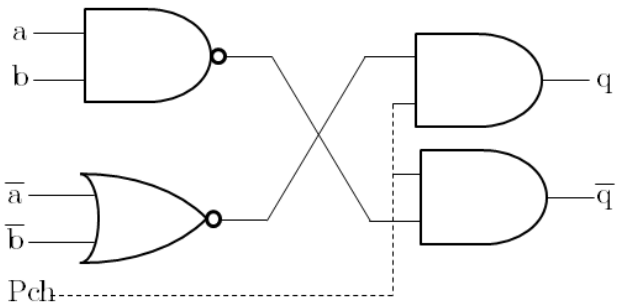

Figura 2.14: Puerta AND con tecnología WDDL y precarga

Los modelos de consumo no pueden aplicarse debido a que siempre se producen el mismo número de transiciones. El problema se presenta dado

que las capacidades asociadas a las líneas de transmisión (Figura 2.3) no son iguales y por tanto los consumos asociados a los transitorios en las señales “q” y “not q” tampoco lo serán. El ruteado que se haga de las líneas de conexión dentro del dispositivo debe ser especialmente elegido para minimizar este efecto [Zha'2011]. En esta línea de investigación [He'2014] presenta nuevas herramientas en el flujo de diseño genérico de los dispositivos FPGA para conseguir un ruteado simétrico y equilibrar el consumo.

El uso de esta tecnología reduce considerablemente la información explotable por el atacante, pero no la elimina totalmente como sería deseable [Man'2007]. Del mismo modo el consumo es sensible a la aparición de efectos tipo “*Glitch*“ o “*Early Propagation*” [Kul'2004] [Dai'2006]. Para evitarlo, en la medida de lo posible, [Tir’2004] [Buk’2011] proponen la inclusión de biestables en las puertas que tratan de sincronizar al máximo la evolución de las señales dentro de la lógica combinacional.

El mayor hándicap de esta tecnología es que incrementa considerablemente el área necesaria para el desarrollo de cualquier procesador criptográfico, debido a que cada celda requiere mayor número de transistores para su implementación que la lógica tradicional [Hwa'2006]. Este hecho provoca que el consumo del dispositivo sea mayor y que su desarrollo sea más costoso.

2.9.2. Enmascarado (Masking)

El enmascarado es una contramedida que se puede aplicar tanto a nivel de algoritmo como de hardware. Se trata de enmascarar el dato procesado con un valor aleatorio, que se genera internamente y que varía a cada ciclo de encriptado o desencriptado [Cha'1999]. De este modo todos los valores intermedios del proceso están formados por la combinación del dato propio del algoritmo y la máscara aleatoria desconocida por el atacante. Una vez ejecutado el algoritmo se elimina la máscara para tener el resultado correcto de la operación de encriptado o des encriptado.

En el caso del algoritmo AES habitualmente se realiza el enmascarado mediante el uso de la operación booleana or-exclusiva (Figura 2.15), pero tanto este como otros algoritmos pueden ser enmascarados con opera-

ciones algebraicas tales como el producto modular o la suma modular [Bae'2005].

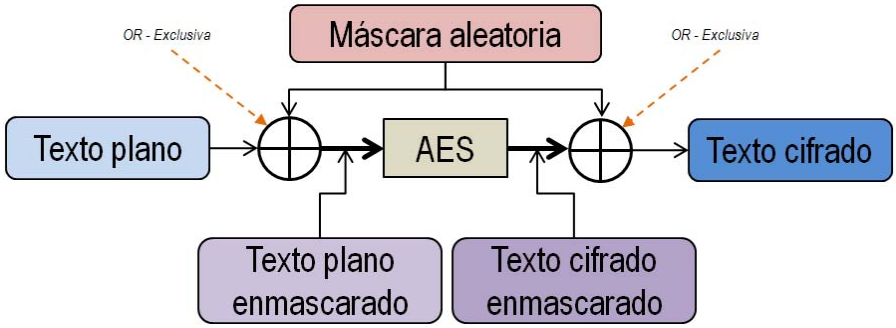

Figura 2.15: Algoritmo AES protegido con enmascarado or-exclusiva.

El algoritmo AES está compuesto por diferentes operaciones [NIST'2001], de ellas las que son lineales cumplen la propiedad:

$$f(x \oplus m) = f(x) \oplus f(m) \tag{2.27}$$

La dificultad se presenta con la operación "*SubByttes*" que es altamente no lineal y por tanto no cumple la propiedad anterior. Esta operación suele realizarse buscando en una tabla, denominada SBOX, el valor de salida correspondiente al valor de entrada. Para poder realizar el enmascarado es necesario modificar la función SBOX y convertirla en una SBOX' según las siguientes operaciones (expresadas en pseudocódigo):

$$\begin{aligned} &for\ x = 0\ to\ 255 \\ &\quad SBOX'(x \oplus m) = SBOX(x) \oplus m \\ &next\ x \end{aligned} \tag{2.28}$$

con:

x: x= 0..255

m: máscara aleatoria generada internamente.

Considerando que la máscara es aleatoria y además se cambia cada ciclo de encriptado o desencriptado, será necesario recalcular la SBOX' cada vez que cambie dicha máscara.

En operaciones más complejas como la "*MixColumn*" es necesario ser extremadamente cuidadoso a la hora de su implementación. Esta operación combina diversos valores de los que están siendo procesados para calcular los siguientes. De este modo, si se realizan operaciones or-exclusiva sobre dos valores afectados por la misma máscara esta última desaparece y el valor intermedio queda desprotegido.

$$\begin{aligned} &x' = x \oplus m; \quad y' = y \oplus m \\ &x' \oplus y' = x \oplus m \oplus y \oplus m = x \oplus y \oplus m \oplus m \\ &x \oplus y \oplus m \oplus m = x \oplus y \oplus 0 = x \oplus y \\ &x' \oplus y' = x \oplus y \end{aligned} \tag{2.29}$$

Por consiguiente, no se recomienda el uso de una única máscara en el proceso, siendo común el uso de múltiples máscaras dentro del algoritmo [Man'2007] para conseguir que el consumo sea efectivamente aleatorio y no existe correlación con el dato. La inclusión de múltiples máscaras [Wan'2014] aumenta la seguridad del sistema pero implica la necesidad de operaciones de cálculo, previas a la ejecución del algoritmo, y operaciones de reenmascarado, durante la ejecución del mismo, que aumentan el tiempo de ejecución haciéndolo más lento.

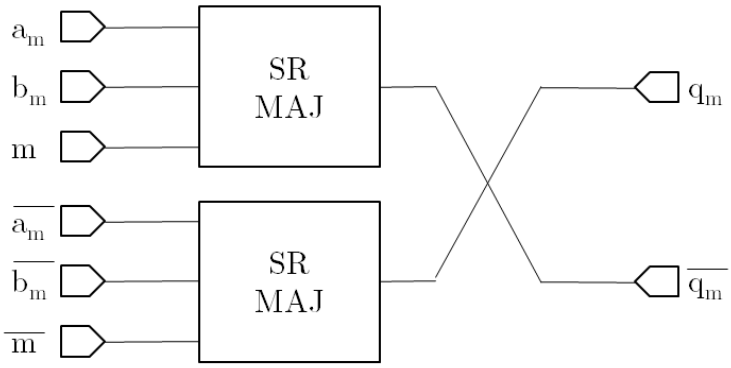

Figura 2.16: Esquema de celda NAND en tecnología MDPL

En hardware, el enmascarado se implementa a nivel de puerta lógica de forma similar a la tecnología WDDL (Apartado 2.9.1.2) pero utilizando puertas lógicas monotónicas, lo que da lugar a la tecnología MDPL[28]. En esta tecnología se procesan valores enmascarados junto con la máscara aleatoria (Figura 2.16). El hecho de utilizar la máscara hace innecesario el

[28] Masked Dual-Rail Precharge Logic

balanceo de las líneas de transmisión para que sus capacidades parásitas sean iguales[29]. El valor de dicha máscara determina en que línea de la estructura "Dual Rail" se realiza la operación siendo el consumo resultante altamente independiente del dato procesado.

a_m	b_m	m	$\overline{a_m}$	$\overline{b_m}$	$\bar{m}$	$q_m = MAJ(\overline{a_m}, \overline{b_m}, \bar{m})$	$\overline{q_m} = MAJ(a_m, b_m, m)$
0	0	0	1	1	1	1	0
0	1	0	1	0	1	1	0
1	0	0	0	1	1	1	0
1	1	0	0	0	1	0	1
0	0	1	1	1	0	1	0
0	1	1	1	0	0	0	1
1	0	1	0	1	0	0	1
1	1	1	0	0	0	0	1

Tabla 2.8: Tabla de verdad de puerta NAND monotónica con tecnología MPDL

Por otro lado, el hecho de usar puertas monotónicas aporta al sistema independencia de errores tipo "*glitch*". En la Tabla 2.8 se muestra la tabla de verdad de la puerta NAND expuesta en la Figura 2.16.

2.10. Ataque de segundo orden

Los ataques por canal lateral de segundo orden consisten en tomar dos puntos de la traza de corriente y procesarlos, de tal forma que al resultado de dicho procesado pueda aplicársele el procedimiento descrito en el apartado 2.7.2. Por tanto, antes de aplicar este procedimiento es necesario determinar el tipo de procesado a aplicar, los puntos a procesar y el modelo de consumo a utilizar.

Sean $\mathbf{u}_\mathbf{m}$ y $\mathbf{v}_\mathbf{m}$ los valores enmascarados asociados a los dos puntos intermedios que desean procesarse. Sus respectivos valores sin enmascarar se denotan por $\mathbf{u}$ y $\mathbf{v}$, de modo que:

$$u_m = u \oplus m \quad y \quad v_m = v \oplus m \tag{2.30}$$

[29] En la práctica "lo más parecidas posibles".

Considerando válido el modelo de consumo basado en la altura de Hamming (HW), y dado que los datos han sido enmascarados con el operador lógico XOR a nivel de bit, se tiene que:

$$Hw(u_m \oplus v_m) = Hw(u \oplus m \oplus v \oplus m) = \ Hw(u \oplus v) \qquad (2.31)$$

Es decir, el consumo teórico para la combinación mediante el operador or-exclusiva de los datos enmascarados y sin enmascarar es coincidente. La expresión (2.31) se toma como modelo de consumo y se utiliza para el cálculo del coeficiente de correlación. Nótese que es necesario disponer de un modelo de consumo basado en el texto plano original (sin enmascarar), dado que ésta es la única información conocida por parte del atacante.

Cuando se ataca el algoritmo AES los puntos elegidos para ello suelen ser la entrada y la salida de la función "*SubBytes*" (modificada) en primera ronda. Una de las razones que justifica esta elección se debe a que el código asociado con ambos puntos se ejecuta prácticamente de forma consecutiva, lo que simplifica el proceso de captura del tramo de interés de la traza de corriente.

A continuación debe tomarse una función que procese los puntos de consumo correspondientes a los dos instantes en que se producen el par de valores (u_m,v_m) y que denominaremos (c_{ti},c_{tk}). Esta función debe poseer un grado de correlación significativo con respecto al modelo de consumo descrito en (2.31). [Mes'2000] propone una función que se basa en calcular el valor absoluto de la diferencia entre los consumos en ambos puntos:

$$F_{procesado} = abs(c_{ti} - c_{tk}) \qquad (2.32)$$

Aunque otros autores [Pro'2009] [Man'2007] proponen utilizar el producto de las muestras, la diferencia, el cuadrado de la suma, o el cuadrado del valor absoluto entre otras.

La función de procesado se aplica sobre cada una de las trazas y el resultado puede ser correlacionado con el modelo para obtener, del mismo modo que se explica en el apartado 2.7.2, la máxima correlación que corresponderá con la hipótesis correcta (Figura 2.17).

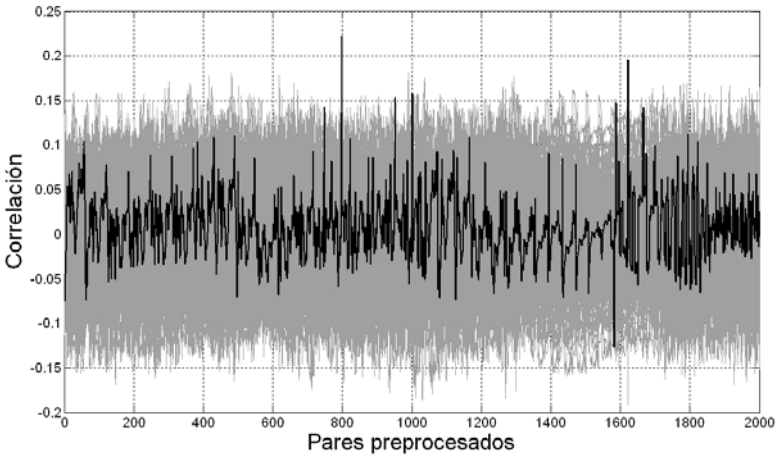

Figura 2.17: Ataque de segundo orden sobre AES en MicroBlaze[30]

El éxito de este proceso requiere tener un conocimiento ajustado del instante en que se producen los puntos (c_{ti},c_{tk}). Cuando estos puntos son desconocidos, el proceso de atacado se aplica sobre todos los pares de valores que surgen de realizar todas las combinaciones posibles con los puntos que componen la traza. Por tanto cada traza procesada se compone de n! valores donde n es el número de puntos de las trazas originales.

2.11. Conclusiones

Desde la propuesta inicial de [Koc'1996] un número significativo de contramedidas para prevenir el ataque por análisis de consumo han sido propuestas y desarrolladas. En todas ellas el costo de aplicarlas se traduce en mayor tiempo de proceso, mayor complejidad de cálculo y/o una mayor área de silicio necesaria. El objetivo de la comunidad científica es reducir el costo de implementar estas contramedidas o encontrar nuevos sistemas de protección que requieran pocos medios adicionales para su funcionamiento.

[30] MicroBlaze™ XILINX FPGA-based soft processor

3. Contramedida basada en "faking"

3.1. Introducción

Las contramedidas propuestas hasta ahora por diversos investigadores se han orientado hacia dos vertientes claramente diferenciadas: ocultar la información explotable o bien enmascararla. Estas contramedidas obligan en muchos casos al uso de estructuras de doble proceso que persiguen un consumo de potencia constante y no dependiente del dato. En la otra línea de trabajo es necesario implementar algoritmos que enmascascaren el dato procesado, para que el consumo no dependa únicamente del dato sino también de la máscara. Ambas tendencias dificultan notablemente la obtención fraudulenta de las claves del sistema, pero en ocasiones el uso de algoritmos de análisis más sofisticados, o el simple hecho de tomar un mayor número de trazas, permite al atacante conseguir su objetivo.

La propuesta que se presenta en esta tesis representa un cambio de paradigma en lo que a la protección de dispositivos criptográficos se refiere. No se pretende conseguir un sistema invulnerable, sino que se trata de aprovechar la propia vulnerabilidad del sistema para conducir al atacante a una conclusión falsa cuando plantee un ataque.

En este capítulo se presenta la base teórica que se ha utilizado para diseñar la contramedida propuesta. Para ello, se describe en primer lugar de

forma esquemática el algoritmo AES en su versión de 128 bits y se estudian sus vulnerabilidades ante ataques de consumo. Seguidamente, se exponen las características de la contramedida basada en “faking” y las modificaciones necesarias para su protección. Finalmente, se exponen las características que debe tener el sistema según si: la implementación de la contramedida es en un sistema puramente software, en un sistema basado en codiseño software-hardware y finalmente en un sistema puramente hardware.

3.2. Marco teórico

Actualmente uno de los sistemas más utilizados para la obtención de claves criptográficas es el análisis por correlación (Apartado 2.7). La correlación se calcula entre un grupo de T trazas, medidas mientras el dispositivo ejecuta el algoritmo con diferentes textos planos, y M modelos teóricos de dicho consumo. Cada uno de estos M modelos corresponde a las M posibilidades de la clave que se pretende obtener. Para la construcción del modelo el atacante debe de conocer el texto plano que se envía al dispositivo.

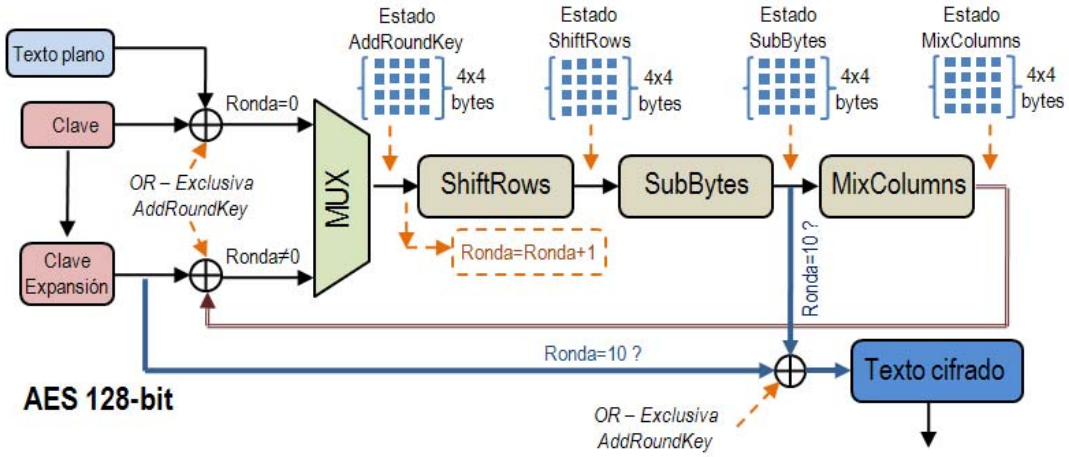

Figura 3.1: Estructura básica del algoritmo AES

Normalmente, para que el ataque sea posible a nivel práctico, éste se centra en el procesado de un único byte de la clave en un determinado instante del algoritmo. El número de posibles valores de la clave en este caso particular disminuye a 2^8=256, lo que reduce la complejidad del proceso

habida cuenta el tiempo necesario para la captura de las trazas de consumo, su procesado y la capacidad de cálculo de los ordenadores actuales.

La Figura 3.1 muestra la estructura básica del algoritmo AES [NIST'2001]. En la versión AES 128-bits el algoritmo se procesa sobre una matriz de 4x4 bytes, que se denomina estado, con una longitud de clave de 16 bytes.

3.2.1. Algoritmo AES

El algoritmo está formado por 4 funcionan básicas (*AddRoundKey, ShiftRows, SubBytes y MixColumns*) que se ejecutan secuencialmente en 11 rondas (0 a 10) según el siguiente esquema:

Ronda 0: *AddRoundKey.*

Rondas 1 a 9*: ShiftRows, SubBytes, MixColumns, AddRoundKey.*

Ronda 10: *ShiftRows, SubBytes, AddRoundKey.*

Las funciones realizan las siguientes operaciones:

AddRoundKey: Consiste en una operación lógica "or exclusiva" entre el estado y la clave. Cada byte del estado se opera con un byte de la clave correspondiente a la ronda que se está calculando.

ShiftRows: Se resuelve con un desplazamiento de los bytes que forman las filas del estado.

SubBytes: Se realiza la sustitución SBOX definida en el algoritmo AES. La forma más habitual de realizar esta operación es hacer una búsqueda en una tabla almacenada en la memoria del dispositivo.

MixColumns: Cada columna se trata como un polinomio en el campo finito de Galois[31] $GF(2^8)$, se realiza el producto modular (módulo x^4+1) con el polinomio $C(x)=3x^3+x^2+x+2$ en caso de encriptado, o con el polinomio $C^{-1}(x)=11x^3+13x^2+9x+14$ en caso de desencriptado.

[31] Evariste Galois, matemático francés 1811-1832.

Esta función representa el primer punto de difusión de la clave pues, a partir de la ronda 1 cada byte del estado resultante depende de 4 bytes de la clave y de 4 bytes del texto plano. La dependencia de la clave aumenta con cada ronda del algoritmo.

A parte de las operaciones ya mencionadas existe una función (*KeyExpand*) que opera únicamente sobre la clave. Ésta es sometida a un proceso de expansión por el cual, partiendo de la clave inicial, se obtienen diversas claves de igual longitud (tantas como rondas). Cada una de estas claves expandidas se usará en las sucesivas rondas del algoritmo. Este proceso favorece la difusión de la clave a lo largo del mismo.

3.2.2. Vulnerabilidades del algoritmo AES

En la Tabla 3.1, se puede apreciar el efecto que tiene sobre los sucesivos valores del estado la variación de un bit de la clave. Para verificarlo se han realizado dos ciclos de encriptado usando el mismo texto plano. En el segundo caso se ha variado únicamente el bit menos significativo del byte 15 de la clave. A causa de ello, puede observarse como todos los valores del estado tras la ejecución de la función *MixColumn* en la ronda 2 han variado. Por consiguiente podemos decir que cualquier valor a partir de este punto dependerá de toda la clave. En la tabla puede apreciarse sombreados aquellos valores que han cambiado del primer al segundo ciclo de encriptado.

Bytes del estado.																	
B0	B1	B2	B3	B4	B5	B6	B7	B7	B8	B10	B11	B12	B13	B14	**B15**		
15	4	83	250	3	140	142	46	132	46	7	109	99	162	137	**103**	AdKey	R0
118	242	237	45	123	100	25	49	95	49	197	60	251	58	167	**133**	Sbytes	
118	100	197	**133**	123	49	167	45	95	58	237	49	251	242	25	60	SRow	
0	**111**	**23**	**42**	47	198	104	65	44	54	247	84	197	19	127	133	MiCol	
220	**255**	**32**	**154**	180	143	**183**	168	187	200	**133**	107	253	146	**106**	**34**	AdKey	R1
134	**22**	**183**	**184**	141	115	**169**	194	234	232	**151**	127	84	79	**2**	**147**	Sbytes	
134	115	**151**	**147**	141	232	**2**	**184**	234	79	**183**	194	84	**22**	**169**	127	SRow	
134	**81**	**110**	**72**	**152**	**248**	**178**	**13**	**107**	**116**	**141**	**66**	**68**	**231**	**138**	**189**	MiCol	
84	**152**	**5**	**255**	**209**	**120**	**6**	**83**	**181**	**10**	**75**	**35**	**162**	**24**	**89**	**123**	AdKey	R2

Tabla 3.1: Dispersión de la clave cuando se varía un bit.

Por consiguiente, la principal vulnerabilidad del algoritmo se encuentra en las rondas 0 y 1. Estas son las rondas en las que la clave no se ha di-

fundido a través del propio estado y por tanto puede ser atacada parcialmente.

En la primera ejecución de la función *AddRoundKey* el estado corresponde con el texto plano y la clave que se usa corresponde con la original sin expandir. La operación que se realiza es la siguiente:

$$Sa_{(i,j)} = T_{(i,j)} \oplus K_{(i,j)} \tag{3.1}$$

siendo:

$Sa_{(i,j)}$: Byte (i,j) del estado tras la función *AddRoundKey*.

$T_{(i,j)}$: Byte (i,j) del texto plano.

$K_{(i,j)}$: Byte (i,j) de la clave.

(i,j): Subíndices de: (filas, columnas) i=1..4, j=1..4

Como puede apreciarse en (3.1) cada byte del estado depende de un byte del texto plano y de un único byte de la clave.

Aunque el ataque dirigido a esta función puede ser efectivo, resulta difícil discriminar la hipótesis correcta ya que, al tratarse de una operación lineal, la correlación obtenida es muy parecida a las correlaciones obtenidas con hipótesis de similar peso de Hamming (Tabla 3.2).

Hipótesis de clave		**Correlación máxima**
209	HW(11010001)=4	0.6282
210	HW(11010010)=4	0.6002
211	HW(11010011)=5	0.6692
196	HW(11000011)=4	0.6545
148	HW(10010011)=4	0.6236

Tabla 3.2: Correlaciones máximas obtenidas con ataque a la función AddRoundKey. Se obtiene la clave correcta (211) capturando 1000 trazas.

En el caso de la función *ShiftRows* los resultados no varían respecto de la anterior función, pues esta función únicamente mueve valores dentro del estado sin realizar ninguna operación con ellos. La salida de esta función se recibe el nombre de "Sr".

Un punto ampliamente usado para el ataque es la salida de la función *SubBytes*. Esta es una función altamente no lineal, de forma que la variación de un bit a la entrada provoca el cambio de varios bits a la salida, lo que permite diferenciar con facilidad la clave real respecto al resto de claves (Tabla 3.3).

Hipótesis de clave		**Correlación máxima**
209	HW(11010001)=4	0.0632
210	HW(11010010)=4	0.0747
211	HW(11010011)=5	0.8143
196	HW(11000011)=4	0.0591
148	HW(10010011)=4	0.0729

Tabla 3.3: Correlaciones máximas obtenidas con ataque a la función SubBytes. Se obtiene la clave correcta (211) capturando 200 trazas.

La sustitución que se realiza en esta función es la siguiente:

$$Sb_{(i,j)} = SBOX\left(Sa_{(i,j)}\right) = SBOX(T_{(i,j)} \oplus K_{(i,j)}) \tag{3.2}$$

donde:

$Sb_{(i,j)}$: Byte (i,j) del estado tras la función *SubBytes*.

Puede comprobarse que los elementos del estado continúan dependiendo únicamente de un byte de la clave y por tanto el número de hipótesis siguen siendo reducidas.

A partir de la ejecución de la función *MixColumns* en la primera ronda cada valor del estado depende de 4 bytes de la clave. Esto supone que la cantidad de claves posibles para calcular el modelo se eleva a 2^{32}. La operación que se realiza en esta función se define mediante (3.3) aplicada a cada una de las 4 columnas del estado:

$$\begin{pmatrix} Sm_{(i,1)} \\ Sm_{(i,2)} \\ Sm_{(i,3)} \\ Sm_{(i,4)} \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} Sb_{(i,1)} \\ Sb_{(i,2)} \\ Sb_{(i,3)} \\ Sb_{(i,4)} \end{pmatrix} \tag{3.3}$$

Siendo:

$Sm_{(i,j)}$: Byte (i,j) del estado tras la función *MixColumns*.

De (3.3) se puede extraer que el cálculo del primer byte del estado se realiza de la siguiente forma:

$$Sm_{(1,1)} = 2 \cdot Sb_{(1,1)} + 3 \cdot Sb_{(1,2)} + Sb_{(1,3)} + Sb_{(1,4)} \quad (3.4)$$

Considerando que el atacante tiene control sobre el texto plano que se envía al dispositivo, [Lu'2009] propone usar un texto plano en el que todos sus bytes sean constantes, a excepción del byte objetivo del ataque. Suponiendo que en la ecuación (3.4) el byte del texto plano que varía sea el $T_{1,4}$ se cumple que:

$$Sm_{(1,1)} = Constante + Sb_{(1,4)} \quad (3.5)$$

Puede apreciarse que (3.5) equivale a un enmascarado pero con una máscara que no varía en el tiempo, el coeficiente de correlación se verá afectado aunque seguirá siendo útil para localizar la clave. En consecuencia, la función *MixColumns* es atacable de modo similar a como se ataca la función *SubBytes.*

Es a partir de la ejecución de la función *MixColumns* en la ronda 2 cuando la difusión de la clave es tal que cualquier byte del estado depende de los 128 bits de la clave [Lu'2009]. Intentar el ataque en estas condiciones equivaldría a realizar un ataque por fuerza bruta.

A parte de las vulnerabilidades expuestas en las primeras rondas del algoritmo existe la posibilidad de atacarlo en sus últimas rondas. El atacante debe conservar la información que retorna el criptoprocesador, tanto si es texto cifrado como texto plano, y plantear las hipótesis sobre la ronda 10. El objetivo es obtener la última parte de la clave extendida que se ha utilizado y, dada la reversibilidad de la función *KeyExpand*, obtener la clave correspondiente.

El punto más sensible mediante esta técnica es al byte inmediatamente anterior a la función *SubBytes* de la ronda 10, aunque [Lu'2009] plantea la posibilidad de atacar cualquier valor intermedio a partir de la ronda 7.

3.3. Fundamentos de la contramedida propuesta.

La idea básica de la contramedida propuesta es realizar el proceso de encriptado mediante el uso de una clave falsa (Key_{FAKE}). Esta clave falsa se obtiene realizando la operación lógica “or exclusiva” de la clave real (Key_{REAL}) con una máscara de clave (Key_{MASK}) de la siguiente manera:

$$Key_{FAKE(i,j)} = Key_{REAL(i,j)} \oplus Key_{MASK(i,j)} \tag{3.6}$$

$Key_{FAKES(i,j)}$: Byte (i,j) de la matriz de la clave falsa.

$Key_{REAL(i,j)}$: Byte (i,j) de la matriz de la clave real.

$Key_{MASK(i,j)}$: Byte (i,j) de la matriz de la máscara de clave.

Las operaciones del algoritmo se realizan utilizando la clave falsa (Key_{FAKE}) y, puesto que el sistema no dispone de ninguna contramedida adicional, cualquier ataque por canal lateral revelará esta clave [Lum’2014].

La Figura 3.2 muestra la estructura que implementa la contramedida basada en “faking”.

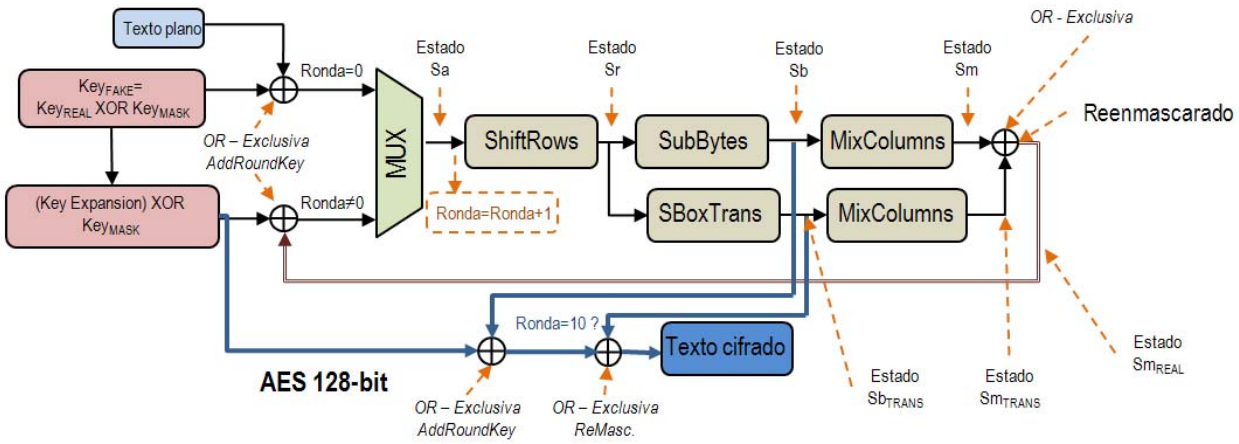

Figura 3.2: Estructura de implementación de AES con la contramedida basada en “faking”.

Se debe tener en cuenta que, sin acciones adicionales, el texto sería encriptado con la clave falsa. En consecuencia se obtendría el texto incorrec-

tamente cifrado. Para que el sistema devuelva los resultados esperados habrá que invertir el proceso en algún punto del algoritmo.

Es a partir de la función *SubBytes* cuando el estado sufre las transformaciones más drásticas. La sustitución SBOX se realiza sobre la salida de la función *ShiftRows.*

$$Sb_{(i,j)} = SBOX\left(Sr_{(i,j)}\right) = SBOX\left(Key_{FAKE(i,j)} \oplus Estado_{(i,j)}\right) \tag{3.7}$$

$Estado_{(i,j)}$: Byte (i,j) de la matriz de estado resultante de la función anterior.

Considerando las expresiones (3.6) y (3.7) se puede observar que el estado resultante corresponde al que se hubiese obtenido usando la clave real pero enmascarado (or exclusiva) con un determinado valor. Estos valores forman una matriz [4x4] denominada Sb_{TRANS}.

$$\begin{aligned} Sb_{(i,j)} &= SBOX\left(Key_{REAL(i,j)} \oplus Key_{MASK(i,j)} \oplus Estado_{(i,j)}\right) \\ Sb_{(i,j)} &= SBOX\left(Key_{REAL(i,j)} \oplus Estado_{(i,j)}\right) \oplus Sb_{TRANS(i,j)} \\ Sb_{(i,j)} &= Sb_{REAL\,(i,j)} \oplus Sb_{TRANS(i,j)} \end{aligned} \tag{3.8}$$

$Sb_{TRANS(i,j)}$: Byte (i,j) de la matriz de la máscaras Sb_{TRANS}.

$Sb_{REAL\,(i,j)}$: Byte (i,j) de la matriz de estado a la salida de la función *SubBytes* si hubiese sido procesado con la clave real.

Finalmente, el resultado de (3.8) se procesa mediante la función M*ix-Columns*. Esta función se realiza en el campo finito de Galois $GF(2^8)$ e incluye las operaciones suma y producto. Una de las propiedades de este tipo de estructura algebraica es la propiedad distributiva del producto respecto de la suma, de forma que:

$$A \cdot (B + C) = A \cdot B + A \cdot C \tag{3.9}$$

Al mismo tiempo, y dado que la implementación de la suma en GF(2^8) se realiza mediante la operación lógica "or exclusiva", la ecuación (3.8) puede escribirse tal que:

$$Sb_{(i,j)} = Sb_{REAL\,(i,j)} + Sb_{TRANS(i,j)} \tag{3.10}$$

Entonces la función *MixColumns*, descrita en (3.3), aplicada a (3.10) resulta:

$$\begin{pmatrix} Sm_{(i,1)} \\ Sm_{(i,2)} \\ Sm_{(i,3)} \\ Sm_{(i,4)} \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} Sb_{REAL\,(i,j)} + Sb_{TRANS(i,j)} \\ Sb_{REAL\,(i,j)} + Sb_{TRANS(i,j)} \\ Sb_{REAL\,(i,j)} + Sb_{TRANS(i,j)} \\ Sb_{REAL\,(i,j)} + Sb_{TRANS(i,j)} \end{pmatrix} \tag{3.11}$$

Que al aplicar la propiedad distributiva (3.9) se convierte en:

$$\begin{pmatrix} Sm_{(i,1)} \\ Sm_{(i,2)} \\ Sm_{(i,3)} \\ Sm_{(i,4)} \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} Sb_{REAL\,(i,j)} \\ Sb_{REAL\,(i,j)} \\ Sb_{REAL\,(i,j)} \\ Sb_{REAL\,(i,j)} \end{pmatrix} + \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} Sb_{TRANS(i,j)} \\ Sb_{TRANS(i,j)} \\ Sb_{TRANS(i,j)} \\ Sb_{TRANS(i,j)} \end{pmatrix} \tag{3.12}$$

De donde se concluye que:

$$\begin{aligned} Sm &= MixColumns\,(Sb_{REAL}) \oplus MixColumns\,(Sb_{TRANS}) \\ Sm &= Sm_{REAL} \oplus Sm_{TRANS} \end{aligned} \tag{3.13}$$

Sm: Matriz de estado a la salida de la función *MixColumns*.

Sm_{REAL}: Matriz de estado a la salida de la función *MixColumns* si hubiese sido procesado con la clave real.

Sm_{TRANS}: Matriz de enmascarado de final de ronda.

De forma análoga al resultado de la ecuación (3.8), el estado resultante de (3.13) corresponde al que se hubiese obtenido usando la clave real pero enmascarado (or exclusiva) con unos valores diferentes. Estos valores forman una matriz [4x4] denominada Sm_{TRANS}.

El algoritmo AES se realiza en múltiples rondas. Para un correcto funcionamiento es necesario iniciar cada una ellas con el estado que se hubiera obtenido procesando con la clave real. Para ello es necesario calcular la matriz Sm_{TRANS} (3.13) para revertir el proceso tal como se muestra a continuación:

$$Sm \oplus Sm_{TRANS} = Sm_{REAL} \oplus Sm_{TRANS} \oplus Sm_{TRANS} = Sm_{REAL} \tag{3.14}$$

La matriz Sm_{TRANS}, se obtiene al aplicar la función *MixColumns* sobre la matriz Sb_{TRANS} (3.13) que resulta de la ecuación (3.8), de forma que:

$$\begin{aligned} Sb_{TRANS(i,j)} &= Sb_{REAL\,(i,j)} \oplus Sb_{(i,j)} \\ Sb_{TRANS(i,j)} &= SBOX\left(Key_{REAL(i,j)} \oplus Estado_{(i,j)}\right) \oplus Sb_{(i,j)} \end{aligned} \tag{3.15}$$

A su vez, aplicando (3.7) en (3.15) se llega a la expresión:

$$\begin{aligned} Sb_{TRANS(i,j)} = SBOX\left(Key_{REAL(i,j)} \oplus Estado_{(i,j)}\right) \oplus \\ \oplus SBOX\left(Key_{FAKE(i,j)} \oplus Estado_{(i,j)}\right) \end{aligned} \tag{3.16}$$

Dado que todos los elementos del estado son bytes, se pueden precalcular los valores que pueden tomar los elementos de la matriz Sb_{TRANS} para los 256 posibles valores de los elementos del estado. Todos estos valores forman una tabla de 256 elementos a la que denominaremos $SBOX_{TRANS}$. Para realizar el cálculo de dicha tabla se plantea la siguiente sustitución,

$$j = Key_{FAKE} \oplus Estado \tag{3.17}$$

que junto con (3.6), hacen posible que la ecuación (3.16) pueda ser generalizada, para cualquier valor de los elementos del estado, de la siguiente forma:

$$SBOX_{TRANS}(j) = SBOX(j \oplus Key_{MASK}) \oplus SBOX(j) \tag{3.18}$$

El uso de esta expresión permite calcular todos los elementos de $SBOX_{TRANS}$ utilizando el pseudocódigo mostrado en (3.19). Finalmente, se puede obtener el valor correspondiente mediante una búsqueda en la tabla $SBOX_{TRANS}$ de forma análoga a como se busca en SBOX (apartado 3.2.1).

$$\begin{aligned} &for\ j = 0\ to\ 255 \\ &\quad SBOX_{TRANS}(j) = SBOX(j \oplus Key_{MASK}) \oplus SBOX(j) \\ &next\ j \end{aligned} \tag{3.19}$$

3.4. Vulnerabilidades añadidas.

Puesto que el sistema está implementado sin contramedidas adicionales el proceso de cálculo de la matriz Sb_{TRANS} es susceptible de ser atacado mediante un SCA que puede originar un falso positivo.

También es cierto que, mediante un análisis SCA de primer orden, el atacante puede conocer el valor de KEY_{FAKE}, por tanto, resulta imprescindible proteger el valor de KEY_{MASK} para que la seguridad del sistema no se vea comprometida. El enmascarado es la contramedida más adecuada para conseguir esta protección, ya que puede ser implementada tanto al nivel de software como de hardware, y por tanto no depende de la estructura interna del dispositivo.

A continuación se exponen cuales son los puntos débiles del sistema y como actúa el enmascarado que los protege.

- Un ataque de segundo orden entre las salidas de las funciones *SboxTrans* y *SubBytes* puede revelar la clave. Este tipo de ataque combina el consumo y el dato procesado en dos puntos del algoritmo (Apartado 2.10).

El modelo a usar en este caso se obtiene mediante la operación "or exclusiva" entre los datos a la salida de la función *SubBytes* (3.8) y a la salida de la función *SboxTrans* (3.15). Este modelo es coincidente con el que se usa en un ataque de primer orden a la salida de la función *SubBytes.*

$$Sb_{(i,j)} \oplus Sb_{TRANS(i,j)} = SBOX\left(Key_{REAL(i,j)} \oplus Estado_{(i,j)}\right) \oplus \\ \oplus Sb_{TRANS(i,j)} \oplus Sb_{TRANS(i,j)} \\ Sb_{(i,j)} \oplus Sb_{TRANS(i,j)} = SBOX\left(Key_{REAL(i,j)} \oplus Estado_{(i,j)}\right) \quad (3.20)$$

Por su parte, el consumo puede ser pre procesado usando la función definida en (2.22), haciendo que C_{ti} sea el consumo a la salida de *SubBytes* y C_{tk} el consumo a la salida de *SboxTrans*.

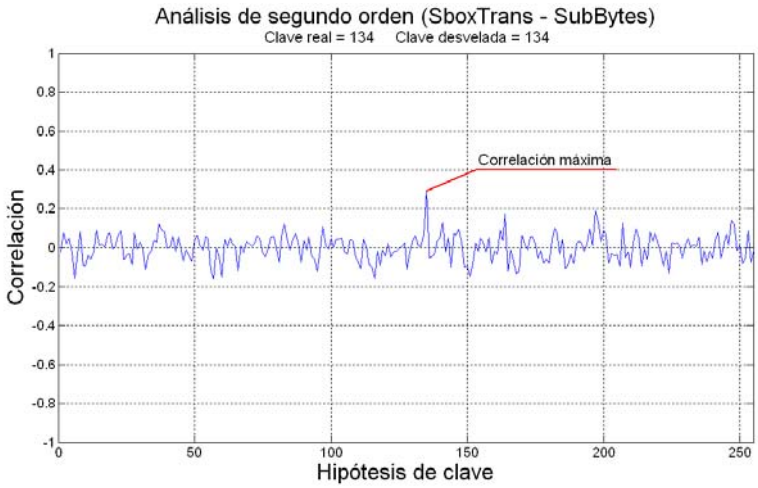

Figura 3.3: Simulación de ataque de segundo orden entre las salidas de *SubBytes* y *SboxTrans*.

La Figura 3.3 muestra el resultado simulado del ataque en estas condiciones. Puede apreciarse que aunque la correlación máxima obtenida es baja, puede ser diferenciada la hipótesis correcta de las que no lo son.

- Tras el proceso de reenmascarado en la ronda 1 (Figura 3.2) los valores que forman la matriz de estado son los que tendrían si se realizase el algoritmo con la clave real. Tras este punto, el sistema puede ser atacado en las mismas condiciones que un AES convencional sin contramedidas, aplicando las mismas restricciones que en la ecuación (3.5). Será necesario proteger el estado a la salida de esta función.

- A la salida de la función *SboxTrans* en la que se implementa la ecuación (3.16). En esta función intervienen el estado, la clave falsa y la clave real junto con la función SBOX. Se da la circunstancia de que el atacante conoce el estado, ya que en la primera ronda corresponde con el texto plano. También conoce KEY_{FAKE}, gracias a que el sistema está diseñado

para desvelarla mediante un ataque SCA. Y finalmente, conoce la función SBOX, puesto que forma parte del algoritmo AES. En estas condiciones el atacante está en disposición de modelar la salida del bloque *SboxTrans* y plantear un ataque de primer orden (Apartado 2.7.3) para desvelar la clave real.

- La función *MixColumn* requiere una especial atención puesto que se combinan bytes de diferentes filas de una misma columna. Si se usa un valor de máscara común para todos los bytes del estado, este puede ser cancelado en alguna de las operaciones “or exclusiva” necesarias para la implementar la función [Man'2007].

Hay que considerar también que el uso de la misma máscara en dos instantes del cálculo hace al sistema susceptible a un ataque de segundo orden tal y como se expone en el Apartado 2.10.

Si la entrada y la salida de la función *MixColumn* se enmascaran con el mismo valor y se utiliza un texto plano en el que únicamente varía uno de sus bytes [Lu'2009] el sistema es vulnerable.

3.5. Protección de la matriz de reenmascarado

El uso de una máscara común a todos los bytes del estado no garantiza la protección total del proceso de cálculo. [Man'2007] propone el uso de una máscara diferente para cada columna del estado para proteger la integridad de los datos en el procesado de la función *MixColumn*.

Los valores de las máscaras se generan de forma aleatoria para cada ciclo de encriptado o des encriptado. Estos valores forman una matriz denominada M_J:

$$M_J = \begin{pmatrix} m_0 & m_0 & m_0 & m_0 \\ m_1 & m_1 & m_1 & m_1 \\ m_2 & m_2 & m_2 & m_2 \\ m_3 & m_3 & m_3 & m_3 \end{pmatrix} \tag{3.21}$$

A causa de la inclusión de esta matriz de máscaras la salida de la función *SBoxTrans* resulta de la siguiente forma:

$$Sb_{TRANS(i,j)} = Sb_{REAL\ (i,j)} \oplus Sb_{(i,j)} \oplus M_{J(i,j)} \tag{3.22}$$

$M_{J(i,j)}$: Byte (i,j) de la matriz de máscaras M_J.

En consecuencia, la inclusión del término M_J, protege el sistema para los ataques definidos en el apartado 3.4. La Figura 3.4 muestra el resultado del ataque planteado anteriormente con la inclusión del término M_J, puede apreciarse que la clave queda protegida.

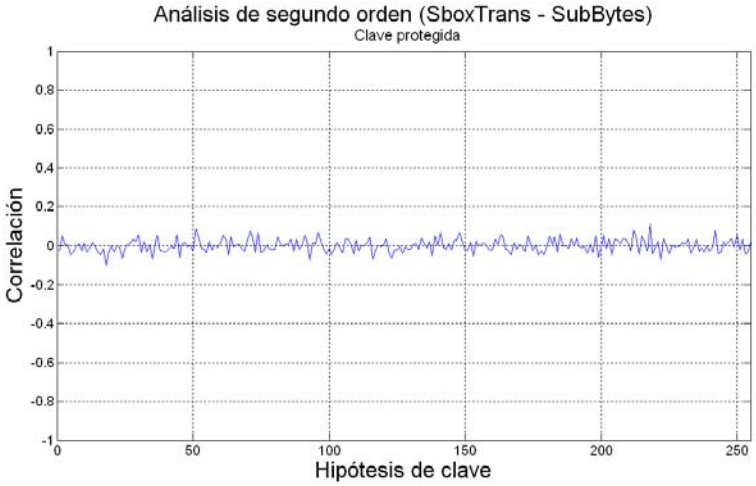

Figura 3.4: Simulación de ataque de segundo orden entre las salidas de *SubBytes* y *SboxTrans* con la inclusión de máscara aleatoria.

Seguidamente, al resultado de (3.22) se la aplica la función *MixColumn*. Aplicando la propiedad distributiva del campo de Galois $GF(2^8)$(3.9), se llega a la conclusión de que la matriz M_J se transforma en una nueva matriz de máscaras que denominada M_K de la siguiente manera:

$$M_K = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} m_0 & m_0 & m_0 & m_0 \\ m_1 & m_1 & m_1 & m_1 \\ m_2 & m_2 & m_2 & m_2 \\ m_3 & m_3 & m_3 & m_3 \end{pmatrix} = \begin{pmatrix} m_a & m_a & m_a & m_a \\ m_b & m_b & m_b & m_b \\ m_c & m_c & m_c & m_c \\ m_d & m_d & m_d & m_d \end{pmatrix} \tag{3.23}$$

Finalmente, al realizar el proceso de reenmascarado el estado queda protegido por la misma matriz M_K. Los valores de esta la matriz de máscaras deben ser cancelados en la función $AddRoundKey$ para realizar la siguiente ronda del algoritmo.

En la Figura 3.5 se muestra el sistema final con la indicación de las líneas de datos que están protegidas por cada una de las matrices de máscaras.

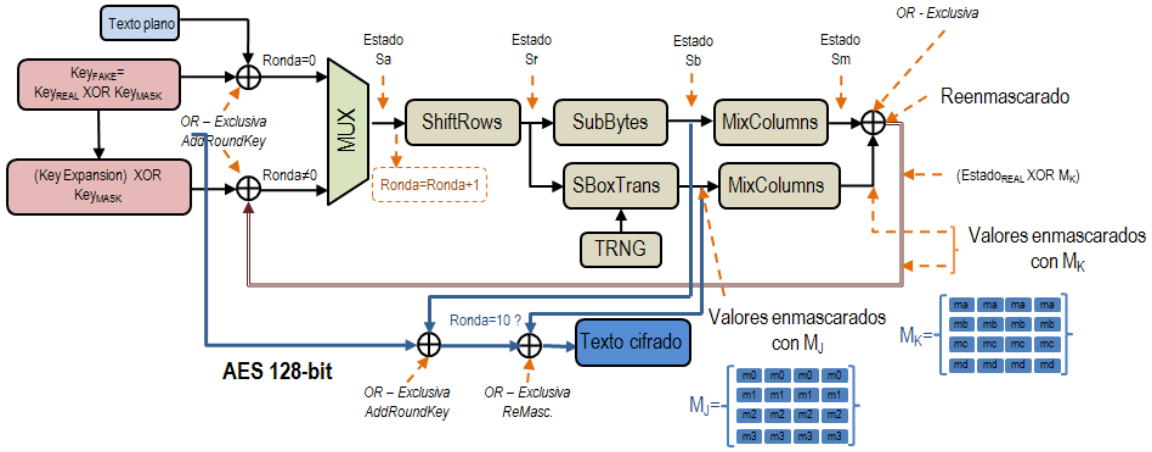

Figura 3.5: Estructura de implementación de AES con la contramedida basada en “faking” y las matrices de máscaras M_J y M_K.

3.6. Implementación software.

Los microprocesadores son dispositivos muy propicios para realizar con éxito ataques por análisis de consumo. Son tres los elementos que los hacen especialmente sensibles (Apartado 2.3.1):

a) Los diversos elementos del procesador se comunican mediante buses de datos y direcciones. Los buses son líneas de comunicación de gran tamaño. La escritura en ellos tiene un importante impacto en el consumo del dispositivo debido a su gran capacidad parásita.

b) La cantidad fija de líneas que forman el bus de datos limita el tamaño de la información que se transfiere, en consecuencia, la cantidad de datos que pueden estar conmutando de forma concurrente al byte que se está atacando.

c) Las operaciones complejas se descomponen en operaciones simples en las que se procesan pequeñas cantidades de información. Esta característica permite aislar el instante preciso en que se procesa cada dato y relacionarlo con el consumo.

Puesto que el sistema debe dar un falso positivo, la implementación del algoritmo AES debe realizarse de forma que pueda ser atacado mediante un modelo de HW o HD.

3.6.1. Acceso a memoria.

El acceso a memoria a través del bus de datos es el punto débil del sistema. En el algoritmo AES el estado es una matriz de 4x4 bytes alojada en la memoria del sistema [NIST'2001]. Cada vez que se accede a la lectura o escritura de un elemento del estado su valor se hace presente en el bus de datos y su marca de consumo se traslada a la línea de alimentación. También se almacenan en la memoria del dispositivo la tabla SBOX propia del AES.

Es necesario estudiar la forma en que se realiza el acceso a memoria en el dispositivo elegido para asegurar el funcionamiento de la contramedida.

3.6.1.1. Programación en bucle

El acceso a la matriz de estado se realiza mediante el anidamiento de dos bucles que barren filas y columnas como se muestra en (3.24). El procesador debe gestionar los índices de los bucles y los saltos de programa al mismo tiempo que realiza la función *SubBytes*. Para completar la función cada valor de la matriz de estado debe ser leído desde la memoria de datos y su valor debe ser usado como índice de búsqueda en la tabla SBOX.

```
for(i=0;i<4;i++)
{
  for(j=0;j<4;j++)
  {
    state[i][j] = sbox[state[i][j]];
  }
}
```
(3.24)

La Figura 3.6 muestra la simulación de la ejecución, sobre un microprocesador soft-core Microblaze, de la función *SubBytes* en un caso en que

los cuatro elementos de la primera fila son X[32]“0A132489” y los valores correspondientes de la tabla SBOX son X“677D36A7.

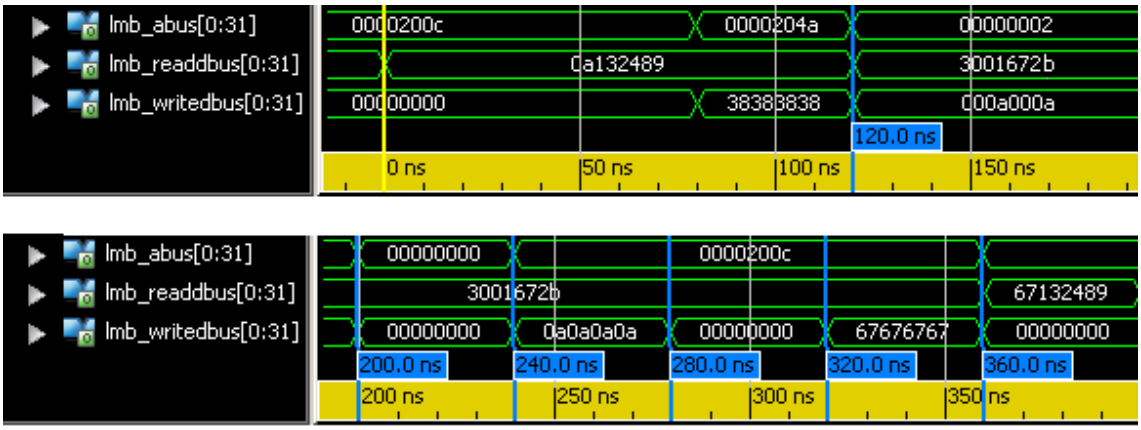

Figura 3.6: Simulación de acceso mediante bucles.

La simulación presenta la siguiente secuencia:

a) **Ts[33]=0ns,** los 4 bytes correspondientes a la primera fila aparecen en el bus de lectura. Puesto que en este instante se está procesando el primer elemento del estado únicamente se usa el dato X“0A”. Este valor es almacenado en un registro interno del procesador.

b) **Ts=120ns,** la parte de la tabla SBOX en la que se encuentra el dato correspondiente a la posición X“0A” es escrita en el bus de lectura de datos.

c) **Ts=200ns**, el bus de escritura es precargado a X“00000000”

d) **Ts=240ns,** en el bus de escritura aparece por cuadruplicado el dato del estado que se está procesando X“0A”. En este punto se produce una pérdida de información modelable mediante HW, puesto que el dato corresponde al resultado de la función anterior es un dato modelable y puede ser atacado.

e) **Ts=280ns,** el bus de escritura es precargado a X“00000000”

[32] Indica que el valor está expresado en hexadecimal.

[33] Tiempo de simulación.

f) **Ts=320ns,** el dato obtenido de la tabla SBOX aparece por cuadruplicado en el bus de escritura. De nuevo es atacable mediante HW y es el punto crítico del ataque a la función *SubBytes*.

g) **Ts=360ns,** se inicia el procesado del segundo byte del estado. Puede apreciarse como el primer byte del estado ya presenta el resultado de la función.

Esta opción se ejecuta en 24.2 µs equivalentes a 605 ciclos de reloj a una frecuencia de 25 MHz.

3.6.1.2. Programación desarrollada

El acceso a la matriz de estado se realiza de forma desarrollada escribiendo directamente las 16 líneas de código que acceden a cada elemento del estado.

```
state[0][0]=sbox[state[0][0]];
state[0][1]=sbox[state[0][1]];
state[0][2]=sbox[state[0][2]];
…
```
(3.25)

En este caso el procesador no debe gestionar los índices y únicamente ha de acceder a la memoria tanto para leer como para escribir. La Figura 3.7 muestra la simulación en la que se procesan los mismos datos que en la Figura 3.6.

La simulación presenta la siguiente secuencia:

a) **Ts=0ns,** los 4 bytes correspondientes a la primer fila aparecen en el bus de lectura. Puesto que en este instante se está procesando el primer elemento del estado, únicamente se usa el dato X“0A”. Este valor es almacenado en un registro interno del procesador.

b) **Ts=120ns,** la parte de la tabla SBOX, en la que se encuentra el dato correspondiente a la posición X“0A”, es escrita en el bus de lectura de datos.

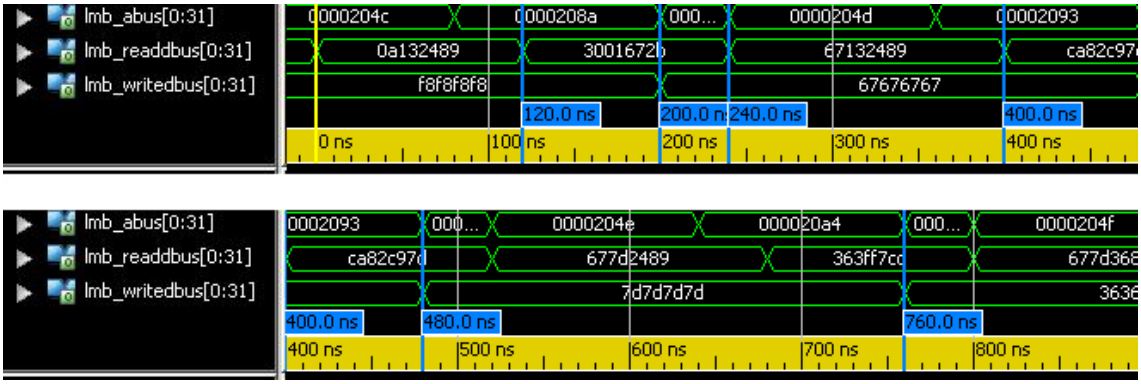

Figura 3.7: Simulación de acceso desarrollado.

c) **Ts=200ns,** el dato obtenido de la tabla SBOX aparece por cuadruplicado en el bus de escritura. Hay que notar que el valor anterior del bus de escritura es un dato arbitrario generado por instrucciones no relacionadas con el algoritmo y que no puede ser modelado por el atacante. En este punto no puede plantearse un modelo HW puesto que el dato anterior no es X“00000000” y tampoco es posible plantear un ataque HD puesto que el dato anterior es arbitrario.

d) **Ts=240ns,** de nuevo los 4 bytes correspondientes a la primera fila aparecen en el bus de lectura, en esta caso el dato a procesar es el segundo cuyo valor es X“13”. Puede apreciarse que el primer dato corresponde a la salida de la función *SubBytes* correspondiente al byte anterior.

e) **Ts=400ns,** la parte de la tabla SBOX en la que se encuentra el dato correspondiente a la posición X“13” es escrita en el bus de lectura de datos.

f) **Ts=480ns,** el dato obtenido de la tabla SBOX aparece por cuadruplicado en el bus de escritura. En este caso tanto el dato que ocupaba el bus anteriormente como el actual son datos modelables.

En teoría podría plantearse un ataque por HD entre la salida *SubBytes* de un byte estado y la salida *SubBytes* del byte posterior. Este análisis es difícil de realizar ya que implicaría modelar 2^{16} hipótesis de clave (2 bytes) y conocer el orden en que se procesas los bytes del estado en cada función.

Esta opción se ejecuta en 4,4 µs equivalentes a 110 ciclos de reloj a una frecuencia de 25 MHz.

A la luz de las simulaciones realizadas, el método expuesto en el apartado 3.6.1.1 deberá ser usado en aquellas partes del algoritmo que deban ser sensibles al ataque por análisis de consumo. En cambio, aquellas partes del algoritmo que pretendan protegerse deberán realizarse mediante el método expuesto en el apartado 3.6.1.2.

3.6.2. Operaciones adicionales

A parte del cálculo de la matriz Sm_{TRANS}, la implementación de la contramedida precisa operaciones adicionales que incrementan el tiempo de procesado. Las operaciones mencionadas son las siguientes.

- **Es necesario el enmascaramiento de la clave expandida.** El algoritmo AES realiza la expansión de la clave para favorecer la dispersión de la misma a lo largo del proceso. En el caso del AES-128 implica que la clave inicial de 16 bytes acaba siendo una tabla de 11x16 bytes (11 vectores de 16 bytes) uno por cada ejecución de la función *AddRounKey* (Apartado 3.2.1). Cada uno de estos vectores debe ser enmascarado con Key_{MASK}. Este proceso debe ser realizado cada vez que se modifique Key_{REAL} o Key_{MASK}.

- Las tablas $SBOX_{TRANS}$ deben ser precalculadas. La implementación de esta función se realiza mediante la ecuación (3.18) que se implementa según el pseudocódigo expuesto en (3.19). En dicha función interviene el valor $Sb_{REAL(i,j),}$ que se obtiene de la tabla SBOX definida en el algoritmo, y cuyo uso representa una importante vulnerabilidad del sistema. Este dato, al ser leído desde la memoria del dispositivo, viaja por el bus y puede provocar pérdidas de información explotable.

Para evitar el posible ataque al valor $Sb_{REAL(i,j)}$, los valores de las tablas $SBOX_{TRANS}$ deben ser enmascarados usando la matriz $M_{J(i,j)}$ (apartado 3.5). En el cálculo se accede a la función SBOX propia del algoritmo, por tanto este acceso ha de realizarse de forma independiente de la clave y del texto plano. El precálculo de $SBOX_{TRANS}$ permite obtener el resultado de la fun-

ción sin la necesidad de acceder a las tablas SBOX, de esta forma se evita la posible pérdida de información.

Puesto que M_J es un valor aleatorio que cambia para cada ciclo de encriptado, las tablas $SBOX_{TRANS}$ también deben ser precalculadas con la misma frecuencia. El pseudocódigo que realiza el precálculo es el mostrado en (3.26).

$$\begin{aligned} &for\ j = 0\ to\ 255 \\ &\quad SBOX_{TRANS}(j) = SBOX(j \oplus Key_{MASK}) \oplus SBOX(j) \oplus M_J \\ &next\ j \end{aligned} \tag{3.26}$$

Las tablas ocupan en memoria 4 kB de espacio correspondientes a 256 B por cada uno de los 16 valores de Key_{MASK} disponibles.

Si bien el primer cálculo de las tablas $SBOX_{TRANS}$ debe realizarse mediante la función expresada en (3.26), en sucesivas rondas puede ser recalculada mediante la simple adición de una máscara nueva a la tabla almacenada en la memoria. La forma de realizarlo es la que se muestra a continuación:

$$\begin{aligned} &for\ j = 0\ to\ 255 \\ &\quad SBOX_{TRANS}(j) = SBOX_{TRANS}(j) \oplus M_J \\ &next\ j \end{aligned} \tag{3.27}$$

Supongamos que la tabla está enmascarada con un valor M_{J1}, y se aplica el algoritmo (3.27) con un nuevo valor M_{J2}, se puede considerar que la nueva tabla $SBOX_{TRANS}$ es la original pero enmascarado con una nueva máscara M_{JN} con valor:

$$M_N = M_{J1} \oplus M_{J2} \tag{3.28}$$

Ahora, aplicando la ecuación (3.23) a la nueva máscara se obtiene la Matriz M_K asociada a ésta:

$$M_{KN} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot (M_{J1} \oplus M_{J2}) \tag{3.29}$$

Que al aplicar la propiedad distributiva se obtiene:

$$M_K = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot M_{J1} \oplus \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \cdot M_{J2} = M_{K1} \oplus M_{K2} \tag{3.30}$$

De donde se deduce que la nueva matriz M_{KN} asociada a la matriz M_{JN} puede obtenerse mediante una operación or-exclusiva entre la M_{K1} y la M_{K2}.

El cálculo de la tabla realizado por este método aporta dos grandes ventajas: el proceso es más rápido al reducirse la complejidad del mismo y se evita realizar búsquedas en la tabla SBOX evitando de este modo la perdida de información asociada a esa operación.

- La matriz de máscaras M_K es necesaria tras la ejecución de cada ronda. Puesto que el estado se enmascara con esta matriz, tal y como se expone en la ecuación(3.23), se deben calcular los valores que la forman para poder cancelarla. El cálculo se realiza aplicando la función *MixColumn* a una de las columnas de la matriz M_J.

Seguidamente esta matriz debe ser eliminada antes de proceder al inicio de la siguiente ronda. Es determinante que esta operación se realice en el orden correcto para no comprometer la seguridad del sistema.

El estado resultante de la ronda anterior está enmascarado con la matriz M_K de forma que:

$$Estado_{(i,j)} = Estado_{REAL\ (i,j)} \oplus M_{K(i,j)} \tag{3.31}$$

En primer lugar debe aplicarse la función *AddRoundKey* y se obtiene el siguiente resultado:

$$Estado_{(i,j)} = Estado_{REAL\,(i,j)} \oplus Key_{ROUND} \oplus Fake_{Mask} \oplus M_{K(i,j)} \tag{3.32}$$

Para posteriormente eliminar la matriz M_K mediante la operación “or exclusiva”:

$$\begin{aligned} Estado_{(i,j)} &= Estado_{REAL\,(i,j)} \oplus Key_{ROUND} \oplus \\ &\quad \oplus Fake_{Mask} \oplus M_{K(i,j)} \oplus M_{K(i,j)} \\ Estado_{(i,j)} &= Estado_{REAL\,(i,j)} \oplus Key_{ROUND} \oplus Fake_{Mask} \end{aligned} \tag{3.33}$$

Entonces el estado queda listo para la siguiente ronda. La inversión de estas operaciones puede dejar al $Estado_{REAL}$ desprotegido y generar pérdida de información explotable.

3.6.3. Coste computacional

Las operaciones adicionales descritas necesitan un tiempo de procesado adicional. Este hándicap debe ser minimizado para perjudicar en lo mínimo posible al tiempo de ejecución del algoritmo.

Las operaciones pueden dividirse en tres categorías:

a) las que se realizan al iniciar el sistema, se varía la Key_{REAL} o se varía la Key_{MASK},

b) aquellas que deben realizarse en cada ciclo de encriptado y

c) las que deben realizarse en cada ronda del algoritmo.

En el primer grupo encontramos únicamente el enmascaramiento de la clave expandida. El impacto de esta operación no es representativo sobre la respuesta del sistema, dado que el tiempo de ejecución se reparte entre todos los textos planos que se procesen sin modificar las claves.

Las operaciones pertenecientes al segundo grupo son: determinar aleatoriamente las máscaras de la matriz M_J y el posterior cálculo de la matriz M_K (Apartado 3.5). Posteriormente, partiendo de los valores de M_J, deben calcularse las dieciséis tablas $SBOX_{TRANS}$ que se usarán en la función *SboxTrans*. La forma de que el tiempo de cálculo no afecte al tiempo total

del proceso es ejecutar el código correspondiente de forma concurrente con la comunicación externa del dispositivo. Es posible realizar las operaciones pertinentes mientras, por ejemplo, se envía un texto encriptado y se recibe el siguiente texto plano vía RS232.

Finalmente, las operaciones del tercer grupo son las necesarias para el cálculo de la matriz Sm_{TRANS}. Al ser necesario realizarlas para cada ronda del algoritmo tienen una gran influencia en el tiempo total de ejecución del mismo. Atendiendo a la dispersión de la clave a lo largo del proceso y a las condiciones de ataque planteadas en [Lu'2009], el ataque puede dirigirse a cualquier valor intermedio previo a la ronda 2 y posterior a la ronda 8, por tanto sería necesario proteger el algoritmo en esas partes de su ejecución. Para reducir el tiempo de ejecución las rondas 3 a 7 se realizarían utilizando el algoritmo normal sin implementar la contramedida.

3.7. Implementación software/hardware.

El siguiente paso en la implementación es el uso de una estructura basada en codiseño software hardware. En este caso el algoritmo se ejecuta mediante software en un procesador, mientras que el procesado de las funciones que calculan la matriz de reenmascarado se ejecutan en un coprocesador hardware diseñado a tal efecto. La Figura 3.8 muestra que parte de la estructura se implementa en coprocesador hardware y que parte se resuelve mediante software.

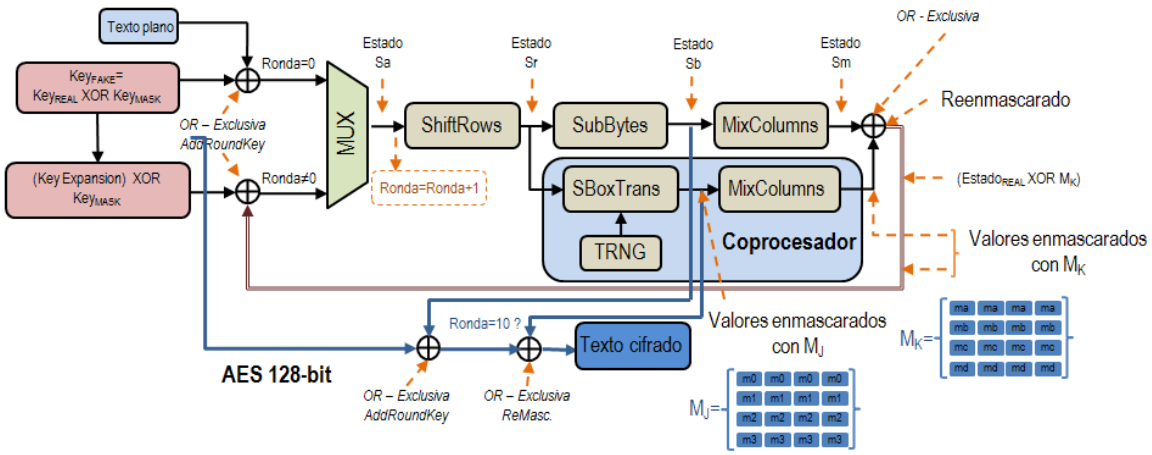

Figura 3.8: Estructura de implementación software/hardware del AES con la contramedida basada en “faking”.

La implementación de este tipo de estructuras suele realizarse en un dispositivo FPGA en el que conviven el procesador "softcore[34]" y el coprocesador correspondiente.

El objetivo principal es que el tiempo de ejecución del algoritmo con la contramedida sea lo más parecido posible al tiempo de ejecución del mismo sin la contramedida.

3.7.1. Vulnerabilidades del coprocesador hardware.

El procesado de la información en el coprocesador está enmascarado por los motivos que se exponen en el apartado 3.5. Cuando se implementa el coprocesador, es muy importante tomar en consideración algunas vulnerabilidades relacionadas con las implementaciones hardware que usan el enmascarado como medida de protección contra el análisis de consumo.

Los dispositivos hardware pueden no ser completamente seguros si las operaciones internas no son cuidadosamente realizadas. Una situación de riesgo ante un análisis de consumo, que utilice como modelo la distancia de Hamming, puede presentarse si dos valores enmascarados con el mismo valor son escritos de forma consecutiva en el mismo registro o bus de datos [Koc'1999].

Sean $v_{(tk)}$ y $v_{(tk+1)}$ dos valores intermedios modelables que se escriben de forma consecutiva en un mismo registro, sus valores enmascarados son:

$$v_{m(t_k)} = v_{(t_k)} \oplus m \quad y \quad v_{m(t_{k+1})} = v_{(t_{k+1})} \oplus m \tag{3.34}$$

El cálculo de la HD se realiza de la siguiente manera:

$$Hd\left(v_{m(t_k)}, v_{m(t_{k+1})}\right) = Hw\left(v_{m(t_k)} \oplus v_{m(t_{k+1})}\right) \tag{3.35}$$

Finalmente, sustituyendo los valores se concluye que la distancia de Hamming es independiente del valor de la máscara que se utilice.

[34] Procesador totalmente implementado usando síntesis lógica en algún dispositivo reconfigurable tipo ASIC, FPGA o CPLD y que puede ser fácilmente configurado según la necesidad.

$$Hw\left(v_{m(t_k)} \oplus v_{m(t_{k+1})}\right) = Hw\left(v_{(t_k)} \oplus m \oplus v_{(t_{k+1})} \oplus m\right) = \\ = Hw\left(v_{(t_k)} \oplus v_{(t_{k+1})}\right) \quad (3.36)$$

$$Hw\left(v_{(t_k)} \oplus v_{(t_{k+1})}\right) = Hd\left(v_{(t_k)}, v_{(t_{k+1})}\right)$$

Este tipo de vulnerabilidad puede evitarse utilizando cualquiera de las siguientes técnicas:

a) **Precarga del registro a un valor determinado:** de este modo la transición no es entre dos datos conocidos y enmascarados con el mismo valor. Por ejemplo, precargando el registro con el valor hexadecimal X”00” los modelos HD y HW son coincidentes y, puesto que la máscara es aleatoria y desconocida, no son modelables.

b) **Modificación de la máscara entre valores consecutivos:** si la máscara es diferente entre valores no se produce su cancelación al calcular HD mediante la fórmula (3.36) y por tanto tampoco es modelable.

3.7.2. Consumo adicional del coprocesador.

La inclusión del coprocesador implica un mayor número de recursos internos del dispositivo reconfigurable que operan en un momento determinado. El consumo del coprocesador se suma al consumo de microprocesador y debe cumplir las siguientes especificaciones:

a) **No debe detectarse mediante un análisis simple de la traza**. Para ello los datos deben ser procesados en bloques lo más pequeños posibles (8 bits), para que el consumo dinámico del coprocesador sea también lo mínimo posible. Al mismo tiempo, el procesado debe distribuirse con el objetivo de que el consumo del microprocesador principal oculte el consumo del coprocesador.

b) **No debe alterar la correlación con la clave falsa.** Considerando que el coeficiente de correlación se ve alterado por el ruido de conmutación del sistema (apartado 2.7.2), el coprocesador no debe interferir en los instantes en que se está filtrando información en el cálculo del AES. Para ello, el coprocesador debe interrumpir

su funcionamiento cuando se den las condiciones de acceso a memoria explicadas en el apartado 3.6.1.1.

3.7.3. Funciones del coprocesador.

Las operaciones propias de la contramedida deben ser distribuidas entre las secciones software y hardware del sistema. El coprocesador hardware ha de calcular la matriz de reenmascarado M_K en cada ronda del algoritmo. Este proceso debe hacerse atendiendo a las medidas de seguridad expuestas en el apartado 3.5.

3.7.3.1. Cálculo de la matriz M_K y tablas SBOXTRANS

Este cálculo requiere la selección aleatoria de las máscaras de la matriz M_J, posteriormente se calculan los valores de las mascaras de la matriz M_K. Esta operación ha de realizarse para cada ciclo de encriptado.

Las tablas $SBOX_{TRANS}$ han de almacenarse en una memoria RAM de 4kB implementada en el propio coprocesador tal y como se muestra en la Figura 3.9. El contenido de las memorias debe ser precalculado en cada ciclo de encriptado puesto que la matriz M_K también se modifica con la misma frecuencia.

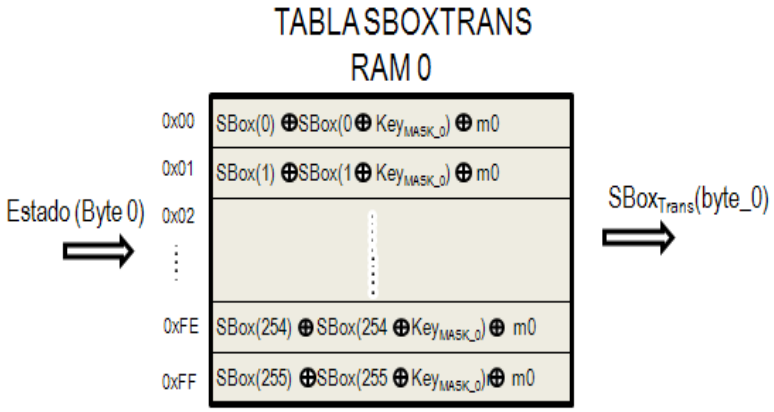

Figura 3.9: Implementación hardware del byte 0 de las tablas SBOXTRANS enmascarada.

Tal y como muestra el esquema de implementación de la Figura 3.9, el dato registrado corresponde al valor enmascarado evitando de esta manera posibles pérdidas de información.

3.7.3.2. Cálculo de la matriz de reenmascarado.

El cálculo de esta matriz de reenmascarado debe realizarse en cada ronda del algoritmo, de forma concurrente a la función *SubBytes*.

El coprocesador recibe los valores que forman el estado tras la ejecución de la función *ShiftRows*. La comunicación entre el microprocesador y el coprocesador se realiza a través de un bus local de comunicación. Acto seguido se inicia el proceso de cálculo de la matriz Sb_{TRANS} mediante una búsqueda en las tablas $SOBX_{TRANS}$, la matriz se obtiene enmascarada mediante $M_{J.}$. Una vez obtenidos los 16 valores que la forman se aplica la función *MixColumns* con el fin de obtener la matriz Sm_{TRANS} enmascarados con los valores de la matriz M_K.

Esta información debe estar disponible antes de que se complete la ejecución de la función MixColumn del algoritmo AES, dado que dicha información es necesaria para proceder al reenmascarado del estado.

El coprocesador deberá retornar también los valores [m_a, m_b, m_c, m_d] que forman la matriz M_K y que han de ser cancelados en la ejecución de la función *AddRoundKey* de la siguiente ronda.

3.8. Implementación Hardware.

Los sistemas puramente hardware son más robustos ante ataques de consumo que los sistemas software. Esto es debido a que un microprocesador es un dispositivo del que el atacante puede obtener documentación sobre su estructura y funcionamiento. Con esta información puede simularse el funcionamiento y planificar el ataque a realizar. Por el contrario, cuando se implementa un sistema hardware no se está atado a ninguna estructura y el diseñador dispone de total libertad para crear la suya propia. La dificultad que un atacante puede encontrar para acceder a esta información es mucho mayor que en el caso de un microprocesador comercial.

La implementación de la contramedida objeto de esta tesis requiere que el sistema sea sensible a un ataque por análisis de consumo basado en los modelos HD, HW o ambos. La estructura debe crearse para conseguir este propósito.

Hay que considerar también que el diseño hardware permite el procesado concurrente de todos los bytes que forman el estado. La correlación que se obtiene, cuando se ataca un byte de la clave, se ve afectada por el ruido de conmutación generado por el resto de los bits procesados (apartado 2.7.2). En este caso, el sistema también debe calcular la matriz de reenmascarado, por tanto, el consumo dinámico producido por este cálculo afectará a la correlación falsa que se pretende conseguir.

Supongamos el siguiente escenario:

a) Disponemos de un sistema hardware que procesa el algoritmo AES en bloques de 32 bits, o lo que es lo mismo, todos los bits de una columna o fila del estado en un mismo ciclo de reloj.

b) Mediante un análisis de consumo se pretende obtener un byte de la clave.

Según estas premisas, el número de bits que están conmutando y no forman parte del byte modelado es 24. En estas condiciones, según la ecuación (2.14), la correlación máxima esperada es 0.5153.

Si de forma paralela se está procesando el cálculo de la matriz de reenmascarado, el consumo dinámico producido por los 32 bits implicados en este cálculo se añade al consumo total del dispositivo. Por tanto el número de bits que forman el ruido de conmutación pasa a ser de 56 y la correlación máxima esperada será de 0.3561. En otras palabras, el cálculo concurrente de la matriz de reenmascarado provoca que la correlación se reduzca al 69%.

Un dato parecido se obtiene de un nuevo escenario en el que se procesa la información en bloques de 128 bits, es decir, todo el estado en un único ciclo de reloj. En el primer caso los bits de ruido son 120 y la correlación máxima esperada 0.2548. En el segundo caso los bits de ruido son 244 y la correlación se reduce a 0.1702. El coeficiente se ha reducido en la misma proporción que en el escenario anterior.

El cálculo de la matriz de reenmascarado influirá notablemente en la correlación máxima esperada. Es importante que el registro sensible al

ataque, y que está destinado a provocar la correlación con la clave falsa, no funcione de forma concurrente con dicho cálculo.

3.9. Forzado de la correlación.

La implementación hardware del algoritmo AES se basa en una estructura en la que se alternan registros y circuitos combinacionales. Estos circuitos combinacionales están descritos mediante un lenguaje HDL[35] para que resuelvan las diferentes funciones del algoritmo. El resultado de las funciones se almacena en registros que son activados por una máquina de estados, de esta forma, el estado evoluciona de un registro al siguiente hasta completar el proceso (Figura 3.10).

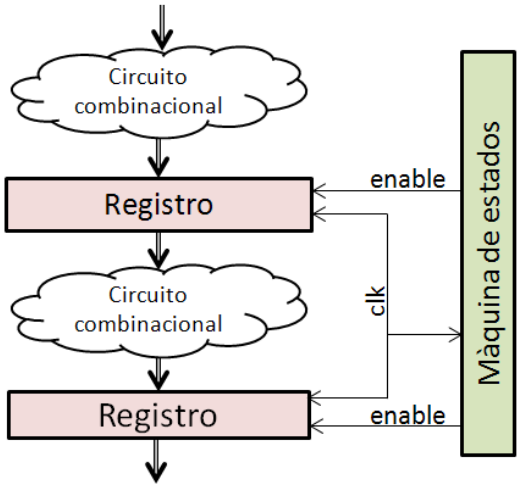

Figura 3.10: Estructura básica de un sistema descrito en HDL.

La pérdida de información explotable se producirá en el momento en que los datos son almacenados en los registros correspondientes. Por tanto, el diseño que se haga de los registros determinara el modelo a utilizar para conseguir romper la clave del sistema.

El planteamiento de un ataque basado en HW (apartado 2.3.2) implica que todos los bits del registro han de estar precargados con en el valor lógico “0” antes de que el valor del estado se almacene en él. La precarga debe realizarla la máquina de estados que controla el sistema.

[35] De las siglas en ingés “Hardware Description Language”

En cambio, si el ataque se basa en HD (apartado 2.3.3), es necesario conocer el estado de todos los bits del registro para calcular la distancia de Hamming con el valor que se almacenará en el siguiente ciclo de reloj. Esto implica que dos valores conocidos por el atacante deben ser escritos de forma consecutiva en el mismo registro. Se da la circunstancia que el único valor conocido por el atacante es el texto plano enviado al dispositivo y, a partir de él y de la hipótesis de clave, se puede obtener el segundo valor necesario para calcular el modelo basado en la distancia de Hamming.

3.10. Conclusiones.

El método presentado oculta la correlación de la clave real tras una fuerte correlación con una clave falsa. Dado que el método trabaja a nivel de algoritmo es posible implementarlo usando diversas técnicas y en diversas plataformas.

Se han establecido las bases para una implementación software y se ha incidido sobre las particularidades de la implementación hardware-software y de la implementación puramente hardware.

Las características de la contra medida deben tenerse en cuenta en cualquiera de las plataformas que se usen y tratar de que su uso perjudique en lo mínimo posible el tiempo de procesado del algoritmo y/o las necesidades físicas de los dispositivos.

4. Sistema de desarrollo, medidas y procesado

4.1. Introducción

Un ataque por análisis de consumo persigue relacionar el dato que se está procesando en un dispositivo electrónico con la energía que consume en el preciso instante en que el dato se procesa.

Para la verificación de la contramedida propuesta en esta tesis, se han implementado diversas soluciones y se han efectuado los ataques por análisis de consumo pertinentes en cada caso. El equipo necesario para realizarlo es el siguiente:

a) Sistema de desarrollo sobre el que implementar las propuestas presentadas y que es el dispositivo objetivo del ataque.

b) Instrumento de captura digital de datos que admita sondas de medida de corriente y de tensión. Ha de tener capacidad para comunicarse con un equipo informático.

c) Hardware y software informático para automatizar el proceso de medida y para el posterior procesado de las trazas.

La Figura 4.1 muestra la conexión de los elementos para la captura de datos previa a la realización del análisis de consumo.

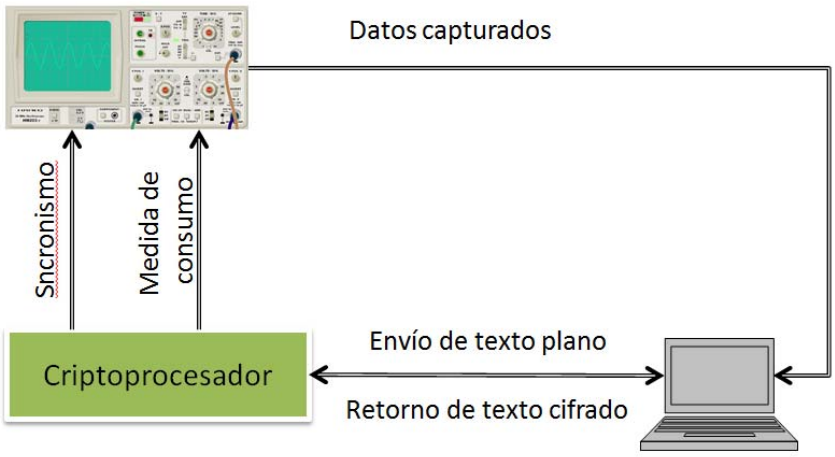

Figura 4.1:Esquema básico de estación de ataque por análisis de consumo

Es necesario destacar que la integridad de las medidas de corriente influye notablemente en el éxito del ataque. Si las trazas presentan interferencias, la relación señal ruido entre éstas y la información explotable por el atacante se reduce. El atacante se ve forzado a tomar una gran cantidad de medidas para minimizar el perjuicio que la presencia de ruido produce. Por otro lado, el objetivo de la parte práctica de esta tesis es verificar el funcionamiento del "Faking". Para ello es necesario obtener las trazas lo más limpias posibles para que ningún otro factor pueda influir en la valoración del mismo.

4.2. Placa de desarrollo

La placa de desarrollo es la Sasebo-GII[36] [AIST'2009]. Esta placa está específicamente diseñada para la realización de ataques por análisis de consumo. En su diseño se han tenido en cuenta algunas características que permiten la captura de trazas de consumo de forma sencilla con una SNR grande. En estas condiciones de trabajo es posible verificar el funcionamiento de las contramedidas sin necesidad de capturar una gran cantidad de trazas; de esta forma se optimiza el tiempo de trabajo [Man'2007]. En la

[36] SAKURA Hardware Security Project, http://satoh.cs.uec.ac.jp/SAKURA/index.html

Figura 4.2 puede verse el diagrama de bloques y el aspecto externo de la misma.

La placa mencionada incorpora una FPGA Virtex 5 XC5VLX50[37] destinada a la implementación de los procesadores criptográficos. Esta FPGA tiene conectividad con los siguientes elementos externos:

a) Memoria SRAM de 2Mbits.

b) 8 indicadores LED para monitorizar señales digitales.

c) 8 interruptores y 1 pulsador para generar señales digitales.

d) 24 líneas de entrada-salida digital de propósito general.

e) 1 línea de señal de CLK externa a la placa.

f) 1 memoria SPI-ROM de 16 Mbits para almacenar los "bitstreams" de configuración inicial de la FPGA.

g) 1 conector de programación JTAG para la descarga de "bitstreams" de configuración.

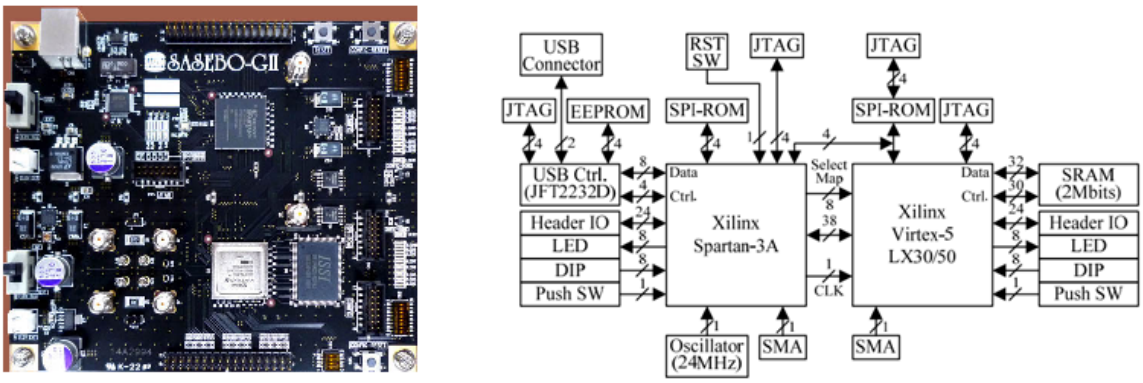

Figura 4.2:Diagrama de bloques y fotografía de la placa de desarrollo Sasebo-GII

Paralelamente el sistema dispone una segunda FPGA Spartan-3A modelo XC3S400A[37] cuya misión es gestionar la comunicación entre la primera FPGA y el sistema informático. Esta FPGA dispone de los siguientes periféricos:

[37] Xilinx Inc: http://www.xilinx.com/

a) Integrado FT2232D [FTDI'2010] de la compañía FTDI[38] que corresponde a un driver de comunicaciones USB. Este integrado puede configurarse para emular un puerto de comunicaciones RS232.

b) 8 indicadores LED para monitorizar señales digitales.

c) 8 interruptores y 1 pulsador para generar señales digitales.

d) 24 líneas de entrada-salida digital de propósito general.

e) 1 línea de señal de CLK externa a la placa.

f) 1 oscilador de 24 MHz.

g) 1 memoria SPI-ROM de 16 Mbits para almacenar los "bitstreams" de configuración inicial de la FPGA.

h) 1 conector de programación JTAG para la descarga de "bitstreams" de configuración.

Adicionalmente, existen 38 líneas bidireccionales de comunicación entre las dos FPGAs que pueden usarse para el intercambio de información entre ellas. La señal de reloj, propia de la placa y que está conectada a la FPGA de comunicaciones, puede ser transmitida a la FPGA criptográfica a través de una línea dedicada.

Puesto que ambas son productos de Xilinc Inc. se han utilizado las herramientas proporcionadas por el fabricante para la implementación de todos los sistemas.

Las características esenciales que hacen de ésta una plataforma de desarrollo específica para el testeo de sistemas criptográficos se encuentran en la estructura de los reguladores de alimentación presentes en la placa y en la distribución de alimentaciones y masas practicada.

La alimentación eléctrica de todo el sistema puede realizarse a través del mismo cable USB que se usa para la comunicación, o desde una fuente de alimentación externa. En ambos casos la alimentación se estabiliza mediante el uso de reguladores lineales de tensión, en ningún caso se usan

[38] Future Technology Devices International Ltd. http://www.ftdichip.com

reguladores conmutados para evitar el ruido de conmutación que generan. El diagrama de bloques de la alimentación se muestra en la Figura 4.3.

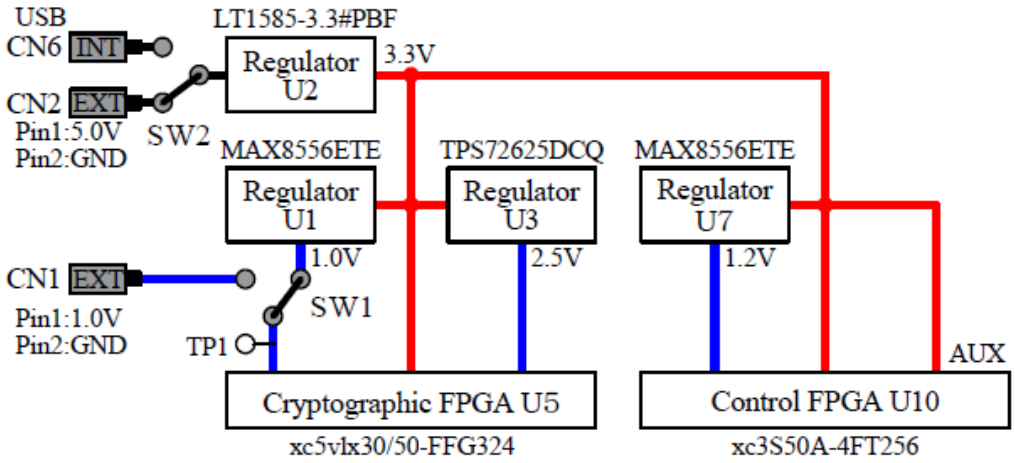

Figura 4.3:Esquema de bloques de la alimentación de la placa.

La alimentación de 5V procedente del cable USB se estabiliza a 3.3 V para alimentar a todos los demás dispositivos de la placa. Con esta tensión estabilizada se alimentan los puertos de entrada/salida de las dos FPGA y también se obtienen los 1.2 V necesarios para alimentar el núcleo lógico de la FPGA de control. La FPGA criptográfica requiere, a parte de los 3.3 V antes comentados, 2.5 V para una parte de los puertos de entrada/salida y 1 V para el núcleo lógico. Ambas tensiones se obtienen a partir del bus de continua de 3.3 V y existe la posibilidad de suministrar externamente la tensión de 1 V destinada a alimentar el núcleo digital. Es la intensidad consumida por el núcleo digital de la FPGA criptográfica la que se mide para proceder al análisis de consumo.

Hay que destacar que, tanto el plano de masa de la FPGA criptográfica, cómo los de alimentación están separados eléctricamente del resto del circuito para evitar la conducción de ruido hacia la medida de corriente. La conexión de referencia de masas se realiza a través de filtros paso bajo para evitar la circulación de señales de alta frecuencia. La Figura 4.4 muestra la forma en que se realiza esta conexión.

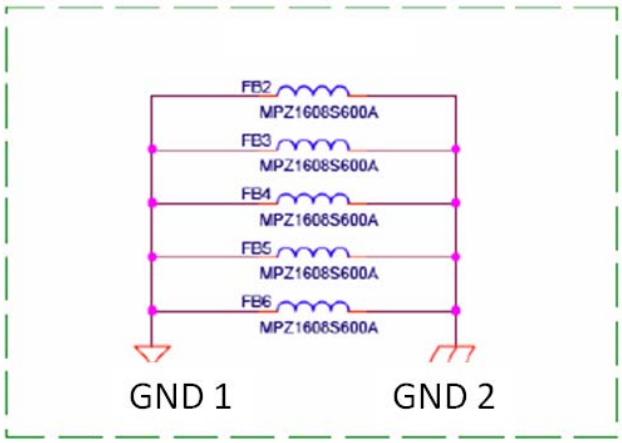

Figura 4.4: Filtrado entre los planos de masa de la FPGA de control (GND2) y la criptográfica (GND1)

Las medidas de corriente se realizan sobre la FPGA destinada a implementar los procesadores criptográficos. En la Figura 4.5 puede verse el esquema de los puntos de medida de que la placa dispone. En el circuito de alimentación y masa se incluyen resistencias de carga de 1 Ω (R1 y R2) para poder medir de forma indirecta la corriente que está consumiendo el núcleo lógico. También se incluyen pines de conexión (JP1 y JP2), en paralelo con dichas resistencia, bien para facilitar el uso de sondas de corriente bien para cortocircuitar las resistencias y eliminar su función.

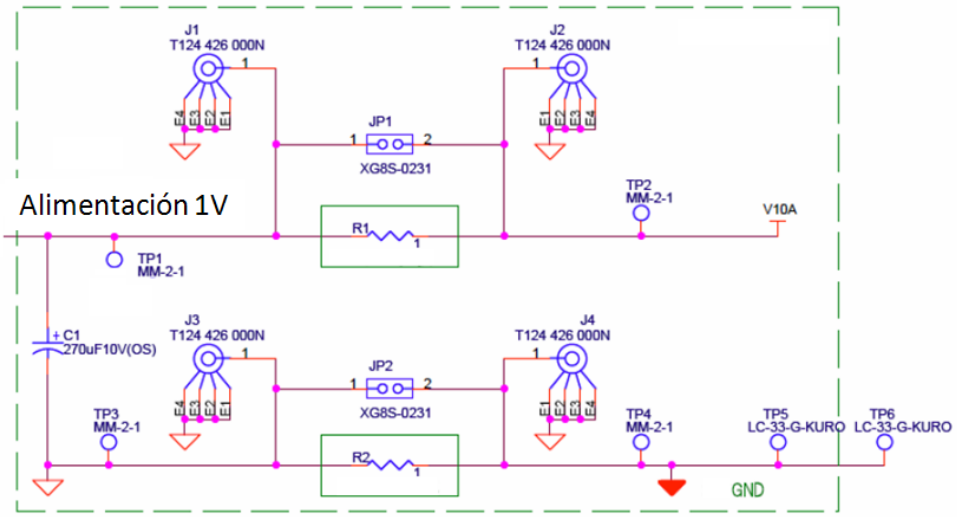

Figura 4.5: Esquema eléctrico del área de medidas de consumo.

4.3. Medida y captura de corriente

4.3.1. Sonda de corriente

La medida de consumo de corriente de la FPGA criptográfica se ha realizado mediante la sonda de corriente TEK CT-1 [TEK'2014] distribuida por la compañía TEKTRONICS[39] y que se muestra en la Figura 4.6.

Figura 4.6: Sonda de corriente TEK CT-1

Se trata de una sonda construida en base a un transformador de medida de corriente especialmente diseñada para ser usada en corrientes alternas y en un amplio margen de frecuencia. Al tratarse de un transformador de intensidad, la sonda no es sensible a la componente continua de la señal medida. Esto no representa un problema pues el coeficiente de correlación no se ve alterado por la adición o sustracción de valores constantes (Apartado 2.7.1).

Las características más destacables de esta sonda de corriente son las que se detallan en la Tabla 4.1:

Parámetro	**TEK – CT1**
Ancho de Banda	25kHz – 1 GHz
Sensibilidad	5 mV/mA
Precisión	± 3%

Tabla 4.1: Características más relevantes de la sonda de corriente TEX CT-1

[39] TEKTRONIX, INC. http://www.tek.com/

El ancho de banda es un parámetro importante para capturar las variaciones de corriente provocadas por la conmutación de los transistores de las celdas CMOS puesto se producen en los primeros picosegundos tras la misma [Man'2007].

4.3.2. Captura de datos

La sonda de captura se usa en combinación con un osciloscopio digital modelo DSO1024A [AGI'2008] fabricado por la compañía Agilent Technlogies[40] (Figura 4.7.).

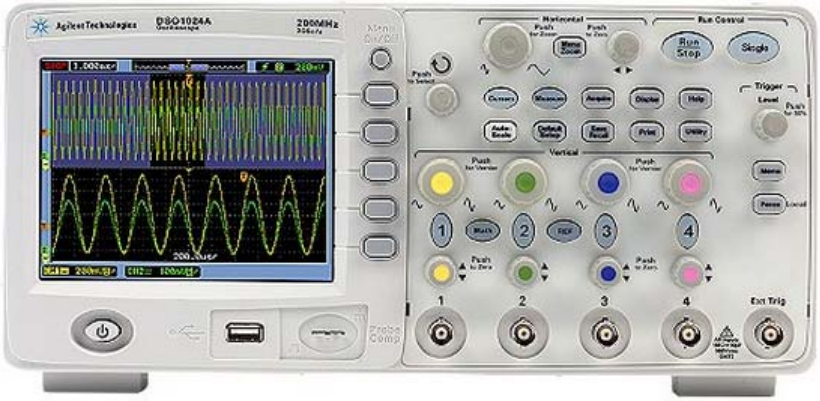

Figura 4.7: Osciloscopio Agilent DSO1024A

Se trata de un osciloscopio de 4 canales, con un ancho de banda de 200 MHz y una frecuencia de muestreo máxima de 2 Giga muestras por segundo. Hay que destacar que las máximas condiciones de trabajo se consiguen cuando el osciloscopio trabaja en modo de dos canales. Las prestaciones se reducen a la mitad si se trabaja con los 4 canales disponibles. Las principales características se pueden ver resumidas en la Tabla 4.2.

El osciloscopio tiene conectividad USB, lo le permite recibir instrucciones desde un ordenador para configurar sus condiciones de trabajo. Tras la captura, el quipo puede enviar a través del bus USB los datos capturados y los parámetros de captura hacia el equipo informático. Éstos son almacenados para su posterior procesado. Es posible acceder a todos los datos almacenados en la memoria de muestras del osciloscopio o descargar únicamente aquella sección que resulte relevante para el análisis.

[40] Agilent Technologies. http://www.agilent.com/home?cc=es (Actualmente Keysight Technologies)

Parámetro	DSO1024A
Ancho de Banda	200 MHz
Canales	4
Frecuencia de muestreo	2/1 Giga muestras por segundo (Modo 2/4 canales)
Profundidad de muestreo	20000 /10000 puntos (Modo 2/4 canales)
Resolución	8 bits

Tabla 4.2: Características más relevantes del osciloscopio Agilent DSO1024A

El fabricante proporciona los drivers y las librerías necesarias para establecer la comunicación entre el osciloscopio y el equipo informático al que está conectado [AGI'2009].

4.4. Configuración de equipos

Para la realización de la parte práctica se han configurado los equipos de la siguiente manera:

En la placa de desarrollo SASEBO-GII, la FPGA de control se ha cargado con una configuración mínima que permite la transmisión directa de los pines de comunicación R232 procedentes del driver USB FT2232D hasta la FPGA criptográfica. De la misma manera se procede con la señal de reloj que proporciona la placa. De esta forma la FPGA de control no toma parte activa en la ejecución de los algoritmos ni en la comunicación con ellos y se dejan esas funciones a la FPGA criptográfica.

Todas las implementaciones testeadas se han cargado íntegramente en la FPGA criptográfica y la frecuencia de trabajo del sistema ha sido de 24 MHz.

En cada caso se ha implementado una señal de sincronismo que se activa en el momento en que se inicia el procesado de la información. Esta señal se conduce al osciloscopio y se usa para disparar el inicio de la captura de datos. De esta forma se consigue que las trazas capturadas estén alineadas y su análisis no se vea alterado por la aparición de una fluctuación temporal.

El osciloscopio DSO1024A se ha configurado para trabajar con dos canales. El canal 2 se usa para la medición de la corriente de carga del

núcleo de la FPGA criptográfica y el canal 4 se usa para la monitorización de la señal de disparo que inicia el proceso de captura.

La frecuencia de muestreo se ajusta automáticamente en función de la duración de la captura realizada. Al concluir la captura de una traza se pueden leer todos o parte de los 20000 datos de 8 bits almacenados en la memoria del osciloscopio y que contienen la información relativa al consumo de la FPGA.

Puesto que la frecuencia de trabajo del sistema criptográfico es de 24MHz, y la frecuencia de muestreo máxima del osciloscopio es de 2 Gs/s[41], la cantidad máxima de puntos que pueden adquirirse en cada ciclo de reloj es de 83 muestra cada ciclo. Esta es una cantidad de muestras más que suficiente para el propósito que se persigue.

El equipo informático de captura está formado por un ordenador portátil SONY VAIO modelo VGN-SR49VN provisto de un procesador Intel® Core™2 Duo Processor P8800 que trabaja a una velocidad de 2.66 GHz. El equipo dispone de 3 GB de memoria RAM.

Todo el proceso de configuración de osciloscopio, captura y almacenamiento de trazas y procesado de datos se ha automatizado usando el lenguaje de computación matemática MATLAB[42] [Mat'2014] en combinación con las librerías aportadas por el fabricante del osciloscopio.

4.5. Proceso de captura.

4.5.1. Texto plano

La variable que el atacante controla totalmente es el texto plano que se envía al criptoprocesador para que sea encriptado. La adecuada elección del texto plano influirá notablemente en el éxito del ataque. En las pruebas se han utilizado tres archivos diferentes con objetivos claramente diferenciados.

[41] Giga "sample" por segundo, equivale a 10^9 muestras por segundo.

[42] The MathWorks, Inc. http://es.mathworks.com/help/matlab/ (Febrero 2015)

El primer archivo es un texto ASCII[43] conocido que se utiliza para verificar el correcto funcionamiento del sistema implementado. Se envía una determinada cantidad de bytes para que sean encriptados con una clave conocida. Seguidamente, los textos encriptados que se reciben del criptoprocesador se almacenan en un archivo. Finalmente, el texto cifrado se desencripta mediante el algoritmo AES implementado en MATLAB y del que se ha constatado su correcto funcionamiento. Si el sistema ha funcionado sin fallos debe poderse leer el texto original. En las pruebas realizadas el texto de control que se ha utilizado es una versión ASCII de "Don Quijote de la Mancha", puede verse el resultado del proceso en la Figura 4.8.

```
El ingenioso hidalgo don Quijote de la Mancha

TASA

Yo, Juan Gallo de Andrada, escribano de Cámara del Rey nuestro señor, de
los que residen en su Consejo


"ëøfƒ'\r□□□°é¼ß£□‡J¬¨¯I¾,B÷¿œ#1apód□"*¿Å.\Dp□%ftÄ~□§"ÔíYu¾›Ý□Å
LMJ'ÇvÅ&þ—-P"Ë“R;*~#
Ù¢DäG™Õä¾ÀŠí4->8Ä„□õÏÄ □³oãþƒ#...□7®ÜI•ÎIân/äÆüØ[£y£□—Î¾□d&ÒkwBY78Ó□Â pn
fŽþ×~
```

Figura 4.8: Ejemplo de encriptado de los 160 primeros caracteres de "Don quijote de la Mancha" A) Sin encriptar B) Encriptado mediante algoritmo AES.

El uso de archivos que contienen texto ASCII no es recomendable para la obtención de trazas en un análisis de consumo. Un byte puede tomar valores que van desde 0 a 255, si la información que se envía al criptoprocesador proviene de un archivo que contiene únicamente datos alfanuméricos la cantidad de valores diferentes que se está enviando al sistema se reduce a alrededor de 135. En la Figura 4.9 se representa la distribución de datos que existen en un fichero ASCII, se puede ver la especial incidencia del valor ASCII 32 que corresponde con el espacio.

[43] De las siglas en inglés "American Standard Code for Information Interchange"

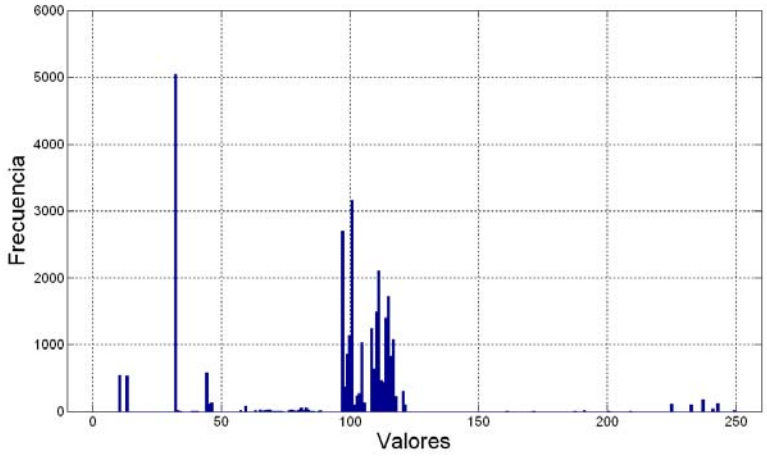

Figura 4.9: Distribución de datos en archivo de texto con 30000 caracteres ASCII.

En virtud de lo expuesto, se hace necesario trabajar con un segundo archivo que contiene datos uniformemente distribuidos en todos sus bytes. Este archivo se ha generado mediante un programa MATLAB y la distribución de valores obtenida se muestra en la Figura 4.10.

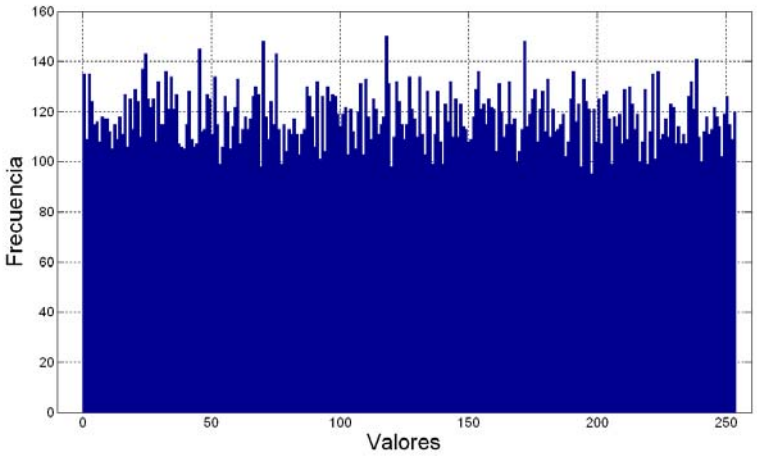

Figura 4.10: Distribución de datos en archivo con 30000 valores uniformemente distribuidos.

Paralelamente, la propia generación aleatoria de valores uniformemente distribuidos provoca que la sucesión de bytes que se envían en cada elemento del texto plano sea independiente del resto. La no existencia de correlación inicial entre los valores utilizados propicia la independencia entre el consumo generado en cada uno de los bytes, esto es especialmente útil cuando se procesan los datos en bloques de 16, 32, 64 o 128 bits.

La finalidad de este segundo archivo es atacar a la salida de la primera ejecución de la función *SubBytes* para conseguir los 16 Bytes de la clave en una única captura de trazas.

Finalmente, se ha generado un tercer lote de datos en el que el texto plano que se envía al sistema criptográfico es constante en todos sus bytes excepto en aquel que es objetivo del ataque. El byte variable se va incrementando secuencial y cíclicamente desde 0 hasta 255 para que tome todos los valores posibles. El objetivo de este archivo es realizar un ataque a la salida de la función *MixColumn* en la primera ronda tal y como se plantea en el apartado 3.2.2 [Lu'2009].

En este caso debe hacerse capturas de datos individuales para cada byte de la clave, puesto que los bytes constantes retornan valores constantes lo que propicia que el cálculo de la correlación no tenga sentido.

4.5.2. Secuencia de captura

La captura de trazas de consumo se automatiza a través del PC conectado al osciloscopio y al equipo de desarrollo. La secuenciación se realiza mediante del entorno de análisis de datos MATLAB y el proceso de captura es el siguiente:

a) Se leen 16 bytes del texto plano desde el archivo correspondiente.

b) Se envían los datos al criptoprocesador implementado en el equipo de desarrollo.

c) Una vez el sistema criptográfico recibe los 16 bytes que forman el estado inicial se procede a su encriptado. El dispositivo genera un pulso de sincronismo, de un ciclo de reloj de duración, cuya finalidad es disparar la captura de datos en el osciloscopio.

d) Mientras tiene lugar el procesado de la información, el osciloscopio lee los datos y los almacena en su memoria interna.

e) Una vez concluido el encriptado, el texto resultante se recibe en el PC y es almacenado en un archivo específico en el disco duro.

f) Finalmente se leen los datos de captura almacenados en el osciloscopio a través del puerto USB. Estos datos se almacenan en la memoria del ordenador y posteriormente se procederá a su análisis.

g) El proceso se repite con tantos textos planos diferentes como sea necesario.

En la Figura 4.11 puede verse el resultado de la captura de una traza de corriente sobrepuesta a la señal de sincronismo generada por el criptosistema. En la Figura 4.12 se aprecia el resultado obtenido de la captura de 50 trazas consecutivas obtenidas mientras se procesan 50 textos planos diferentes.

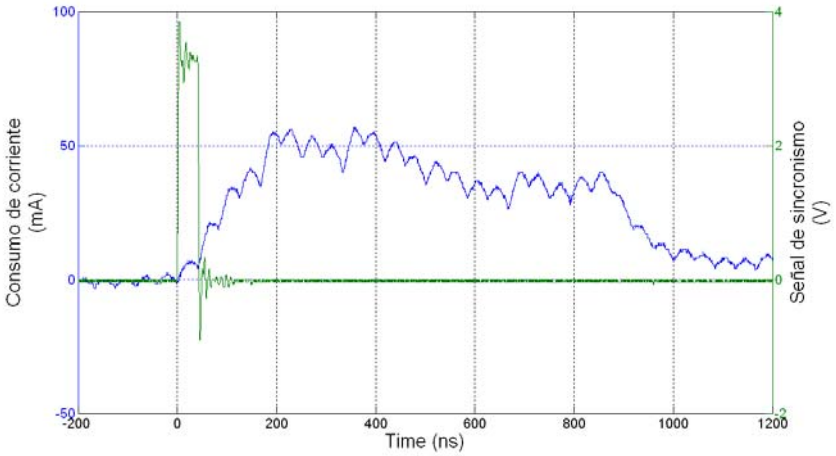

Figura 4.11: Traza capturada (azul) junto a la señal de sincronismo generada (verde).

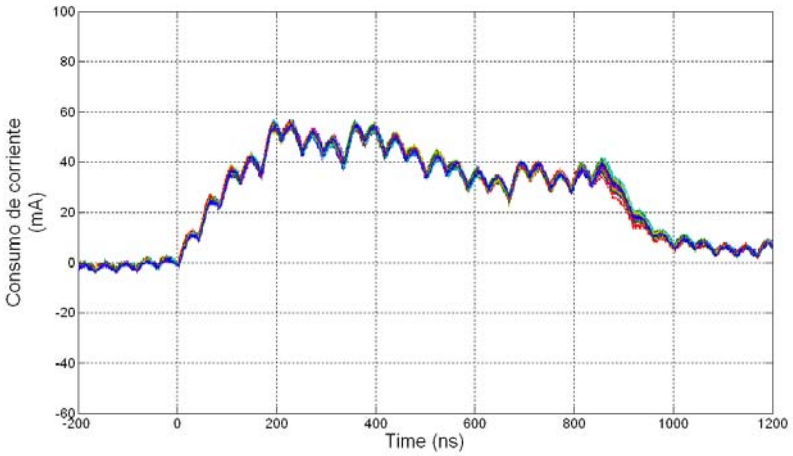

Figura 4.12: Juego de 50 trazas obtenidas durante el procesado de 50 textos planos.

El tiempo necesario para capturar una traza está claramente determinado por la cantidad de datos que deban leerse desde el osciloscopio. Un captura con 3500 puntos puede hacerse en 0.5 segundos por traza, mientras que una captura con 15000 necesitará 1.57 segundos por traza para completarse.

4.5.3. Integridad de las trazas

La cantidad de ruido en las trazas altera notablemente la fiabilidad del ataque criptográfico (Apartado 2.7.2). Es necesario evaluar el ruido que puede introducirse en la cadena de medida y valorar el posible efecto que éste pueda tener sobre los resultados obtenidos.

Para realizar esta valoración se han tomado 1000 trazas consecutivas en las que se procesa el mismo texto plano. Puesto que los datos procesados son los mismos, el consumo también ha de serlo, y por tanto, podemos considerar que las diferencias entre las capturas corresponden al ruido que se haya introducido en la cadena de medida. Se toma el valor medio de las 1000 trazas como referencia para el cálculo y la diferencia entre cada una de las trazas con la referencia representa el ruido existente en la medida. En la Figura 4.13 aparecen la traza media usada como referencia y el valor máximo del ruido observado en cada punto de la misma.

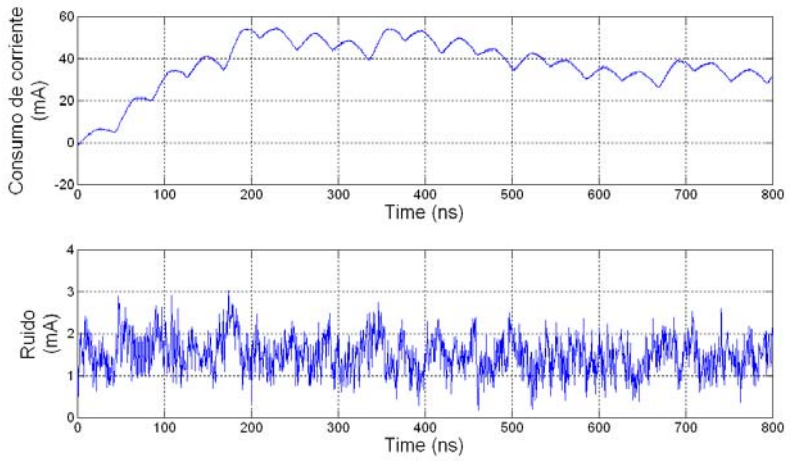

Figura 4.13: Gráfica de valoración del ruido en las medidas.

De la ecuación (2.6) obtenemos el valor de la relación señal ruido y de la ecuación (2.14) el porcentaje de reducción producido por la presencia de ruido. Los resultados pueden verse en la Tabla 4.3.

A la luz de los resultados obtenidos puede asegurarse que el ruido procedente de la cadena de medida no alterará el resultado del análisis de consumo, ya que la correlación se verá reducida en menos de un 1.8 % a causa del mismo.

Parámetro	**Valor**
Varianza de la señal	4.49
Varianza del ruido	0.17
SNR	26.41
Correlación máxima esperada	98.2 % del valor teórico [Man'2007]

Tabla 4.3: Resultados del cálculo de la SNR

4.6. Procesado de las trazas.

Tras la captura de las trazas de consumo debe procederse a su procesado. La finalidad es extraer la información que llevará al atacante a la obtención de las claves criptográficas con las que trabaja el sistema.

En esta parte del proceso es necesario trabajar con gran cantidad de datos. Por un lado están todos los puntos medidos en cada traza de corriente, y por otro están los valores hipotéticos que forman el modelo de consumo. Finalmente, debe ser correlacionado un conjunto de valores con el otro.

Es habitual efectuar un procesado de las trazas previo al cálculo de la correlación. El objetivo es reducir la cantidad de muestras obtenidas a valores más manejables y que el tiempo necesario para el cálculo se reduzca lo más posible.

[Man'2007] propone dos formas de reducir la cantidad de valores que forman la traza:

a) Extracción de máximos: De los puntos capturados en cada ciclo de reloj se toma el que mayor valor muestra y se desprecian los demás.

b) Integración: Se suma el valor de todos los puntos dentro de un intervalo de tiempo definido. Suele usarse un intervalo que coincide con el ciclo de reloj del sistema, aunque se obtienen buenos resultados usando intervalos más grandes.

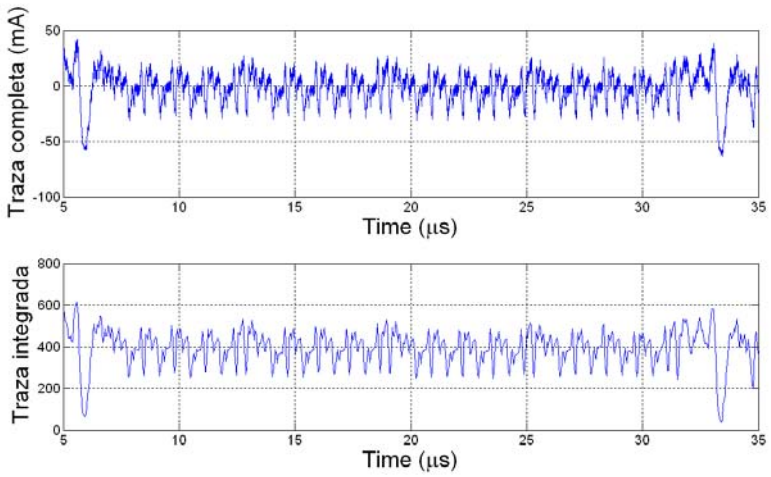

Figura 4.14: Comparación entre una traza completa (superior) y la misma traza integrada por ciclos de reloj (inferior).

En esta tesis se ha optado por el segundo de los métodos ya que en la traza comprimida hay información de todo el periodo capturado. En cambio, si se utiliza la extracción de máximos parte de la información del consumo es eliminada. En la Figura 4.14 puede verse la diferencia entre una traza completa y la misma traza integrada con un tiempo de integración de un ciclo de reloj. En este caso la cantidad de puntos se reducen a la quinta parte del total.

4.7. Presentación de resultados

El análisis se realiza buscando la correlación entre las trazas y cada una de las 256 hipótesis de clave que forman el modelo de consumo. El resultado del análisis es un juego de 256 trazas de correlación, de las cuales, aquella que muestre una correlación mayor corresponderá a la hipótesis de clave que coincide con la clave del dispositivo criptográfico.

Puede verse el resultado de un análisis en la Figura 4.15. En él, la correlación obtenida cuando el valor de la clave del modelo coincide con la realidad es considerablemente mayor que cuando no se produce dicha coincidencia. Incluso en valores muy próximos la correlación es manifiestamente mayor y por tanto diferenciable del resto de valores hipotéticos.

En la Figura 4.16 aparece el mismo análisis pero sobrepuesto sobre el grupo de trazas que se han utilizado para realizarlos. La gráfica muestra el instante preciso en el que se produce la perdida de información y la corre-

lación obtenida en ese punto. Puede apreciarse que, las hipótesis no correctas muestran un valor máximo no superior a 0.2 mientras que la hipótesis correcta alcanza una correlación de 0.8.

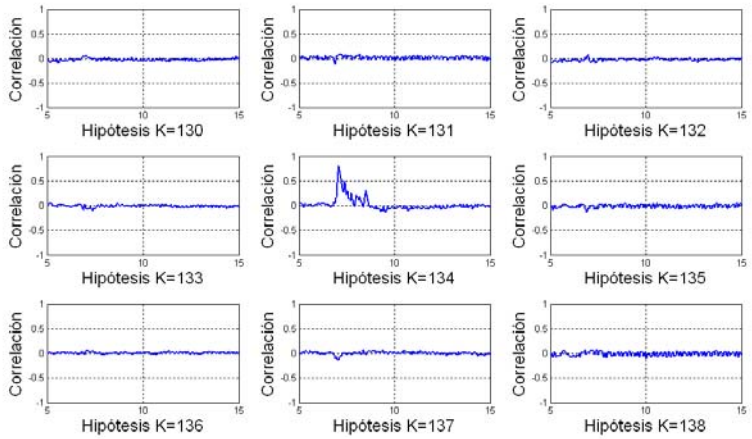

Figura 4.15: Correlaciones calculadas a lo largo de la traza con diversas hipótesis de clave. En el centro aparece la hipótesis correcta k=134.

El tiempo de cálculo necesario para la realización de este análisis de consumo ha sido de 3.89 segundos. Hay que tener en cuenta que el análisis se ha realizado con 500 trazas y que estas han sido previamente integradas para reducir el número de puntos que las forman.

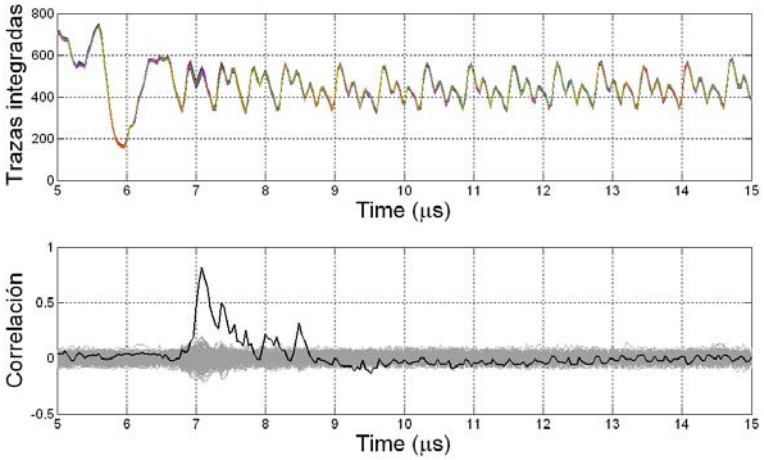

Figura 4.16: Análisis de correlación sobre 1000 trazas capturadas.

En primer lugar, los resultados se presentan en forma de análisis temporal de la traza. Si se ha realizado un análisis por diferencia de medias (Apartado 2.8) el eje de abscisas muestra el tiempo transcurrido desde el inicio de la toma de datos y el eje de ordenadas muestra la diferencia del

consumo medio. En la Figura 4.17 se muestra el resultado de un análisis de este tipo.

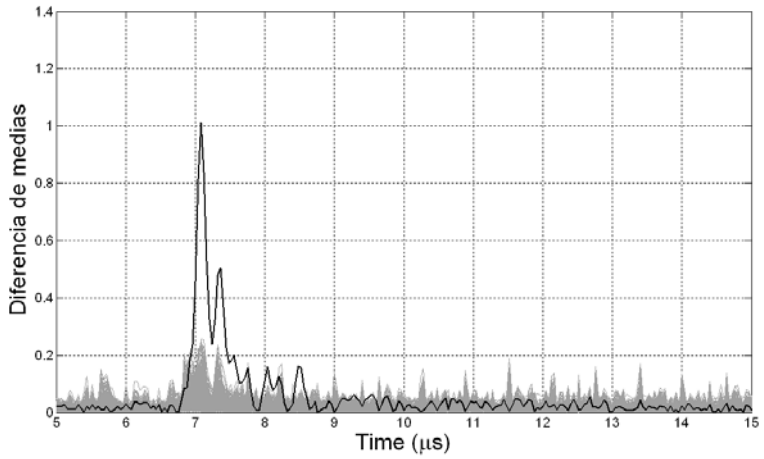

Figura 4.17: Presentación del resultado de análisis por diferencia de medias.

Si el análisis practicado ha sido un análisis de correlación el eje de ordenadas muestra el valor de la correlación en cada punto tal y como se puede ver en la Figura 4.18.

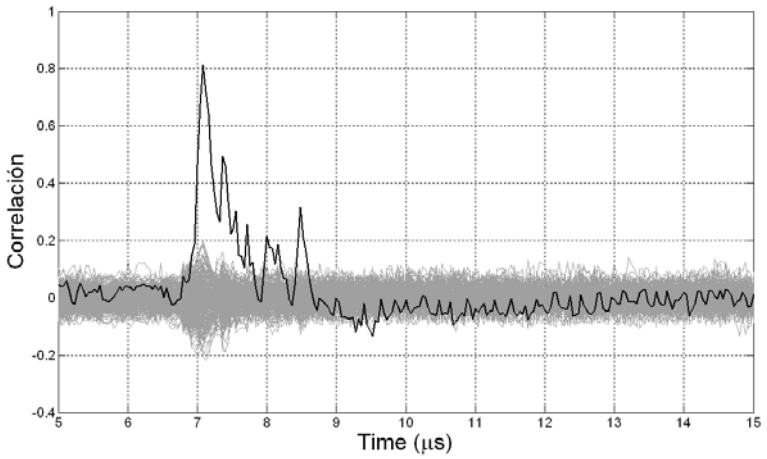

Figura 4.18: Presentación del resultado de análisis por correlación del consumo.

En segundo lugar, el resultado de correlación se presenta en función de las trazas tomadas. Esta gráfica indica la cantidad de trazas mínima que deben tomarse para poder diferenciar la hipótesis correcta y nos da una indicación gráfica de la vulnerabilidad del sistema.

En la Figura 4.19 aparece el resultado de este análisis y podemos comprobar que es posible diferenciar la hipótesis correcta tomando un mínimo de 40 trazas.

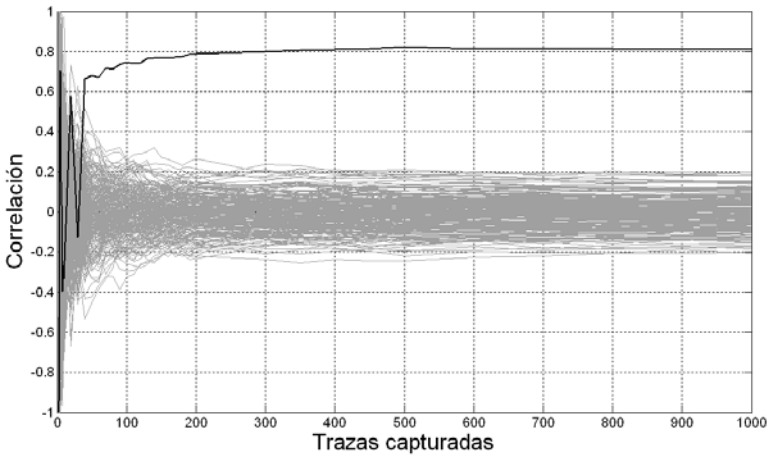

Figura 4.19: Correlación en función del número de trazas capturadas.

4.8. Conclusiones

La plataforma de desarrollo utilizada permite experimentar contramedidas contra ataques por análisis de consumo de forma rápida y sin la obligación de tomar un gran número de trazas para verificar los resultados. La práctica ausencia de ruido en las trazas tomadas y la velocidad del reloj del sistema así lo propician. Del mismo modo, la disposición de las conexiones permite que la toma de medidas de corriente de consumo de la lógica interna de la FPGA criptográfica pueda realizarse con absoluta facilidad. No obstante, se trata de una plataforma de desarrollo ampliamente utilizada en este tipo de investigación, tanto el modelo utilizado en esta tesis como otros modelos producidos por la misma compañía [Tos'2010] [Yoh'2013] [Dee'2014].

5. Resultados experimentales

5.1. Introducción

En este capítulo se mostrarán los resultados obtenidos de la implementación de la contramedida propuesta en esta tesis frente a los ataques por análisis de consumo. La implementación se ha realizado en tres supuestos:

a) Sistema software: Un microprocesador ejecuta el algoritmo y la contramedida en un entorno puramente software.

b) Sistema hardware/software: Un microprocesador ejecuta el algoritmo en un entorno software mientras que la contramedida se ejecuta en un coprocesador específico.

c) Sistema hardware: Se prescinde del microprocesador y se implementa el algoritmo y la contramedida en un entorno totalmente hardware.

Para la verificación se ha estudiado el comportamiento del algoritmo sin ninguna contramedida; esta versión se usa como prueba de control para medir la vulnerabilidad del sistema. Posteriormente, se ha implementado la contramedida y se han repetido las mismas medidas para comprobar su correcto funcionamiento.

5.2. Sistema software de referencia

5.2.1. Implementación del sistema

La implementación se ha realizado en base al microprocesador de 32 bits “Soft Core” Microblaze V8.10a proporcionado por Xilinx y optimizado para sus dispositivos FPGA.

Este procesador es sensible a los ataques por análisis de consumo [Lum'2013] y permitirá valorar el funcionamiento de la contramedida.

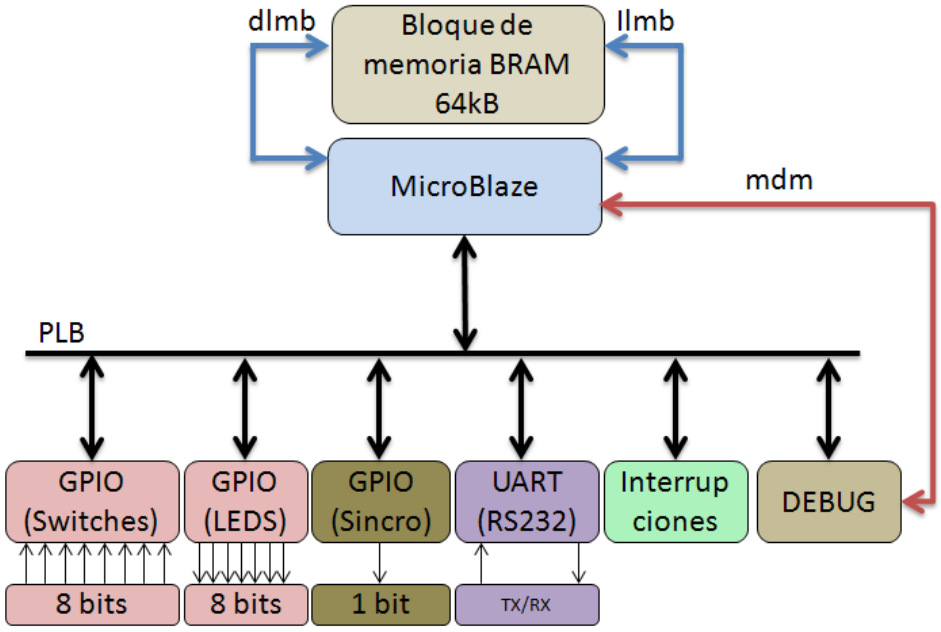

Figura 5.1:Diagrama de bloques del sistema para pruebas software.

El sistema implementado se muestra en la Figura 5.1. Los bloques incluidos en el sistema son::

a) Procesador Microblaze: Se ha implementado el modelo de 5 etapas “pipeline”. No se ha incluido unidad FPU[44] puesto que no debe realizar operaciones en coma flotante, tampoco se ha incluido la unidad “barrel shifer” ni el multiplicador de 32 bits.

b) Memoria BRAM interna: Se utilizan 64kB de memoria RAM en el sistema, esto corresponde a 16 de las 48 unidades BRAM que ofrece la FPGA [XIL'2010c]. Este espacio de memoria es suficiente para

[44] De las siglas en inglés “Float Point Unit”

las pruebas que se realizarán, considerando que se almacenarán en RAM interna tanto el programa como los datos.

El procesado y el resto de dispositivos se comunican a través de un bus local PLB[45] versión 1.05.a [XIL'2010]. Los elementos conectados al bus son:

a) Unidad GPIO con 8 bits de entrada conectados a otros tantos interruptores. Su utilidad queda reducida al desarrollo del software y se utiliza para funciones auxiliares.

b) Unidad GPIO[46] con 8 bits de salida conectados 8 indicadores LED. Se utiliza para verificar el funcionamiento de partes del programa, especialmente durante el desarrollo.

c) Unidad GPIO con 1 bit de salida, esta es la señal de sincronismo que se usa para sincronizar la captura de trazas con el proceso de encriptado del AES.

d) Unidad UART[47], su misión es permitir la comunicación vía RS232 [XIL'2010b] con el PC, se usa para enviar textos plano y recibir el texto cifrado.

Parámetro	**Valor**
Frecuencia de reloj máxima	161 MHz
"Slices" consumidos	1256 de 7200 17.4%
"LUTS" consumidos	2264 de 28800 7.9%
"FF" consumidos	2006 de 28800 7.0%
36K BRAMS	16 de 48 33.3%

Tabla 5.1: Recursos de la FPGA consumidos por el sistema software

e) Gestor de interrupciones, con este módulo se monitoriza el puerto RS232 y se activa la interrupción cada vez que un carácter es recibido.

f) Módulo de depuración, conectado al procesador mediante el bus MDM[48] que permite la depuración de los programas desarrollados.

[45] De las siglas en inglés "Processor Local Bus"

[46] De las siglas en inglés "General-purpose input/output"

[47] De las siglas en inglés "Universal Asynchronous Receiver-Transmitter"

Los datos de implementación del sistema se muestran en la Tabla 5.1.

5.2.2. Programación del algoritmo AES

La programación se ha realizado partiendo de la versión programada en "C" propuesta por [Niy'2007], sobre la cual se han efectuado varias modificaciones para adaptarla a la plataforma de trabajo.

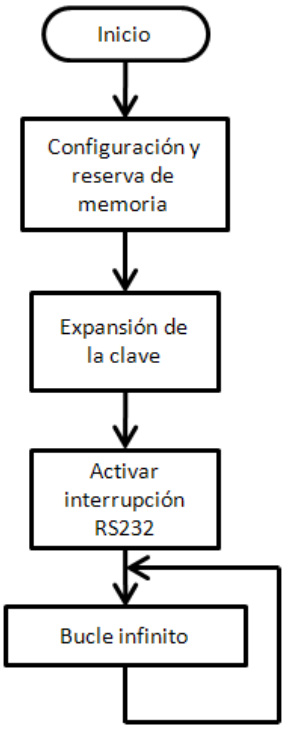

Figura 5.2:Diagrama de flujo general.

El diagrama de flujo general del programa puede verse en la Figura 5.2 y se compone de los siguientes procesos:

a) Primeramente se configuran los diferentes periféricos del sistema: GPIO de test, GPIO de sincronismo y unidad UART.

b) En segundo lugar se reserva la memoria para las variables propias del programa: Texto plano, Clave, Clave expandida, Estado, Texto encriptado; y para las tablas que el algoritmo necesita: tablas SBOX y RCON.

[48] De las siglas en inglés "Microblaze Debug Module"

c) Seguidamente se calcula la clave expandida a partir de la clave de encriptado fijada en el código del programa.

d) A continuación se configura el registro de interrupciones para activar aquella que está relacionada con la recepción de un byte a través del puerto RS232.

e) Finalmente el programa entra en un bucle infinito a la espera de que se reciban datos a través del puerto serie.

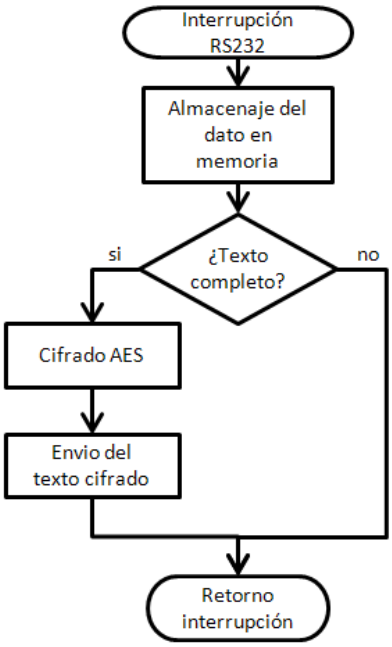

Figura 5.3:Diagrama de flujo de la función de interrupción.

La interrupción de recepción de datos a través del RS232 se activa cada vez que se recibe un dato. La misma rutina almacena este dato en la memoria y verifica si se han recibido los 16 datos que forman el texto plano de trabajo. En el momento en que el texto plano está completo se inicia el cifrado y, una vez concluido, el texto cifrado retorna hacia el PC para que sea almacenado. La Figura 5.3 representa el diagrama de flujo de la interrupción.

Finalmente el algoritmo AES se estructura según el esquema general del mismo [NIST'2001], siguiendo diagrama de flujo mostrado en la Figura 5.4.

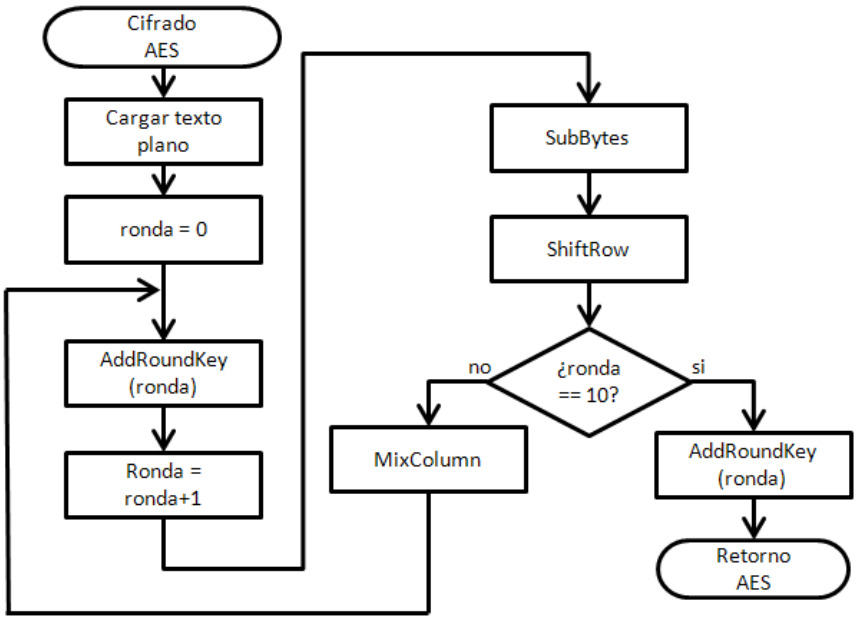

Figura 5.4:Diagrama de flujo del algoritmo AES.

El tiempo necesario para el procesado de un ciclo de encriptado completo y la memoria RAM ocupada por el programa son los que se muestra en la Tabla 5.2.

Parámetro	**Valor**
Tiempo de ejecución AES	1.14 ms
Memoria ocupada	11770 bytes

Tabla 5.2: Datos de ejecución y compilación.

5.2.3. Captura de trazas.

Para la sincronización de la captura de datos se activa la salida digital correspondiente justo antes de iniciarse la parte del algoritmo que se pretende calcular. El diagrama de flujo de la Figura 5.5 incorpora la ubicación de los procesos de activación y parada del bit de sincronismo cuando se pretende capturar el consumo relacionado con el procesado de la función *SubBytes*. El osciloscopio se configura para que capture un único disparo de la señal en cada ejecución del algoritmo; de esta forma se obtiene la traza de corriente de procesado de la ejecución de la función en la primera ronda del algoritmo.

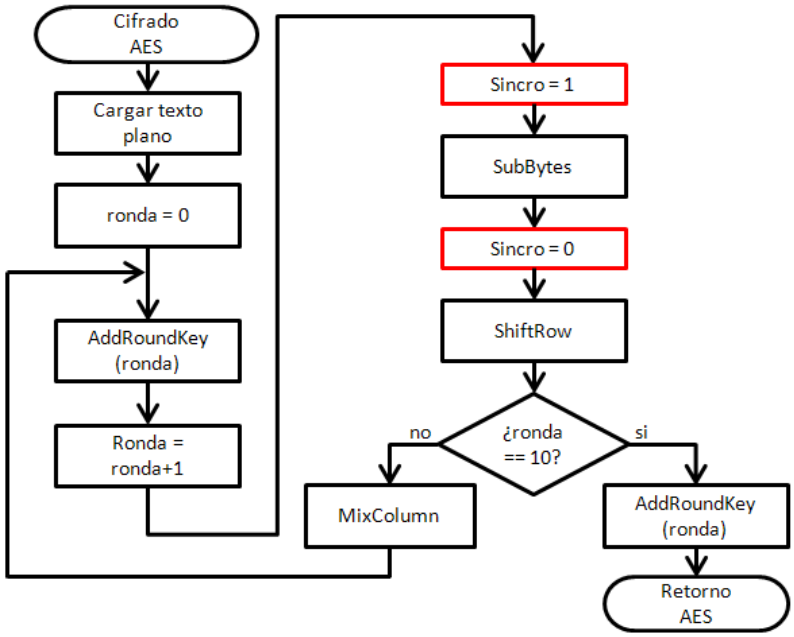

Figura 5.5:Diagrama de flujo del algoritmo AES con los procesos de sincronismo.

Los bloques de sincronismo deben ser ubicados antes y después de la función que pretenda ser capturada. Posteriormente, el osciloscopio se configura para capturar el tramo elegido con la máxima resolución posible y se desprecia el resto de la traza.

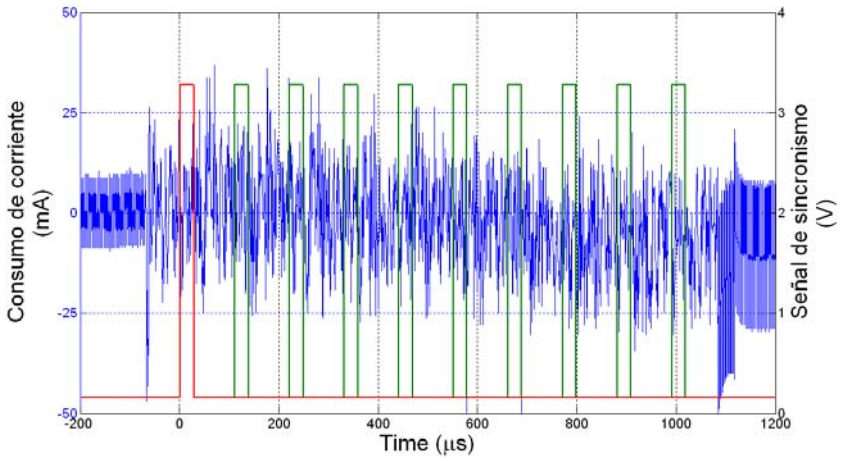

Figura 5.6:Consumo del procesador y señal de sincronismo en una ejecución del algoritmo AES.

El resultado del proceso de captura se muestra en la Figura 5.6. El consumo aparece en azul, las señales de sincronismo en verde y en rojo el tramo de traza que se captura y que posteriormente servirá para los análisis.

Las condiciones de la captura utilizadas para el análisis de las implementaciones software son las que se muestran en la Tabla 5.3.

Parámetro	**Valor**
Frecuencia del sistema	24 MHz
Frecuencia de muestreo	125 Ms/s (5,2 muestra/clk)
integración	≈1 ciclo de reloj (5 muestras)
Tiempo de captura medio	727 s /1000 trazas
Tiempo de integración	19.7 s /1000 trazas
Tiempo de cálculo de la correlación	5.2 s/1000 trazas integradas · byte
Tiempo de cálculo, diferencia de medias	54 s/1000 trazas integradas · byte

Tabla 5.3: Parámetros de captura para sistema software.

5.2.4. Ataque a la salida de la función *AddRoundKey*.

Usando el procedimiento anterior se ha realizado el ataque a la salida de la función *AddRoundKey* en la ronda 0. Dado que esta es una función lineal, las trazas de consumo de las hipótesis cercanas a la clave real serán muy parecidas y, por tanto, las diferencias de correlación muy pequeñas. No obstante puede ser diferenciada la hipótesis correcta de las que no lo son.

Para este ataque se utiliza un texto plano en el que se envían valores diferentes uniformemente distribuidos a todos los bytes del estado (Apartado 4.5.1).

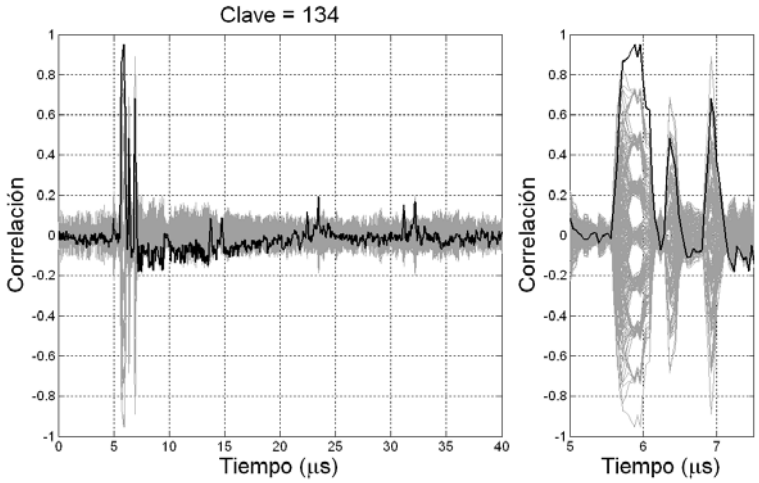

Figura 5.7:Análisis de correlación del sistema software no protegido. Atacado el byte 0 tras la función *AddRoundKey* en la ronda 0.

La Figura 5.7 muestra el resultado del análisis de correlación. En ella puede verse diferenciada la clave correcta, que coincide con la hipótesis de máxima correlación. El valor máximo de la correlación es de 0.95, fácilmente diferenciable del siguiente valor que es 0.86.

En la Figura 5.8 puede distinguirse como a partir de 30 trazas capturadas la correlación de la hipótesis correcta ya es diferenciable del resto.

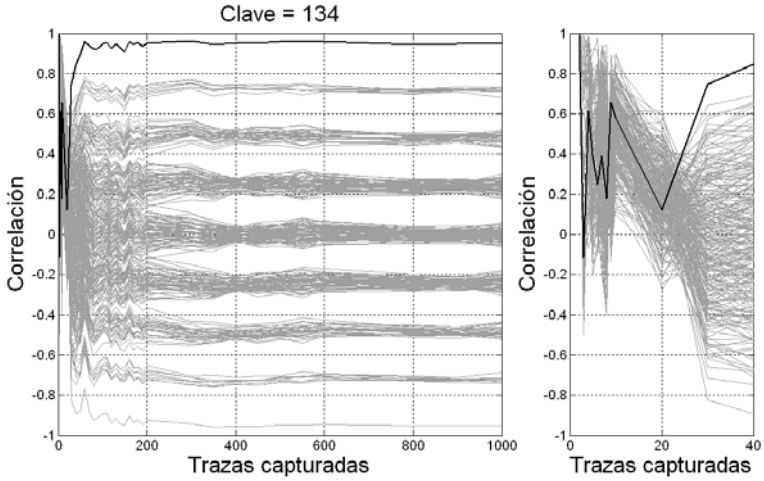

Figura 5.8:Análisis de correlación del sistema software no protegido. Atacado el byte 0 tras la función *AddRoundKey* en la ronda 0. Representación en función de las trazas capturadas.

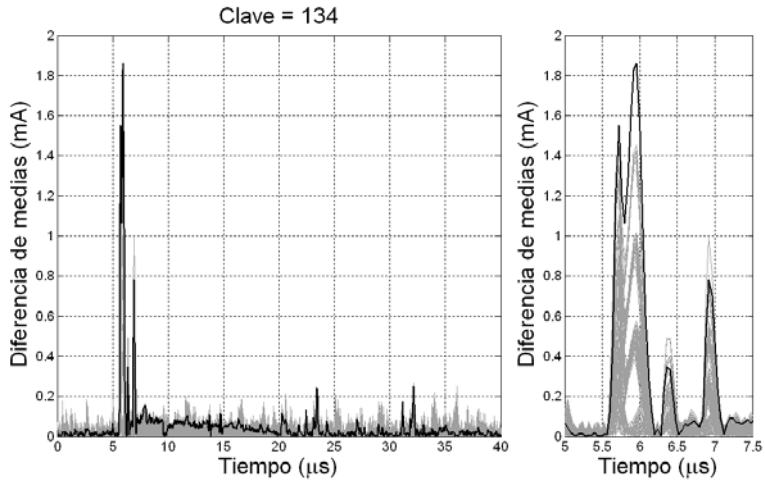

Figura 5.9:Análisis por diferencia de medias del sistema software no protegido. Atacado el byte 0 tras la función *AddRoundKey* en la ronda 0.

También se ha realizado el análisis por diferencia de medias, el resultado del cual puede verse en la Figura 5.9. En este caso la diferencia de

medias máxima obtenida tiene un valor de 1.86 mA y es mayor que la siguiente que aparece con un valor 1.45 mA.

5.2.5. Ataque a la salida de la función *SubBytes*

En las mismas condiciones que en el apartado anterior se capturan 1000 trazas a la salida. La no linealidad de la función *SubBytes* provoca que la variación de un bit a la entrada no se corresponda con una variación de un bit a la salida. De este modo, la diferenciación de la clave es mucho más clara, dado que la correlación máxima tendrá un valor considerablemente más alto que el resto de correlaciones. Concretamente la máxima es 0.82, en cambio el siguiente valor es de 0.22. El resultado se muestra en la Figura 5.10.

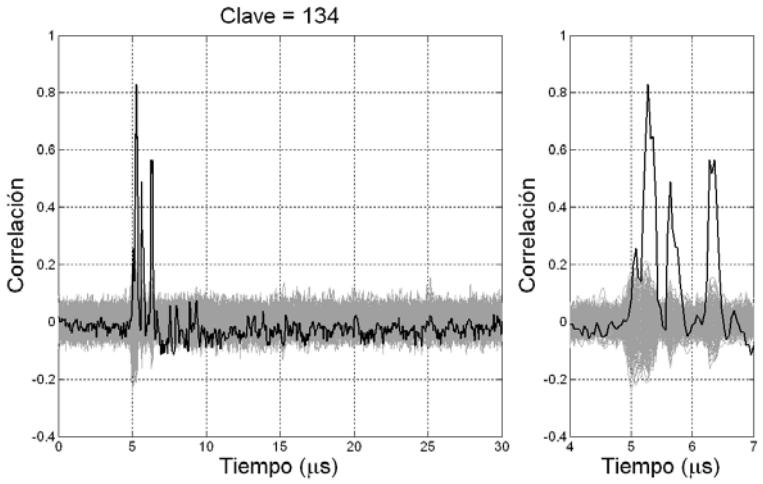

Figura 5.10:Análisis de correlación del sistema software no protegido. Atacado el byte 0 tras la función *SubBytes* en la ronda 1.

En la Figura 5.11 se aprecia como la clave correcta puede diferenciarse a partir de la captura de 50 trazas.

Finalmente, en el análisis por diferencia de medias los resultados también son claros. La máxima diferencia tiene un valor de 1.22 mA mientras que la siguiente en valor alcanza únicamente los 0.35 mA. (Véase la Figura 5.12)

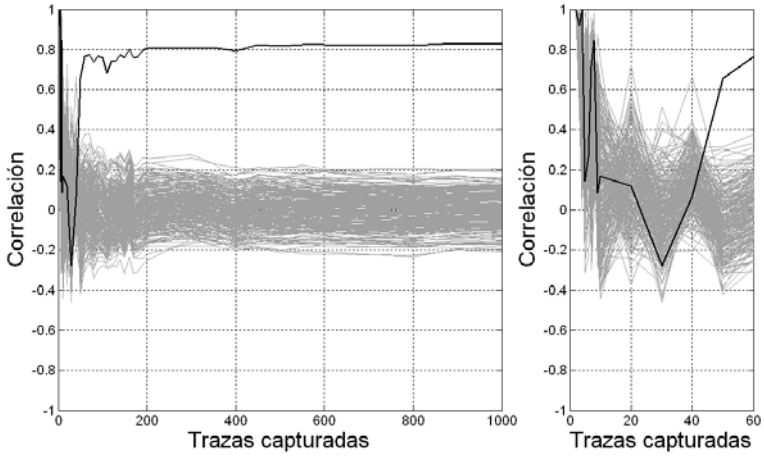

Figura 5.11:Análisis de correlación as del sistema software no protegido. Atacado el byte 0 tras la función *SubBytes* en la ronda 1. Representación en función de las trazas capturadas.

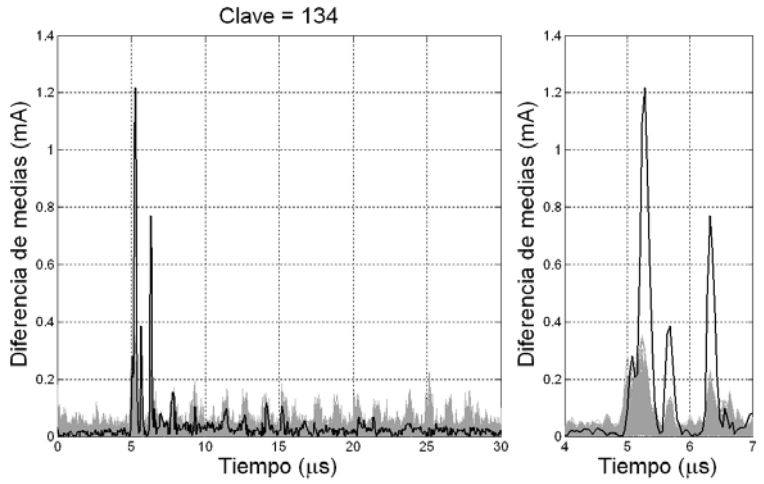

Figura 5.12:Análisis por diferncia de medias del sistema software no protegido. Atacado el byte 0 tras la función *SubBytes* en la ronda 1.

5.2.6. Ataque a la salida de la función *MixColumns*.

En el ataque dirigido a este punto se utiliza un texto plano en el que todos sus bytes son constantes a excepción del byte que se pretende atacar (Apartados 3.2.2 y 4.5.1).

Esta técnica de ataque requiere de una captura de trazas diferente para cada byte de la clave [Lu'2009]. El modelo de consumo a utilizar es el resultado del producto de Galois (GFx2 o GFx3) descrito en el algoritmo AES.

La Figura 5.13 muestra el resultado del análisis de correlación usando el modelo GFx3. La hipótesis coincidente con la clave correcta presenta una correlación de 0.47, que es perfectamente diferenciable del resto de hipótesis.

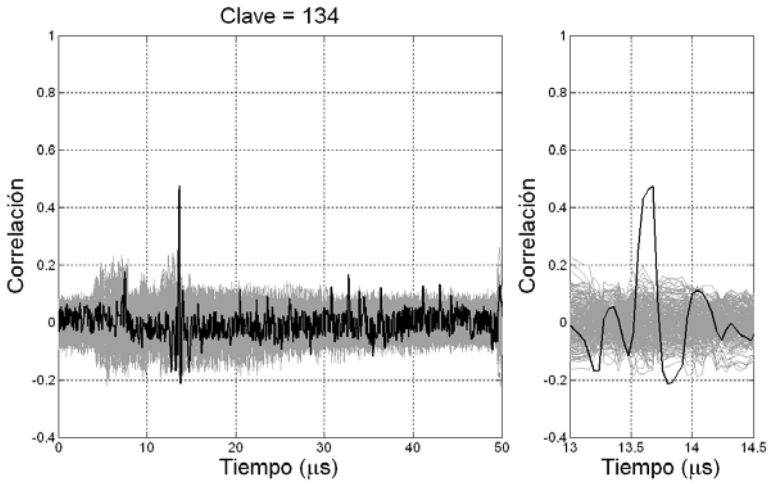

Figura 5.13:Análisis de correlación del sistema software no protegido. Atacado el byte 0 tras la función *MixColumns* en la ronda 1.

Tal y como muestra la Figura 5.14, en el análisis en función de las trazas se obtiene la clave correcta a partir de la captura de 130 trazas.

Y la diferencia de medias, presentada en la Figura 5.15, muestra un resultado claro mediante el uso del modelo GFx2. La diferencia máxima alcanza una valor de 2.37 mA, que es claramente superior al resto.

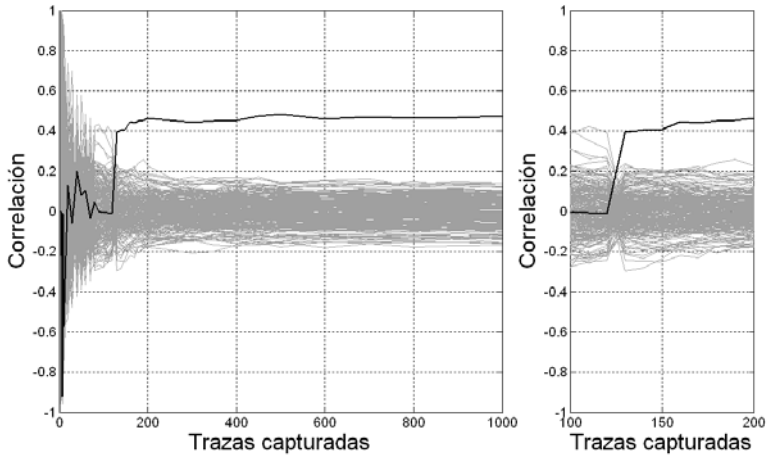

Figura 5.14:Análisis de correlación del sistema software no protegido. Atacado el byte 0 tras la función *MixColumns* en ronda 1. Representación en función de las trazas capturadas

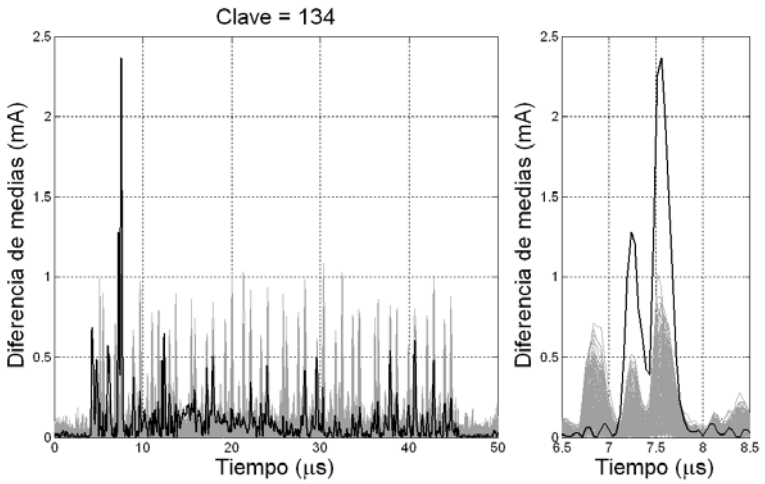

Figura 5.15:Análisis por diferencia de medias del sistema software no protegido. Atacado el byte 0 tras la función *MixColumns* en la ronda 1.

5.3. Resultados de la implementación de la contramedida en software.

La contramedida se ha implementado en software utilizando la misma plataforma que en la versión de referencia descrita en el apartado 5.2 (Figura 5.1).

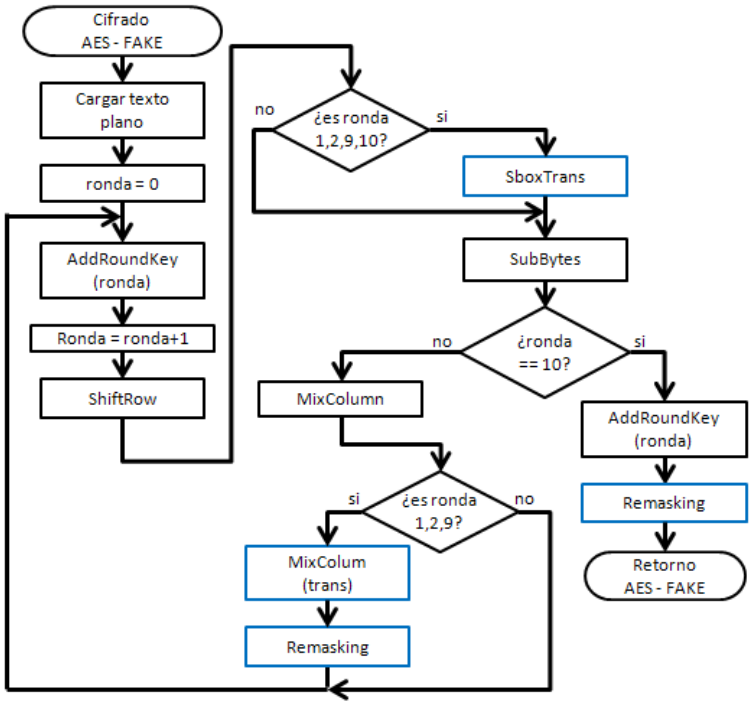

Figura 5.16:Diagrama de flujo del algoritmo AES con la contramedida implementada.

Aplicar la contramedida implica incorporar algunos procesos dentro del flujo del programa del algoritmo. El diagrama correspondiente puede apre-

ciarse en la Figura 5.16. En él se muestran en color azul aquellos bloques que implementan la contramedida. Atendiendo a la dispersión de la clave en el algoritmo (Tabla 3.1), atacar más allá de la ronda 2 equivale a atacar por fuerza bruta toda la extensión de la clave. Por esta razón no se aplica la contramedida en las rondas intermedias del algoritmo, con el objeto de conseguir una ejecución más rápida.

Por otro lado, el precálculo de las tablas $SBOX_{TRANS}$ (Apartado 3.4) se ubica fuera del algoritmo. Una vez se ha concluido el procesado y se ha enviado el texto plano a la unidad UART, se procede a la selección de las cuatro máscaras aleatorias de la matriz M_J, al cálculo de las cuatro máscaras de la matriz M_K y al precálculo de las mencionadas tablas. Estos serán los valores que se utilizaran en el siguiente ciclo de encriptado. Este precálculo se produce de forma concurrente al proceso de comunicación, los datos de la Tabla 5.4 muestran que la comunicación requiere un tiempo superior al pracálculo, en consecuencia las tablas estarán disponibles en la memoria antes del siguiente ciclo de encriptado.

Parámetro	**Valor**
Tiempo de ejecución AES-FAKE	1.59 ms
Tiempo de precálculo de tablas SBOXTRANS	1.90 ms
Tiempo de transmisión y recepción de datos (a 115200 bps)	2.78 ms
Memoria ocupada	19094 bytes

Tabla 5.4: Datos de ejecución y compilación del AES-FAKE software.

La captura de trazas se realiza mediante el método descrito en el apartado 5.2.3, así como el procesado de las trazas y el cálculo de las correlaciones. Para poder efectuar una comparativa válida se atacará a los mismos puntos y mediante la misma técnica que en el caso del sistema de referencia.

La representación de los resultados indica mediante una línea de color negro la correlación o diferencia de medias obtenida para la clave real y mediante una línea de color azul la correlación obtenida para la clave falsa.

En la Figura 5.17 se muestra el diagrama de bloques de la interrupción en el que se ha añadido el correspondiente bloque.

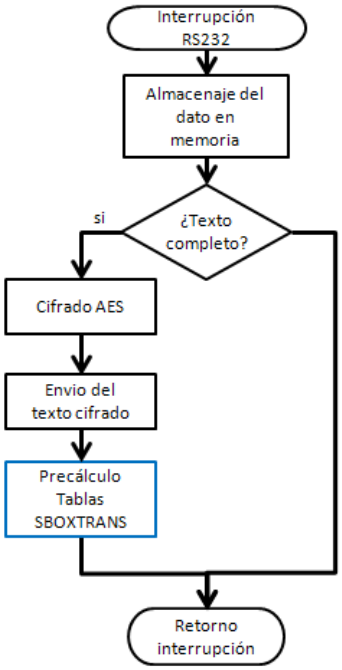

Figura 5.17:Diagrama de flujo de la función de interrupcoón con el bloque de calculo de las tablas SBOXTRANS

5.3.1. Ataque a la salida de la función *AddRoundKey*.

Los resultados de este apartado son comparables con los resultados del apartado 5.2.4 en el que se ataca el mismo punto del sistema de referencia. En la Figura 5.18 se aprecia como la máxima correlación se produce en la clave falsa con un valor de 0.95. En cambio, la clave correcta queda protegida entre los resultados de todas las demás claves.

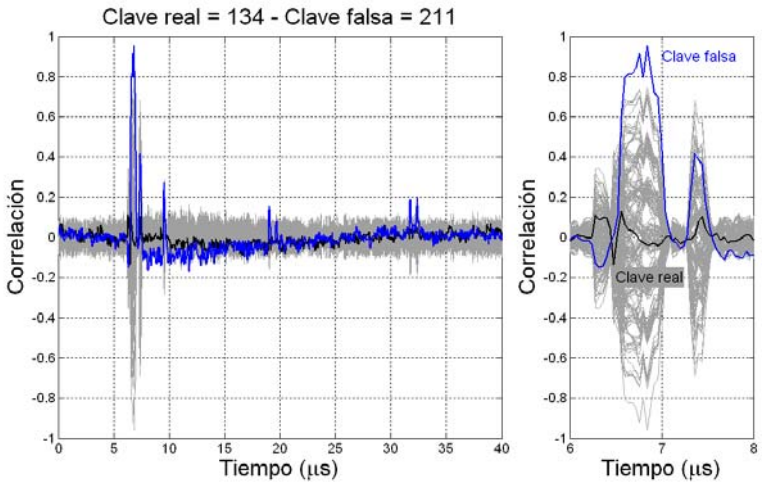

Figura 5.18:Análisis de correlación del sistema software protegido. Atacado el byte 0 tras la función *AddRoundKey* en la ronda 0.

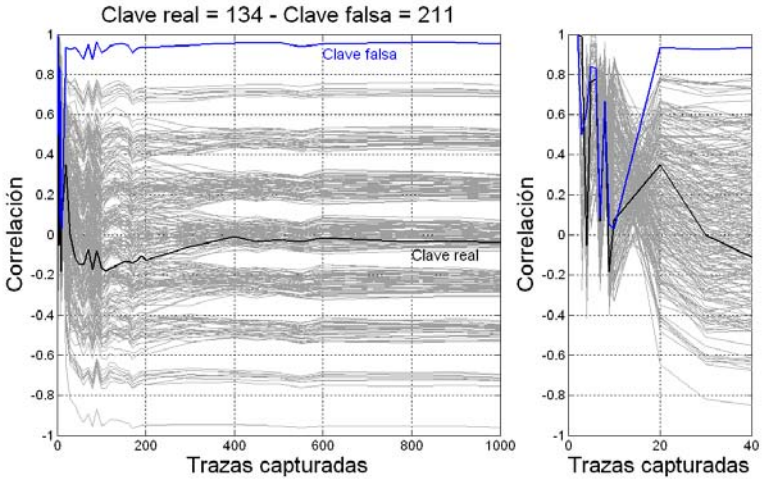

Figura 5.19:Análisis de correlación del sistema software protegido. Atacado el byte 0 tras la función *AddRoundKey* en la ronda 0. Representación en función de las trazas capturadas.

En la representación de la correlación en función de las trazas de la Figura 5.19 se comprueba como la correlación falsa se diferencia a partir de 20 trazas capturadas.

La diferencia de medias representada en la Figura 5.20 devuelve resultados similares al ataque de referencia. El valor máximo de la diferencia de medias se localiza en 1.55 mA.

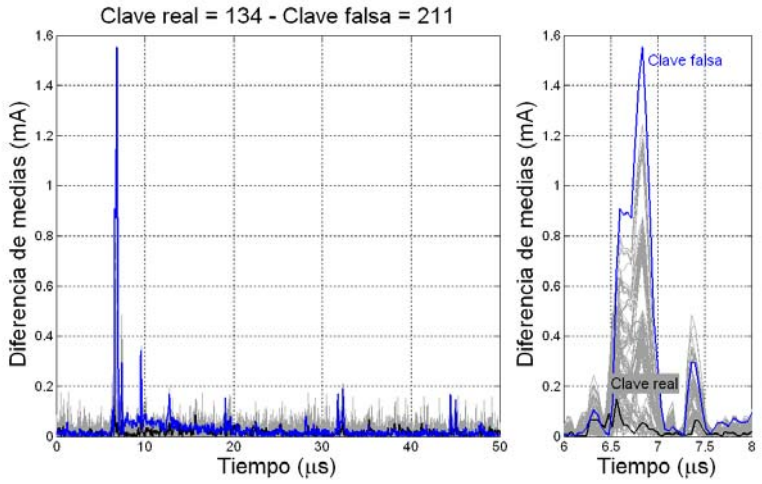

Figura 5.20:Análisis por diferencia de medias del sistema software protegido. Atacado el byte 0 tras la función *AddRoundKey* en la ronda 0.

5.3.2. Ataque a la salida de la función *SubBytes*

El resultado del siguiente análisis debe compararse con los que se muestran en el apartado 5.2.5.

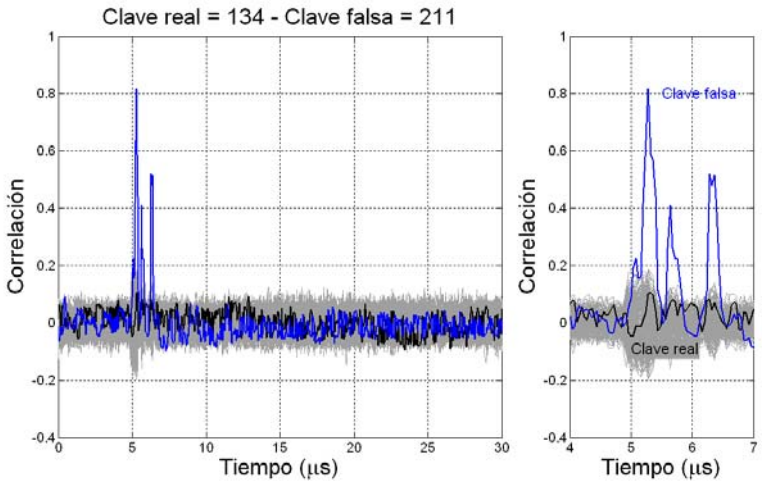

Figura 5.21:Análisis de correlación del sistema software protegido. Atacado el byte 0 tras la función *SubBytes* en la ronda 1.

Tal y como muestra la Figura 5.21, la correlación máxima se produce en la hipótesis de la clave máxima mientras que la clave real se oculta entre el resto de hipótesis. La correlación máxima se da con un valor de 0.81.

En la representación de la correlación en función de las capturas se observa como la correlación de la clave falsa se diferencia a partir de la captura de 30 trazas (Figura 5.22).

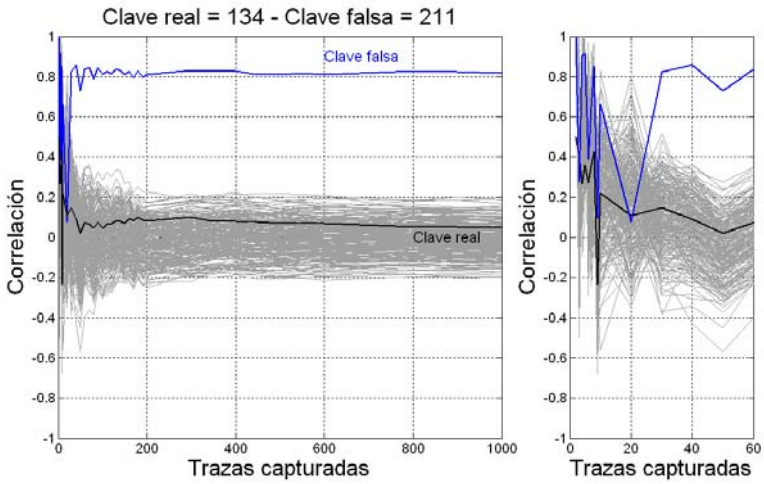

Figura 5.22:Análisis de correlación del sistema software protegido. Atacado el byte 0 tras la función *SubBytes* en la ronda 1. Representación en función de las trazas capturadas.

La máxima diferencia de medias devuelve una valor de 1.14 mA que diferencia la clave false de entre todas las demás. El resultado se muestra en la Figura 5.23.

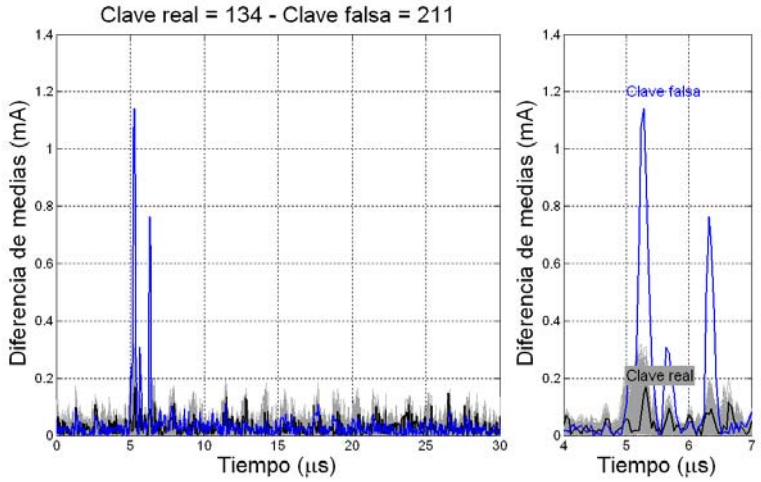

Figura 5.23:Análisis por diferencia de medias del sistema software protegido. Atacado el byte 0 tras la función *SubBytes* en la ronda 1.

5.3.3. Ataque a la salida de la función *MixColumns*

El resultado de los siguientes análisis debe compararse con los que se muestran en el apartado 5.2.6. El análisis de correlación, que aparece en la Figura 5.24, destaca la clave falsa sobre todas las demás con una correlación de 0.46. La clave real queda protegida.

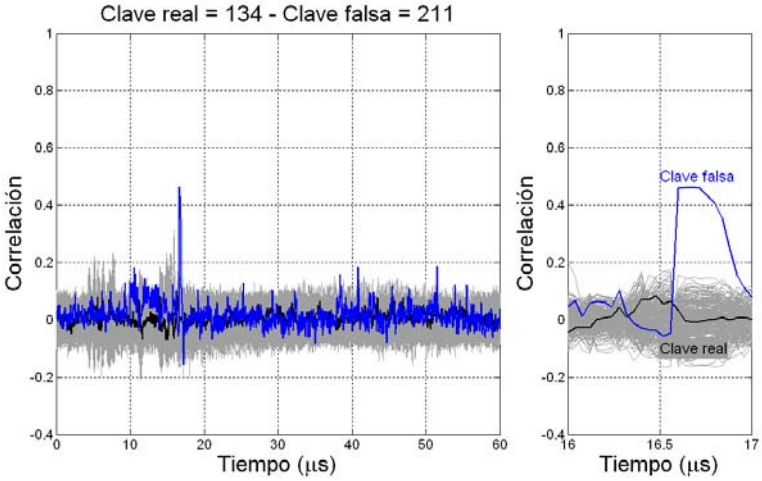

Figura 5.24:Análisis de correlación del sistema software protegido. Atacado el byte 0 tras la función *MixColumn* en la ronda 1.

La representación en función del número de trazas mostrada en la Figura 5.25, indica que la correlación es diferenciable a partir de 190 trazas.

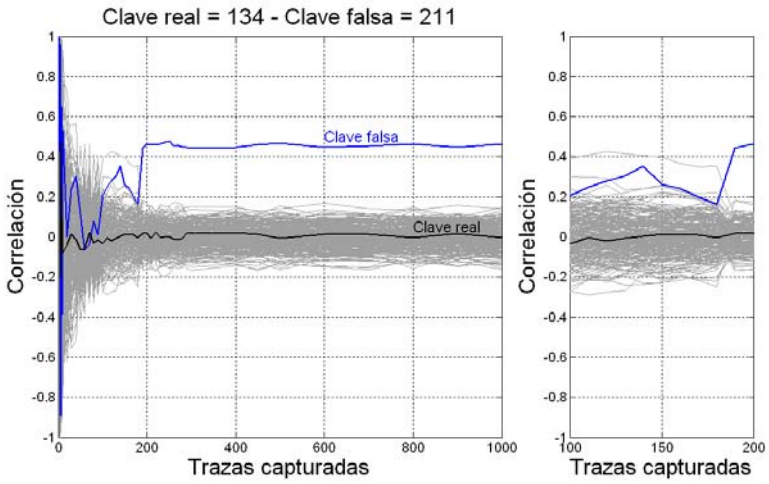

Figura 5.25:Análisis de correlación del sistema software protegido. Atacado el byte 0 tras la función *MixColumn* en la ronda 1. Representación en función de las trazas capturadas.

Finalmente la diferencia de medias representada en la Figura 5.26, muestra un valor de 2.36 mA para el valor de la clave correcta.

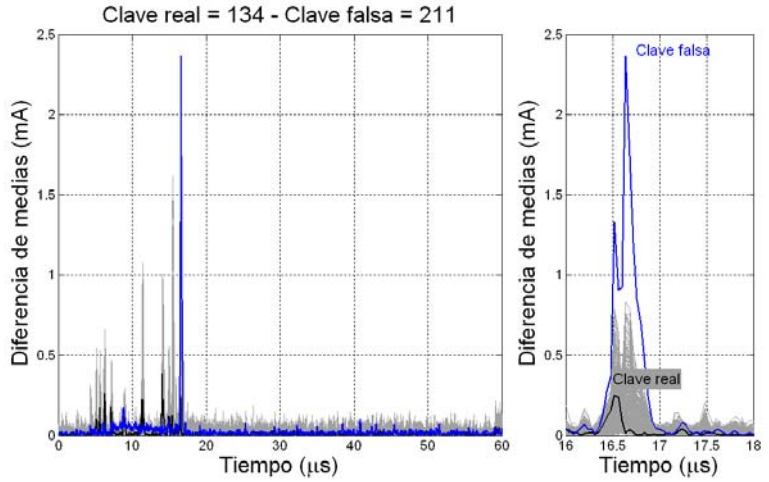

Figura 5.26:Análisis por diferencia de medias del sistema software protegido. Atacado el byte 0 tras la función *MixColumn* en la ronda 1.

5.3.4. Análisis de resultados

La implementación de la contramedida ha supuesto primeramente un incremento en la memoria necesaria para almacenar tanto el programa como las tablas $\text{SBOX}_{\text{TRANS}}$. En segundo lugar ha supuesto un aumento en el tiempo necesario para procesar cada texto plano. La Tabla 5.5 recoge los resultados comparados de la implementación software de referencia y la implementación software de la contramedida. Para una comparación más

real se han considerado los casos en los que el cálculo de $SBOX_{TRANS}$ pueden ser realizados de forma concurrente a la comunicación y los que no.

Parámetro	**Sist. referencia**	**Sist. Software**	**Variación**
Tiempo de ejecución	1.14 ms	1.59 ms	39 %
Tiempo de ejecución[49]	1.14 ms	2.49 ms	206 %
Capacidad de proceso	877 textos/s	629 textos/s	-28 %
Capacidad de proceso [49]	877 textos/s	401 textos/s	-54 %
Memoria necesaria	11770 bytes	19094 bytes	62 %

Tabla 5.5: Comparación de resultados versión software.

Tal y como puede apreciarse, el tiempo necesario para ejecutar el sistema protegido es un 39% superior al sistema de referencia. Concretamente en [Lum'2014] se establece que un enmascarado, como propone [Man'2007], representa un incremento del tiempo de proceso del 33%. En otra publicación [Her'2011] afirma que, en su caso, el enmascaramiento implica un aumento del tiempo de proceso del 74%.

Otra conclusión que se extrae de los resultados prácticos, es que la máxima penalización sobre el tiempo de procesado se produce debido al cálculo que debe hacerse de las tablas $SBOX_{TRANS}$. En sistemas en los que la velocidad de reloj es baja y/o no disponen de suficiente memoria para almacenar dichas tablas, es posible reducir la cantidad de elementos en la Key_{MASK}. En la ecuación (3.18) se observa que debe ser calculada una tabla $SBOX_{TRANS}$ por cada elemento diferente en Key_{MASK}. Si se usa una máscara para cada 4 elementos de la clave serán necesarios 4 elementos en la lista Key_{MASK} y, por tanto, los valores a calcular serán 4x256 bytes. La influencia que esta variación tiene sobre la seguridad del sistema es mínima, dado que el atacante desconoce que bytes de la clave están enmascarados por cada elemento de Key_{MASK}.

Este tipo de implementación es útil para aquellas aplicaciones en que la cantidad de datos a procesar no es elevada y cuentan con un sistema de comunicación vía serie.

[49] Añadiendo el tiempo necesario para el cálculo de las tablas SBOXTRANS

5.4. Resultados versión software/hardware

El uso de un dispositivo tipo FPGA permite, no sólo la configuración particular del microprocesador, sino la creación de coprocesadores específicos dedicados a realizar una función concreta. Estos coprocesadores hardware son capaces de resolver, en pocos ciclos de reloj, operaciones complejas para las que el microprocesador debe emplear un tiempo de ejecución superior.

En este caso el coprocesador se ha configurado para resolver el cálculo de la matriz de reenmascarado M_K (Apartado 3.3). La Figura 5.27 muestra la modificación del sistema base al que se le ha añadido el coprocesador que calcula la matriz mencionada.

El coprocesador se comunica con el procesador mediante el uso del bus FSL[50] [XIL'2011b]. Este bus permite al procesador una conexión rápida punto a punto con una pila FIFO[51] que facilita notablemente la comunicación entre dispositivos.

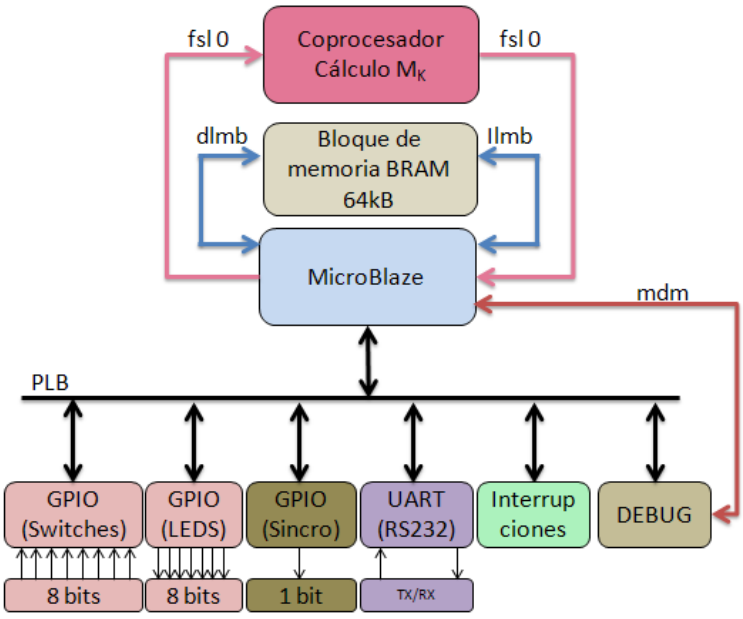

Figura 5.27:Diagrama de bloques del sistema software/hardware.

[50] De las siglas en inglés “Fast Simple Link”

[51] De las siglas en inglés “First In, First Out”

5.4.1. Implementación.

El coprocesador está encapsulado en una interfase que le permite la comunicación con el microprocesador a través del bus FSL (Figura 5.28). Esta interfase recibe 5 palabras de 32 bits del procesador, las envía hacia el coprocesador para que sean procesadas y devuelve 5 palabras de 32 bits hacia el FIFO del bus donde quedarán a la espera de ser leídas.

Las 5 palabras de 32 bits que recibe el coprocesador contienen información relevante para su funcionamiento: la primera palabra es un comando que indica al coprocesador la función que ha de realizar en cada momento y las otras 4 palabras contienen los datos que han de ser procesados. El coprocesador retorna 5 palabras hacia el procesador cuyo contenido depende del modo de funcionamiento del mismo.

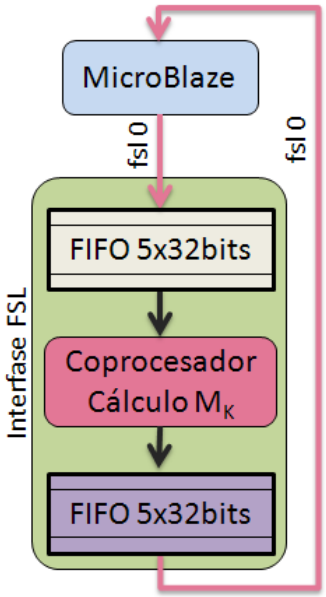

Figura 5.28:Diagrama de bloques de la inerfase FSL

La palabra que contiene el comando distingue entre los tres modos de funcionamiento siguientes:

a) Configuración de Key_{MASK}: los datos que acompañan a este modo constituyen los valores de Key_{MASK} del sistema y son necesarios para el cálculo de las tablas $SBOX_{TRANS}$. Este modo se usa para configurar el coprocesador en el momento del inicio y para modificar el

valor de Key_{MASK}. El coprocesador no retorna ningún dato hacia el procesador.

b) Procesado: con los datos recibidos se realiza la sustitución *SboxTrans* y posteriormente la función *MixColumn.* Este es el modo de funcionamiento más frecuente pues se usa en 9 de las rondas del algoritmo. El coprocesador retorna los valores que forman la matriz M_K y la matriz de reenmascarado de cada ronda.

c) Procesado reducido: En este caso únicamente se procesa la función *SboxTrans* y está destinado a calcular la matriz de reenmascarado en la última ronda del algoritmo. Los datos retornados son los valores que forman la matriz M_J y la matriz de reenmascarado de la última ronda.

El funcionamiento del coprocesador se secuencia mediante una máquina de estados. Una vez que los 16 bytes del estado han sido recibidos se inicia el procesado en bloques de 1 byte. Hay que tener en cuenta que el microprocesador se encuentra ejecutando del algoritmo AES de forma paralela y el consumo del coprocesador no debe afectar a la correlación producida para la clave falsa.

Las tablas $SBOXT_{RANS}$ se almacenan en una memoria BRAM[52][XIL'2010c] configurada para un tamaño del bus de datos de 8 bits, una capacidad de 4kB y un doble puerto de lectura y escritura. Los datos se distribuyen de forma que cada página de datos de 256kB contiene los datos correspondientes a la $SBOX_{TRANS}$ del un byte del estado.

La reprogramación de las tablas debe realizarse en cada ciclo de encriptado. Este proceso implica escribir 4096 bytes y se requieren otros tantos ciclos de reloj para completarlo. La forma de evitar que la reprogramación afecte a la velocidad de procesamiento del algoritmo es trabajar con dos memorias de 4kB de forma concurrente. Mientras se realiza un encriptado, se fijan los valores de la matriz M_J, se calcula la matriz M_K y se reprograma la memoria BRAM que se usarán en el siguiente ciclo de encrip-

[52] "Block RAM (BRAM)" implementadas en las FPGA de la familia VIRTEX 5 de XILINX.

tado. Alternativamente se cambian las funciones de las memorias BRAM para cada texto plano.

El algoritmo de reprogramación es el definido en las ecuaciones (3.27) y (3.28), del cual puede verse la implementación en la Figura 5.29.

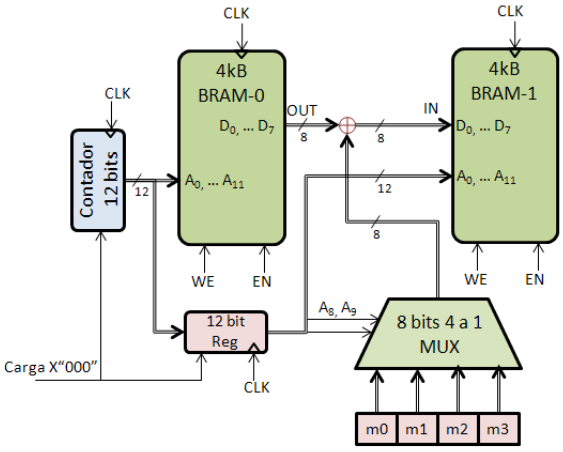

Figura 5.29: Esquema de reprogramación de la memoria BRAM.

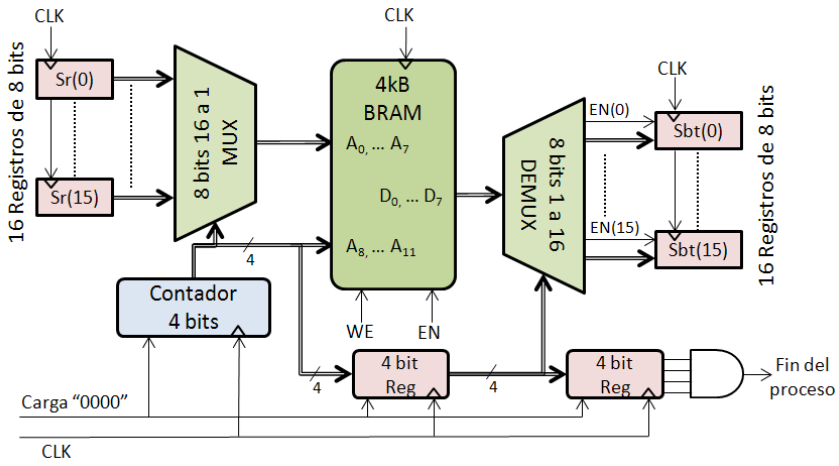

Figura 5.30: Implementación hardware de la función *SboxTrans*[53].

La función *SboxTrans* procesa secuencialmente los 16 bytes que forman el estado. En la Figura 5.30 puede verse el esquema de la implementación hardware de esta función. El procesado requiere de 18 ciclos de reloj, 16 para procesar los 16 datos recibidos y 2 ciclo debidos al retardo de respuesta de la memoria BRAM. Una vez concluido el proceso se informa a la

[53] Todos los buses son de 8 bits a excepción de aquellos en que se indica lo contrario.

máquina de estados para que se continúe con la secuencia de funcionamiento establecida.

Para la implementación de la función *MixColumn,* del algoritmo de encriptado AES, es necesario realizar las operaciones "x2" y "x3" del campo finito de Galois (Apartado 3.2.1). Estas funciones pueden implementarse utilizando la función *xtime()* [NIST'2001] [Dae'1999], y cuya realización puede hacerse mediante un circuito puramente combinacional. En la Figura 5.31 se muestra la implementación de la operación "x2". El byte de entrada es sometido a un desplazamiento a la izquierda de sus bits introduciendo un '0' por la derecha. Si el bit más significativo del dato de entrada es de valor '1' al resultado del desplazamiento debe aplicársele la operación lógica or-exclusiva con el valor constante X"1B".

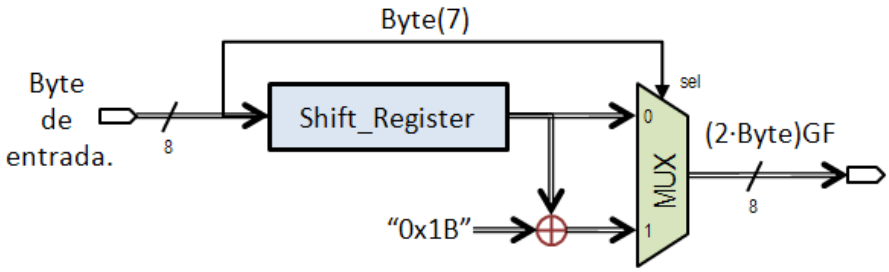

Figura 5.31:Implementación de la operacin x2 en el campo GF^2 de Galois.

La operación "x3" se resuelve mediante la operación or-exclusiva del resultado de la operación "x2" con el valor de entrada. En la Figura 5.32 puede verse la implementación de esta operación.

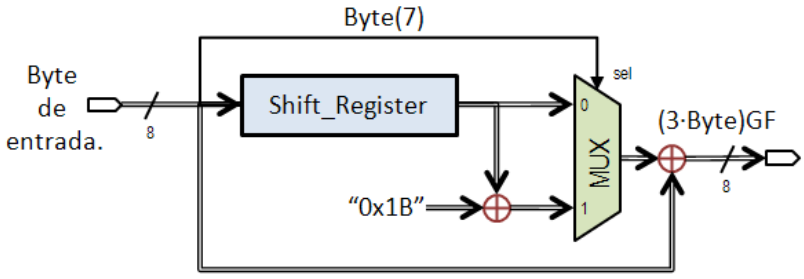

Figura 5.32:Implementación de la operacin x3 en el campo GF^2 de Galois.

Utilizando estos dos módulos combinacionales puede implementarse la función *MixColumn* como se muestra en la Figura 5.33:

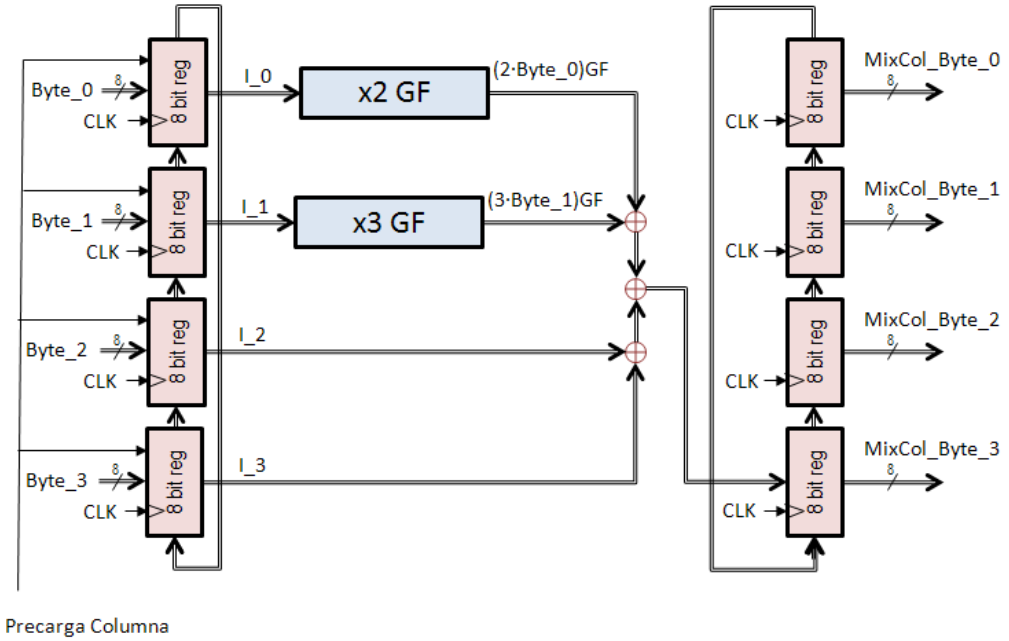

Figura 5.33:Implementación de la función *MixColumn*.

El procesado se realiza por columnas. Para ello los 4 registros de entrada se precargan con los valores de la columna. Seguidamente, a cada ciclo de reloj, los valores de entrada se desplazan de un registro a otro para ir efectuando la operación correspondiente a la cada uno de los elementos de la columna de salida tal y como que se muestra en la ecuación (3.3). Serán necesarios 5 ciclos de reloj para completar el proceso, uno para la precarga y 4 para el procesado de los 4 bytes de la columna de salida. El mismo tratamiento se realiza para cada una de las columnas del estado.

La Tabla 5.6 muestra los recursos de la FPGA que el coprocesador necesita. Estos datos pueden ser comparados con los de la implementación del procesador Microblaze expuestos en la Tabla 5.1. En ella se muestran por separado que parte de estos recursos son necesarios para el coprocesador, propiamente dicho, y que recursos necesita el driver que facilita la comunicación con el micro a través del bus FSL.

Parámetro	**Copro.**	**Driver FSL**	**Total**
Frec. de reloj máxima		196 MHz	
"Slices" consumidos		489 de 7200 6.8%	
"LUTS" consumidos	653	303	956 de 28800 3.3%
"FF" consumidos	688	601	1289 de 28800 4.5%
36K BRAMS	2	--	2 de 48 4.2%

Tabla 5.6: Recursos de la FPGA consumidos por el coprocesador

En la Tabla 5.7 se recogen el tiempo necesario para procesar un texto plano y la memoria ocupada por el programa.

Parámetro	Valor
Tiempo de ejecución AES-FAKE	1.30 ms
Memoria ocupada	14034 bytes

Tabla 5.7: Tiempos de ejecución de la versión software hardware.

5.4.2. Consumo del coprocesador.

El consumo eléctrico de la lógica que constituye el coprocesador no debe ser identificable por un atacante para que no pueda ser aislado. Por otro lado, tampoco debe modificar el consumo del procesador en los puntos en los que se pierde información relativa a la clave falsa, para que no la altere la correlación que pretende ser forzada.

El coprocesador es capaz de resolver el cálculo de la matriz de reenmascarado en 50 ciclos de reloj considerando:

a) 1 CLK de carga de datos

b) 18 CLK función SBoxTrans

c) 1 CLK Cambio de estado en la máquina de estados

d) 1x4 CLK carga de datos de una columna

e) 4x4 CLK de procesado de una columna

f) 1x3 CLK Cambio de columna

g) 1 CLK Cambio de estado en la máquina de estados

h) 1 CLK Salida de datos

i) 5 CLK Escritura en el FIFO del bus.

La ejecución de todas las operaciones del coprocesador de forma continuada afecta de forma notable al consumo del dispositivo. La Figura 5.34 muestra una comparativa del consumo del dispositivo en dos casos: en primer lugar se captura el consumo cuando el coprocesador está funcionando, seguidamente se captura el consumo cuando el procesador está parado y, finalmente, se calcula la diferencia entre ambos. El resultado de la

resta mostrará el consumo del procesador aislado del consumo del resto del sistema.

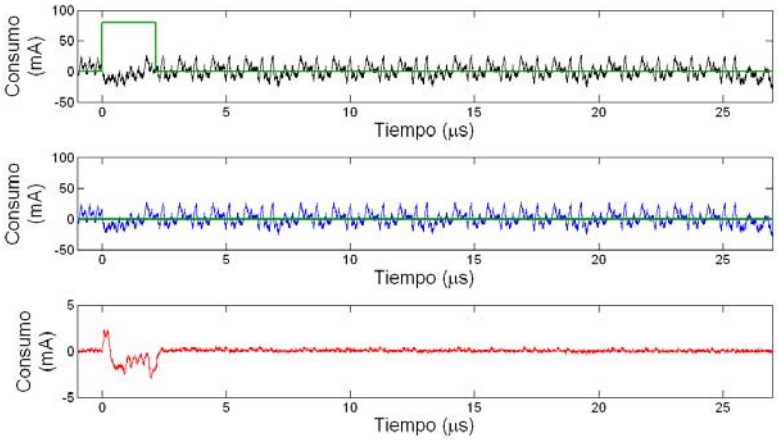

Figura 5.34: Consumo concentrado del coprocesador; a) Coprocesador activo b) Coprocesador inactivo c) diferencia entra ambos.

Como puede apreciarse, la diferencia entre uno y otro alcanza el valor absoluto de 2.8 mA. Además, en un análisis simple de la traza, pueden ser identificadas las diferentes partes en las que se descompone funcionamiento del coprocesador. En la Figura 5.35 se representa el tramo de consumo relacionado con la ejecución de la función *SboxTrans*, el procesado de cada una de las 4 columnas mediante la función *MixColumn* y la escritura de los datos en el FIFO del bus FSL.

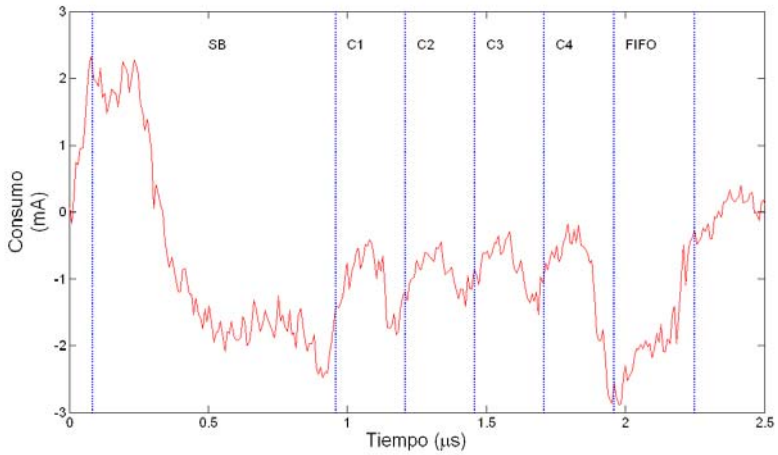

Figura 5.35: Análisis simple del consumo del coprocesador.

.

En cambio, si el consumo se distribuye en el tiempo, la diferencia es inapreciable. En la Figura 5.36 se muestra la misma comparación que se ha realizado para construir la Figura 5.34. El coprocesador realiza su función de forma concurrente a la ejecución de la función *SubBytes* en el procesador que ejecuta el algoritmo AES.

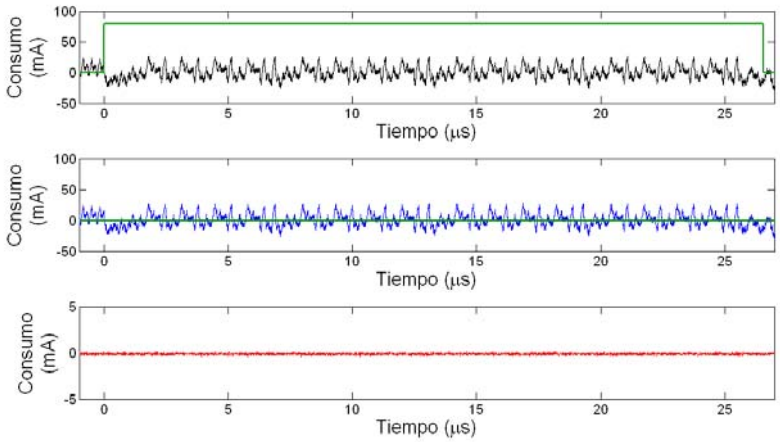

Figura 5.36: Consumo distribuido del coprocesador. a) Coprocesador activo b) Coprocesador inactivo c) diferencia entra ambos.

En este caso, la diferencia no supera los 0.5 mA absolutos. Por consiguiente, el consumo del coprocesador no es detectable ni aislable del consumo del resto del sistema

En segundo lugar, para que el consumo del coprocesador no altere la correlación que se pretende forzar sobre la clave falsa, su funcionamiento no debe ser concurrente con aquellos instantes en los que se filtra información relevante en el consumo. En la Figura 3.6 se muestran los instantes en que la pérdida de información se produce, estos coinciden con la lectura de los valores de la tabla SBOX y su posterior escritura en el estado.

Considerando que, tanto la tabla SBOX como el estado están almacenados en la memoria del dispositivo, el coprocesador deberá interrumpir su funcionamiento siempre que el bus de direcciones del bus de datos del microprocesador apunte a una de las dos zonas de memoria.

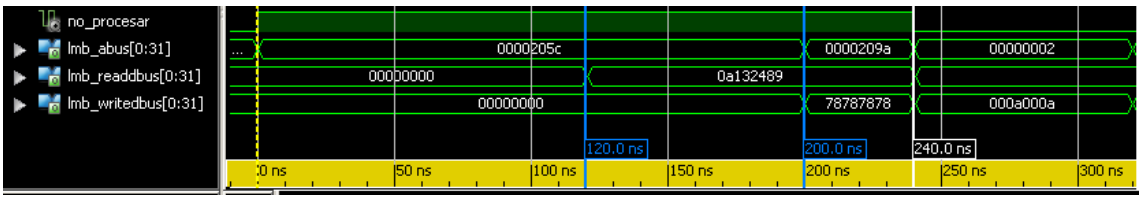

Figura 5.37: Simulación de la protección del acceso a memoria para la lectura del estado.

En la Figura 5.37 y en la Figura 5.38 se simula el comportamiento de la señal “no_procesar” que es la que indica al procesador que detenga su funcionamiento cuando se accede a la lectura del estado. En la simulación se procesa la primera fila del estado con valores X”0A132489”.

La simulación presenta la siguiente secuencia:

a) **Ts=0ns,** se escribe en el bus de direcciones la posición de memoria en la que se encuentra el primer byte del estado.

b) **Ts=120 ns,** los valores de la primera fila se escribe en el bus de datos. En este punto se produce perdida de información que queda protegida por la señal “no_procesar”

c) **Ts=200 ns,** en el bus de direcciones se escribe la posición de memoria en la que se encuentra el valor SBOX correspondiente al primer byte del estado.

d) **Ts=240 ns,** se habilita de nuevo el funcionamiento del coprocesador.

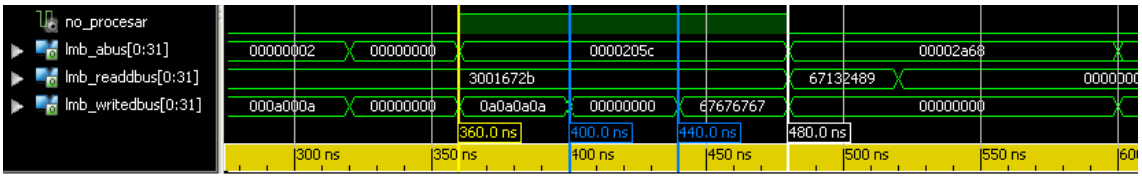

Figura 5.38: Simulación de la protección del acceso a memoria para la escritura del estado.

e) **Ts=360ns**, Se direcciona de nuevo la posición de memoria donde se encuentra el estado almacenado, esto implica el bloqueo del coprocesador. Al mismo tiempo, en el bus de escritura de datos se escribe el valor del byte procesado por cuadruplicado.

f) **Ts=400ns**, el bus de escritura de datos se carga al valor X"00000000", en este punto se produce perdida de información modelable mediante HW.

g) **Ts=440ns,** ahora en el bus de escritura se carga el valor correspondiente a la tabla SBOX, por tanto, se produce perdida modelable mediante HW.

h) **Ts=480ns,** se desbloquea el funcionamiento del procesador hasta que se opere con el siguiente byte del estado.

5.4.3. Ataque a la salida de la función *AddRoundKey*.

Los escenarios planteados para los ataques son los mismos que en los dos casos anteriores. A continuación se muestran los resultados de los análisis efectuados al consumo durante la ejecución de la función *AddRoundKey* en la ronda 0 del algoritmo.

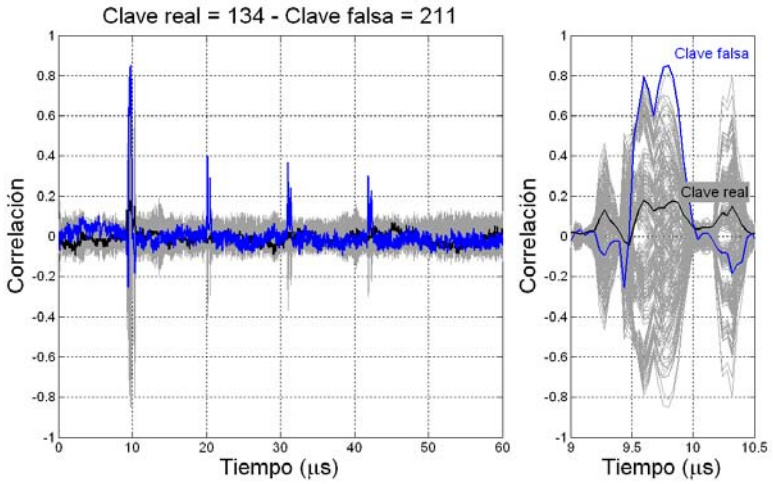

Figura 5.39:Análisis de correlación del sistema software-hardware protegido. Atacado el byte 0 tras la función *AddRoundKey* en la ronda 0.

En la Figura 5.39 se aprecia que la correlación máxima alcanza un valor de 0.85 y en la Figura 5.40 se muestra que la diferenciación de la traza se produce con 125 capturas. En ambas figuras se puede ver como la hipótesis sobre la clave real queda totalmente protegida.

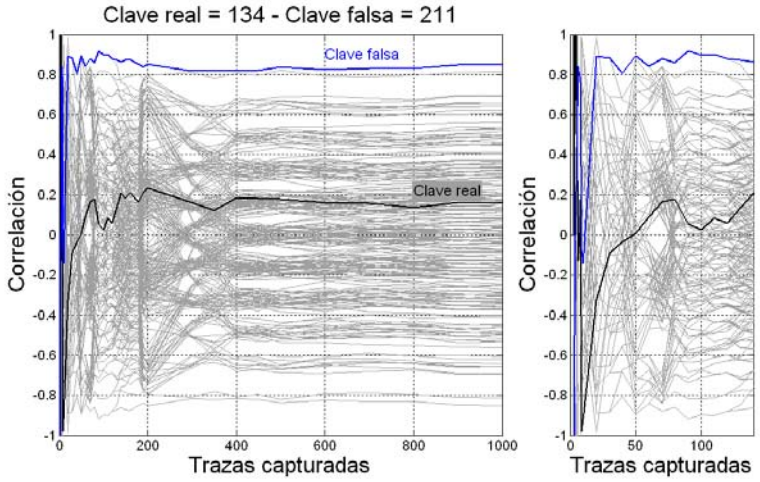

Figura 5.40:Análisis de correlación del sistema software-hardware protegido. Atacado el byte 0 tras la función *AddRoundKey* en la ronda 0. Representación en función de las trazas capturadas.

El resultado del ataque por diferencia de medias que se expone en la Figura 5.41 verifica que el sistema también está protegido. La diferencia máxima observada en este caso es de 1.17 mA.

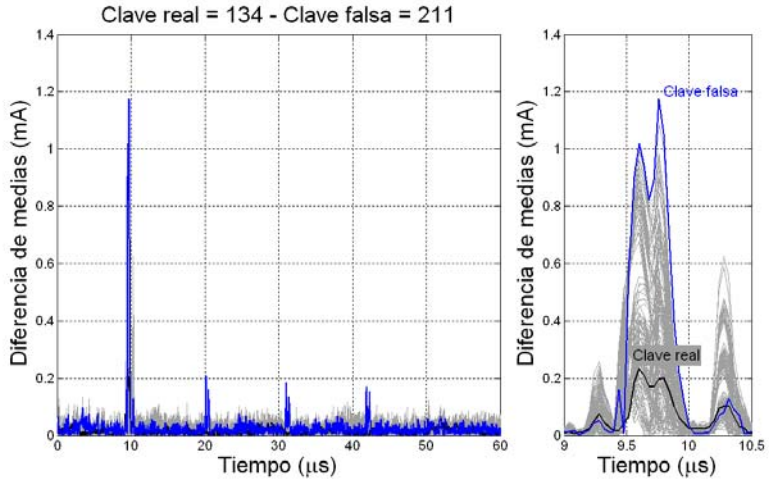

Figura 5.41:Análisis por diferencia de medias del sistema software-hardware protegido. Atacado el byte 0 tras la función *AddRoundKey* en la ronda 0.

5.4.4. Ataque a la salida de la función *SubBytes*.

Los resultados del ataque a la salida de la función *SubBytes* en la ronda 1 arrojan resultados similares, la correlación máxima alcanza 0.89 en la hipótesis de la clave falsa. Nuevamente la hipótesis real queda protegida. La Figura 5.42 muestra el análisis de correlación.

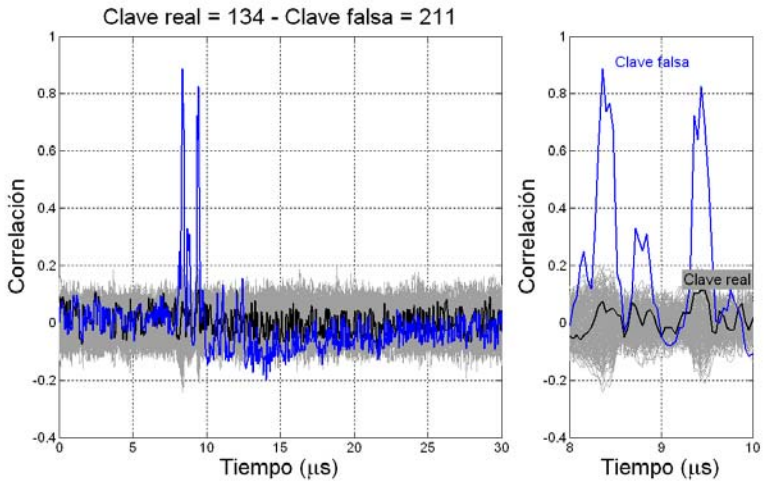

Figura 5.42:Análisis de correlación del sistema software-hardware protegido. Atacado el byte 0 tras la función *SubBytes* en la ronda 1.

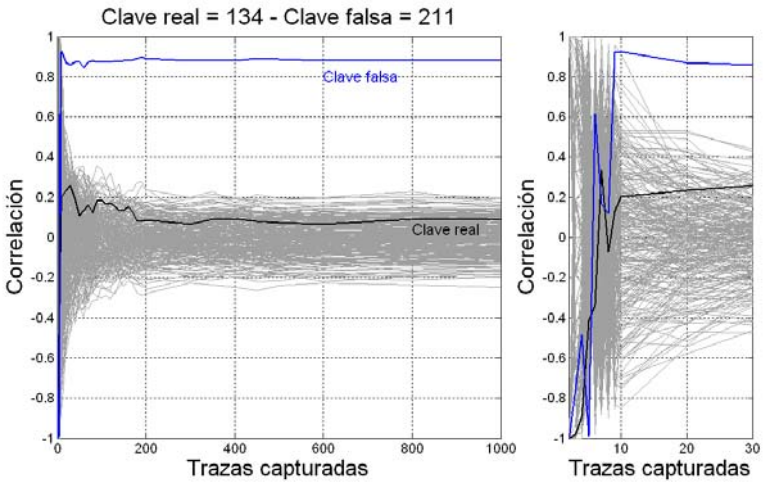

Figura 5.43:Análisis de correlación del sistema software-hardware protegido. Atacado el byte 0 tras la función *SubBytes* en la ronda 1. Representación en función de las trazas capturadas.

En la Figura 5.43 puede verse que la diferenciación de la hipótesis correcta se da con 20 trazas capturadas.

Finalmente, el análisis de diferencia, que se muestra en la Figura 5.44, de medias muestra una diferencia absoluta de 1.66 mA que diferencia claramente la hipótesis falsa sobre la real.

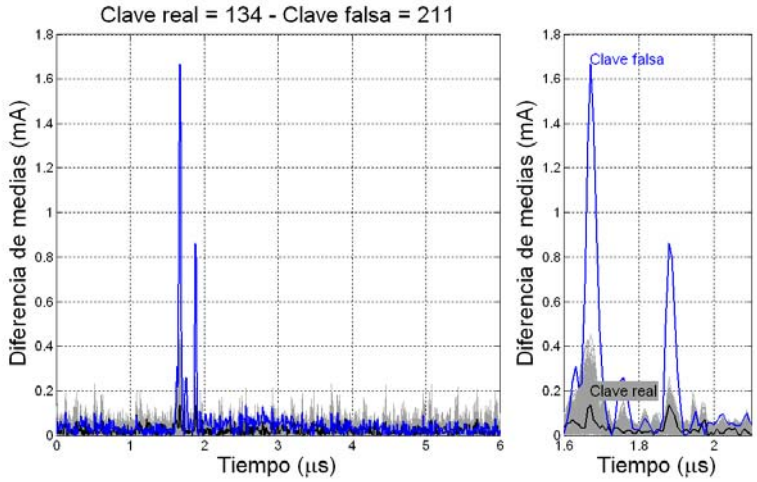

Figura 5.44:Análisis por diferencia de medias del sistema software-hardware protegido. Atacado el byte 0 tras la función *SubBytes* en la ronda 1.

5.4.5. Ataque a la salida de la función *MixColumn*.

El último punto de ataque demuestra que el sistema se comporta del mismo modo a como lo hace el sistema de control sin contramedidas. La Figura 5.45 presenta el resultado del análisis a la salida de la función *MixColumn* en el que se da una correlación de 0.47. Paralelamente, en la Figura 5.46 se aprecia que la correlación para la clave falsa se diferencia a partir de 50 capturas.

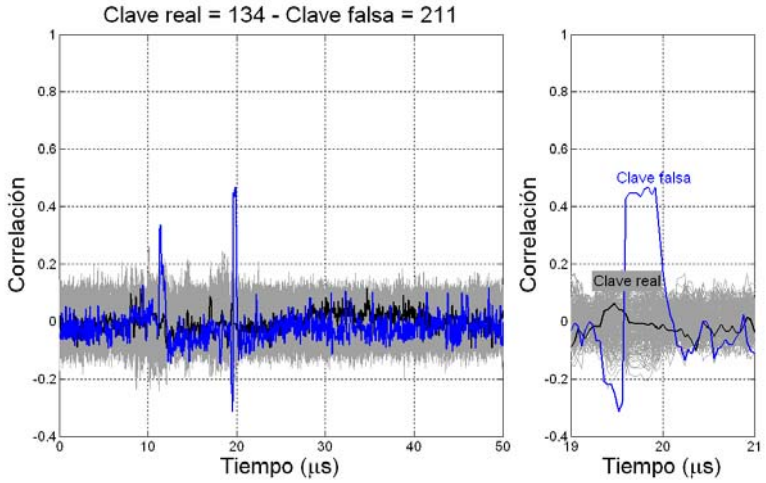

Figura 5.45:Análisis de correlación del sistema software-hardware protegido. Atacado el byte 0 tras la función *MixColumn* en la ronda 1.

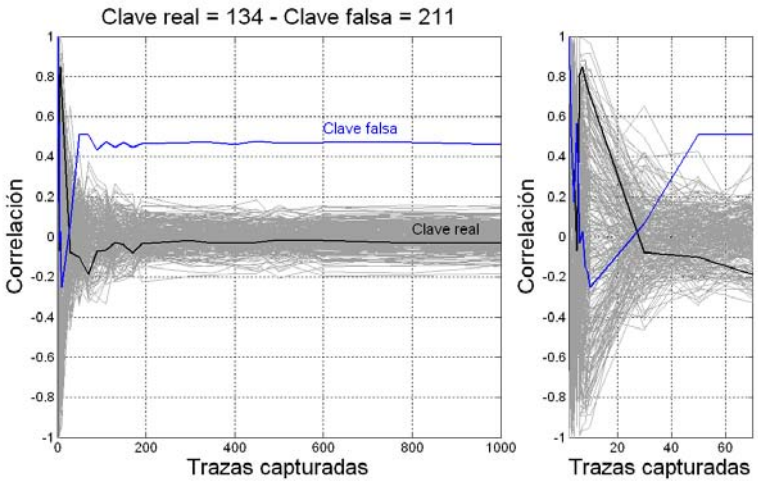

Figura 5.46:Análisis de correlación del sistema software-hardware protegido. Atacado el byte 0 tras la función *MixColumn* en la ronda 1. Representación en función de las trazas capturadas.

Concluyendo la comparativa, el análisis por diferencia de medias da un falso positivo de forma contundente al detectar una diferencia de 2.5 mA en la hipótesis falsa quedando la hipótesis real oculta entre todas las demás hipótesis analizadas. El resultado aparece en la Figura 5.47.

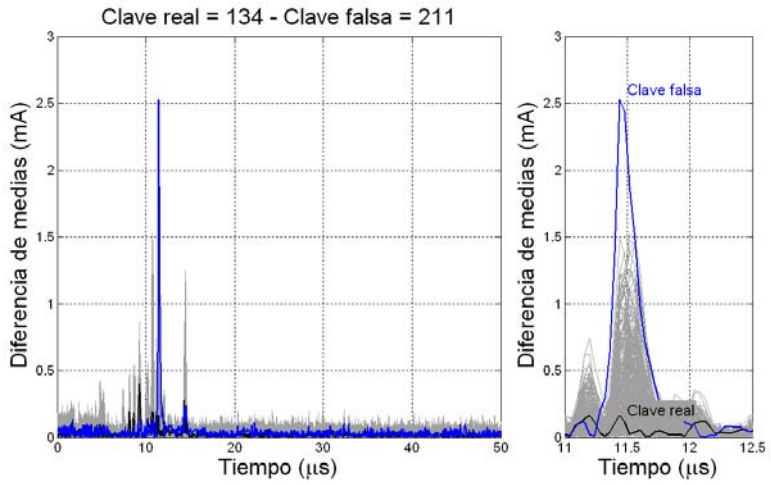

Figura 5.47:Análisis por diferencia de medias del sistema software-hardware protegido. Atacado el byte 0 tras la función *MixColumn* en la ronda 1.

5.4.6. Análisis de resultados

Dado que el coprocesador debe comunicarse con el resto del sistema utilizando un bus, será necesario implementar un driver que facilite esta tarea. A pesar de que, estrictamente el driver no forma parte del coprocesador, deben considerarse los recursos necesarios para su funcionamiento.

La Tabla 5.8 recoge la comparativa de los recursos consumidos por el sistema software de referencia y el sistema con la contramedida implementada.

Parámetro	**Sistema de referencia.**	**Sistema Hard./Soft.**	**Variación**
Frec. de reloj máxima	161 MHz	161 MHz	0%
"Slices" consumidos	1256 (17.4%)	1745 (6,8%)	38.9 %
"LUTS" consumidos	2264 (7.9%)	3220 (11.2%)	42%
"FF" consumidos	2006 (7.0%)	3295 (11.4%)	64%
36K BRAMS	16 (33.3%)	18 (37.5%)	18.7%

Tabla 5.8:Comparación de recursos consumidos por el sistema de referencia y el sistema hardware/software. Entre parentesis el porcentaje respecto al total de la FPGA.

La contramedida implica un aumento de recursos de aproximadamente el 50% respecto del sistema de referencia, aunque el sistema protegido no llega a consumir el 12% de los recursos de la FPGA.

Comparando estos resultados con otras contramedidas de implementación hardware se observa que, en sistemas que incorporan estructuras RSL el área necesaria para implementar la contramedida se multiplica por 2 y el tiempo necesario para realizar el proceso aumenta a razón de 1.5 a 1 [Dai'2004]. Del mismo modo en estructuras WDDL el área necesaria se multiplica por 3 [Tir'2005] o por 2.3 usando la propuesta de implementación parcial DDL de [Kap'2010].

En lo que respecta a la parte software del AES, ésta debe incluir las funciones de comunicación con el coprocesador a través del bus FSL. Este hecho provoca un incremento en el tiempo de ejecución respecto de la versión de referencia. En la Tabla 5.9 puede verse la comparativa entre estos dos sistemas.

Parámetro	**Sist. referencia**	**Sist. Sof./Hard.**	**Variación**
Tiempo de ejecución	1.14 ms	1.30 ms	14 %
Capacidad de proceso	877 textos/s	769 textos/s	-12 %
Memoria necesaria	11770 bytes	14034 bytes	19 %

Tabla 5.9:Comparación de resultados versión software.

La implementación hardware/software de la contramedida representa una mejora sustancial, en tiempo de ejecución, respecto a la implementa-

ción software. Mientras que ésta última es capaz de procesar 629 textos/s la versión hardware/software puede procesar 769 textos/s lo que representa un 22.4% más. Si la comparativa se realiza con los resultados obtenidos cuando deben calcularse las tablas $SBOX_{TRANS}$ antes de cada ciclo de encriptado, la mejora asciende al 91%.

Esta implementación resulta útil en un sistema en el que la codificación de datos sea una más de las funciones del microprocesador. Por ejemplo, un sistema de identificación biométrica, el procesador debería codificar los parámetros de identificación que están almacenados en un soporte externo para proteger su integridad. En cambio también debe ejecutar los algoritmos que le permitan resolver positiva o negativamente una identificación. Por supuesto que este coprocesador podría formar parte de un sistema reconfigurable dinámicamente y ser configurado en el momento en que deba utilizarse [Des'2012].

5.5. Resultados versión Hardware

El uso de dispositivos FPGA permite implementar el algoritmo AES de múltiples formas. El diseño puede orientarse hacia la optimización de diversos parámetros como por ejemplo: el tiempo necesario para procesar cada texto plano, la cantidad de recursos que se consumen del dispositivo o la velocidad máxima a la que deba trabajar el sistema.[Goo'2005].

En ese orden de ideas, es posible implementar un sistema que procese datos en bloques de 8 bits, que consuma escasos recursos del dispositivo pero que necesite un gran número de ciclos de reloj para completar el proceso [Jen'2007]. También puede optarse por una implementación de 32 bits [Ben'2012], o una de 128 bits [Hua'2007], acudiendo, o no, al uso de estructuras *pipeline* [Qia'2013], en función de la aplicación principal y/o de las necesidades del sistema.

Por otro lado, considerando que las operaciones básicas del algoritmo [NIST'2001] pueden implementarse de forma puramente combinacional, es posible encadenarlas para resolver varias de ellas en un único ciclo de reloj.

5.5.1. Implementación del sistema de referencia.

Para la parte práctica se ha optado por utilizar una implementación en la que se procesan 128 bits (la totalidad del estado) de forma concurrente. La comunicación con el ordenador se realiza mediante una unidad UART que envía y recibe los datos a través de un puerto RS232 a una velocidad de 115200 bps. La UART presenta los datos en tamaño de un byte a medida que los recibe. Una interfase los almacena y los concatena en un vector de 128 bits para enviarlos al módulo AES junto a una señal de inicio. Tras la conclusión del proceso, la unidad AES devuelve 128 bits a una segunda interfase que los presenta en tamaño de 1 byte al módulo UART para que lo envíe de vuelta al ordenador. El diagrama corresponde al que se muestra en la Figura 5.48.

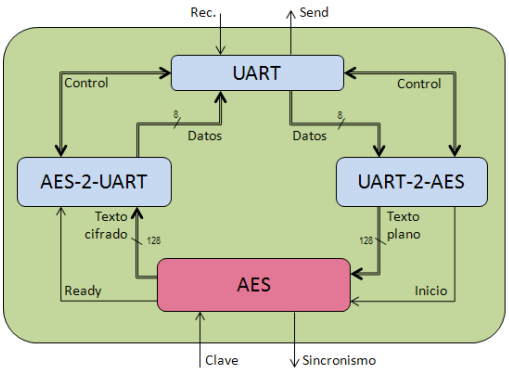

Figura 5.48: Implementación general del sistema hardware.

El módulo AES dispone de una salida digital destinada al sincronismo de la captura de datos por el osciloscopio. También dispone de una entrada digital que se activa externamente para indicar que los próximos 128 bits que se reciban forman la clave de trabajo del sistema.

La estructura interna del bloque AES incluye un registro y un bloque combinacional (Figura 5.49). El registro almacena el resultado de la función *SubBytes* de una ronda y lo presenta para que sea procesado en la siguiente. El bloque combinacional incorpora las operaciones *AddRoundKey*, *ShiftRows*, *SubBytes* y *MixColumn* de forma que resuelve una ronda del algoritmo en cada ciclo de reloj.

La complejidad de estas operaciones es reducida, por tanto, su concatenación no implica una sucesión grande de circuitos combinacionales co-

nectados en cascada ni una disminución apreciable de la frecuencia máxima de trabajo del sistema. Primeramente, la función *AddRoundKey* consiste en la función or-exclusiva entre el estado y la clave de ronda. En segundo lugar, la función *ShifRows* se trata en un reordenamiento de las señales que conduce cada uno de los valores del estado. Seguidamente, la función *SubBytes* es una búsqueda en una memoria ROM de 256 bytes. Finalmente la función *MixColumn* se resuelve implementando para cada byte del estado las funciones "x2" y "x3" del campo finito de Galois, tal y como se muestra en la Figura 5.31 y en la Figura 5.32 respectivamente, para resolver la ecuación añadiendo las funciones or exclusiva necesarias.

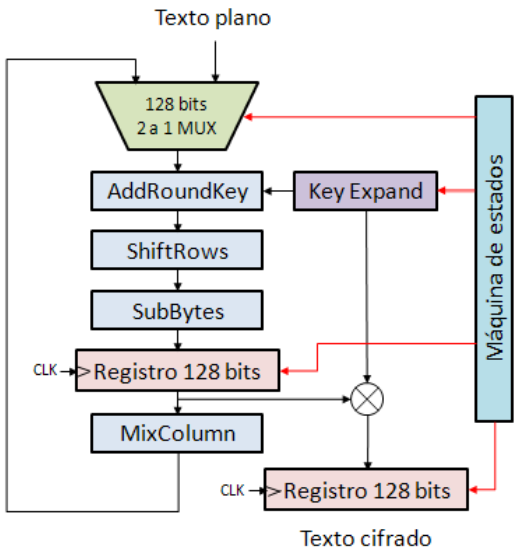

Figura 5.49: Estructura del sistema AES en implementación Hardware.

La expansión de la clave puede realizarse en paralelo a la ejecución del algoritmo, de forma que cada clave de ronda se calcula a partir de la clave de la ronda previa. Por el contrario, la clave expandida puede ser precalculada una única vez y almacenada en una memoria RAM a la espera de ser utilizada [Sri'2012]. En esta implementación se ha optado por el segundo de los métodos. El objetivo es evitar que las operaciones necesarias para realizar la expansión de la clave de forma concurrente al algoritmo añadan ruido de conmutación a la medida de corriente y alteren la correlación que se persigue (Apartado 2.7.2).

Todo el sistema está secuenciado mediante una maquina de estados que se encarga de controlar la ronda que se está ejecutando en cada momento para proporcionar la clave correspondiente, registrar el texto cifrado

de salida y activar la señal de fin de encriptado para que éste sea enviado al ordenador.

Puesto que se pretende obtener una correlación alta en una clave falsa, el sistema debe ser diseñado considerando esa condición. El punto débil del sistema está en el registro en el que se almacena el resultado de cada ronda. Para conseguir que el sistema sea sensible a un ataque mediante el modelo HW es necesario precargar el registros al valor lógico "0" antes de la ejecución de la primera ronda del algoritmo.

La Tabla 5.10 muestra los datos de implementación del sistema de referencia.

Parámetro	**Valor**
Frecuencia de reloj máxima	192 MHz
"Slices" consumidos	1193 de 7200 15%
"LUTS" consumidos	2684de 28800 9%
"FF" consumidos	2610 de 28800 9 %

Tabla 5.10: Recursos de la FPGA consumidos por el sistema Hardware de referencia.

Para la obtención de los resultados experimentales se han utilizado los parámetros de captura que se muestran en laTabla 5.11.

Parámetro	**Valor**
Frecuencia del sistema	24 MHz
Frecuencia de muestreo	2 Gs/s (83,33 muestra/clk)
integración	≈1 ciclo de reloj (83 muestras)
Tiempo de captura	208 s /1000 trazas
Tiempo de integración	0.84 s /1000 trazas
Tiempo de cálculo de la correlación	0.22 s/1000 trazas integradas · byte
Tiempo de cálculo, diferencia de medias	39 s/1000 trazas integradas · byte

Tabla 5.11:Parámetros de captura para el sistema Hardware.

5.5.2. Ataque al registro post *SubBytes*.

El ataque se ha planteado al registro de la primera ejecución del *SubBytes*. En los sistemas hardware, la correlación esperada se ve radicalmente afectada por dos motivos. En primer lugar se están procesando 128 bits mientras que el modelo se aplica únicamente a 8 bits; y en segundo lugar,

tras el registro aparece un bloque combinacional formado por puertas en cascada, que modificará el estado de sus señales en el instante en que se registren los valores. Estas dos circunstancias afectarán a la fidelidad del modelo, por tanto, la correlación será mucho menor y serán necesarias un mayor número de trazas para diferenciar la hipótesis correcta de las demás.

En la Figura 5.50 puede verse el resultado del análisis de correlación para el que se han capturado 10000 trazas. La hipótesis correcta arroja una correlación de 0.049, lo que representa un 44% por encima del resto de valores. En consecuencia, este resultado es más que suficiente para diferenciar una hipótesis de las otras.

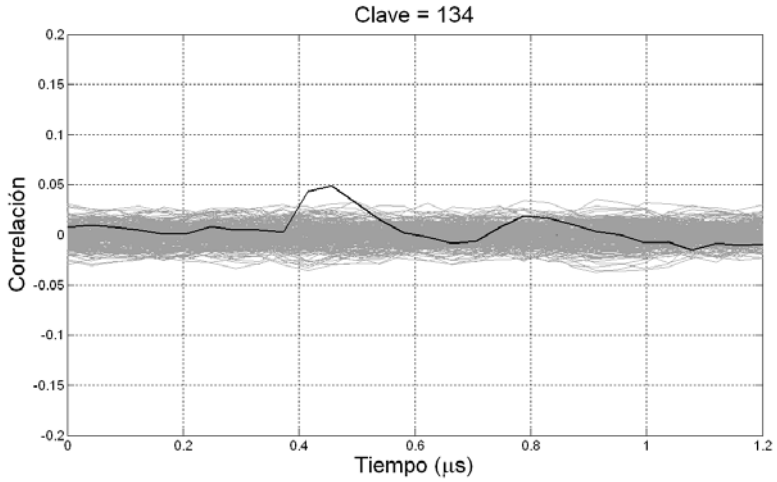

Figura 5.50:Análisis de correlación del sistema hardware no protegido. Atacado el byte 0 en el registro post *SubBytes* en la ronda 0.

La Figura 5.51 muestra como la hipótesis correcta se diferencia de las demás a partir de la toma de 4500 trazas. Su valor se mantiene al incrementar las trazas, pero la correlación de las demás va reduciéndose haciendo la diferencia cada vez más notable.

El análisis por diferencia de medias se muestra útil para diferenciar la clave correcta de la que no lo es, aunque la diferencia entre la máxima diferencia de medias obtenida y el resto sea únicamente del 10%. (Véase Figura 5.52)

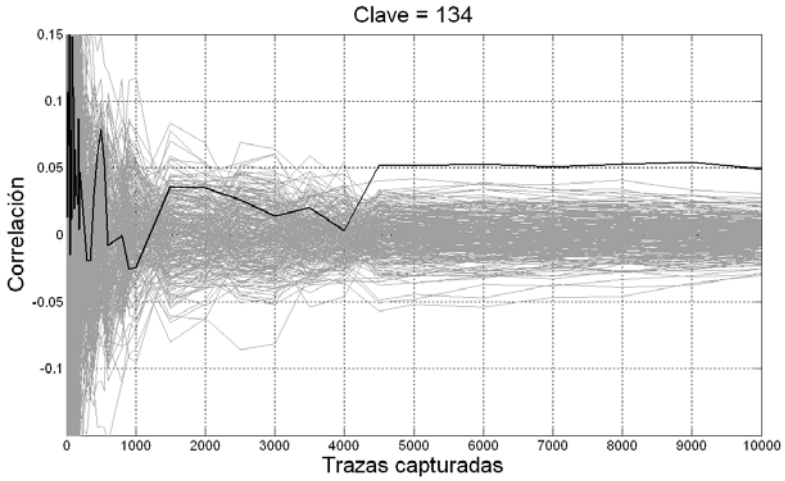

Figura 5.51:Análisis de correlación del sistema hardware no protegido. Atacado el byte 0 en el registro post *SubBytes* en la ronda 0. Representación en función de las trazas capturadas.

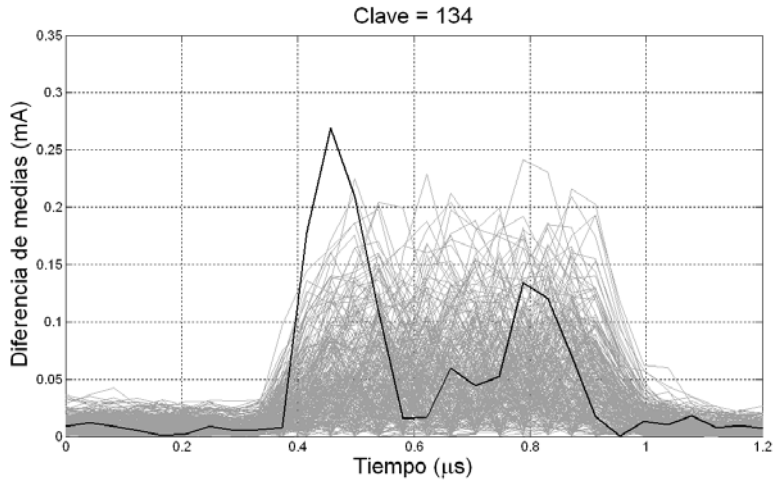

Figura 5.52:Análisis por diferencia de medias del sistema hardware no protegido. Atacado el byte 0 en el registro post *SubBytes* en la ronda 0.

5.5.3. Implementación del sistema protegido.

El sistema protegido parte de la estructura mostrada en la Figura 5.49 a la que se le ha añadido la parte que calcula la matriz de reenmascarado que aparece en la Figura 5.53. En el esquema se ha obviado la máquina de estados para simplificar la representación.

En el interior del módulo que calcula la matriz de reenmascarado aparecen las dos operaciones que lo forman. El bloque *MixCol* es idéntico al bloque *MixColumn* del algoritmo. En el bloque *SbTrans* se obtienen los valores realizando la búsqueda en 16 memorias BRAM en las que se alma-

cenan las 16 tablas $SBOX_{TRANS}$ correspondientes a cada uno de los elementos del estado.

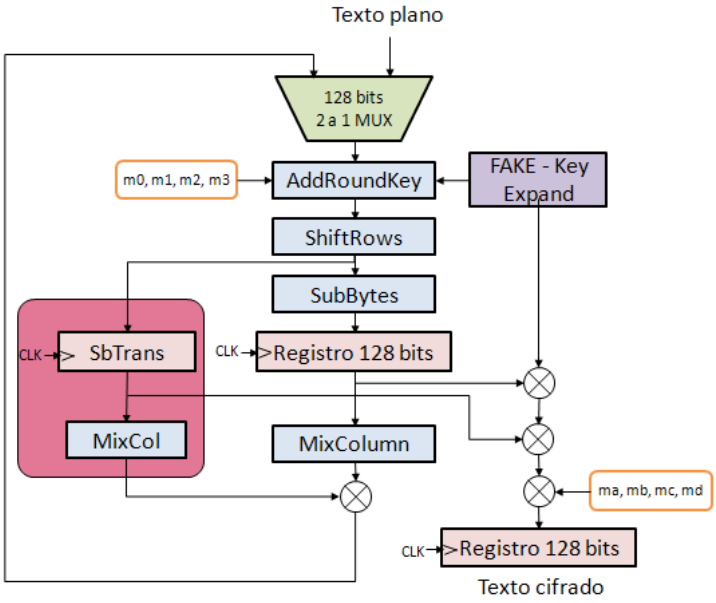

Figura 5.53: Estructura del sistema AES protegido en implementación Hardware.

Del mismo modo que en las versiones software, las tablas $SBOX_{TRANS}$ deben ser protegidas para evitar un ataque a la salida de las mismas. Para protegerlas se añade la matriz de mascaras aleatorias M_J con sus correspondientes valores: m_0, m_1, m_2 y m_3 (Apartado 3.5). Esto implica recalcular las tablas $SBOX_{TRANS}$ cada vez que esta matriz se modifica.

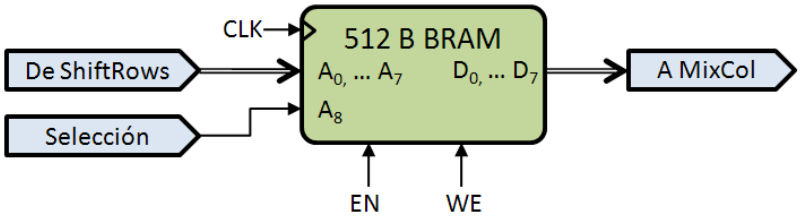

Figura 5.54: Implementación del bloque *SbTrans* (1 byte)

Puesto que se procesan 128 bits de forma concurrente, las tablas $SBOX_{TRANS}$ deben ser almacenadas en 16 memorias BRAM para poder ser leídas también de forma concurrente. Su reprogramación implica el recalculo de 256 bytes. Para no perjudicar al tiempo de procesado del algoritmo, cada unidad BRAM almacena 2 páginas de memoria. Una de ellas almacena los datos de las tablas mientras la otra está siendo precalculada

para ser usada en la ronda siguiente. En la Figura 5.54 se puede ver la implementación de cada uno de los bytes del estado, el registro de salida de la BRAM mantendrá el valor disponible para su posterior procesado en el bloque *MixCol*.

Usando el segundo puerto de la BRAM se realiza la reprogramación. El puerto se configura como lectura-escritura en 32 bits para poder acceder a los datos en bloques de 4 bytes. Ya que la reprogramación se realiza leyendo el contenido de la última tabla $\text{SBOX}_{\text{TRANS}}$ y efectuando la operación or exclusiva con la nueva matriz M_J, es necesario leer el valor y escribirlo al siguiente ciclo. Esto fuerza a utilizar 2 ciclos de reloj por cada 4 bytes reprogramados. El proceso se completará en 128 clk y se implementa tal como se muestra en la Figura 5.55.

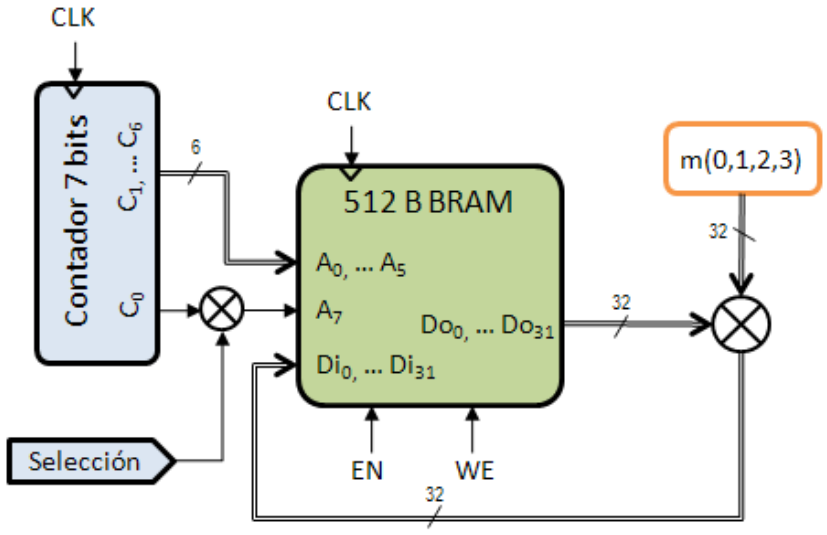

Figura 5.55: Implementación de la reprogramación de tabla SBOXTRANS (1 byte)

Hay que tener en cuenta que el ciclo de encriptado se resuelve en 10 ciclos de reloj mientras que la reprogramación de las tablas necesita 128 ciclos de reloj. Esto implica que, las tablas y la matriz de máscaras se actualizan cada 13 textos planos procesados, o lo que es lo mismo, se efectúan 13 encriptaciones con la misma matriz de máscaras M_J y M_K.

Contrariamente a lo que pueda parecer, esta particularidad no compromete la seguridad del sistema, dado que, para obtener un resultado positivo, es necesario capturar alrededor de 5000 trazas.

La Tabla 5.12 muestra los datos de implementación del sistema de referencia al que se le ha añadido la contramedida.

Parámetro	**Valor**
Frecuencia de reloj máxima	166 MHz
"Slices" consumidos	1535 de 7200 21%
"LUTS" consumidos	3985 de 28800 13 %
"FF" consumidos	2750 de 28800 9.5 %
"BRAM" consumidas	16 de 48 33 %

Tabla 5.12: Recursos de la FPGA consumidos por el sistema Hardware.

5.5.4. Ataque al registro post *SubBytes*.

Los resultados del análisis al sistema protegido demuestran el correcto funcionamiento de la contramedida. En la Figura 5.56 aparece el resultado del análisis de correlación. La clave máxima presenta una correlación de 0.045, un 40% superior al resto. Por el contrario, la correlación en la hipótesis de la clave real queda escondida entre el resto de correlaciones.

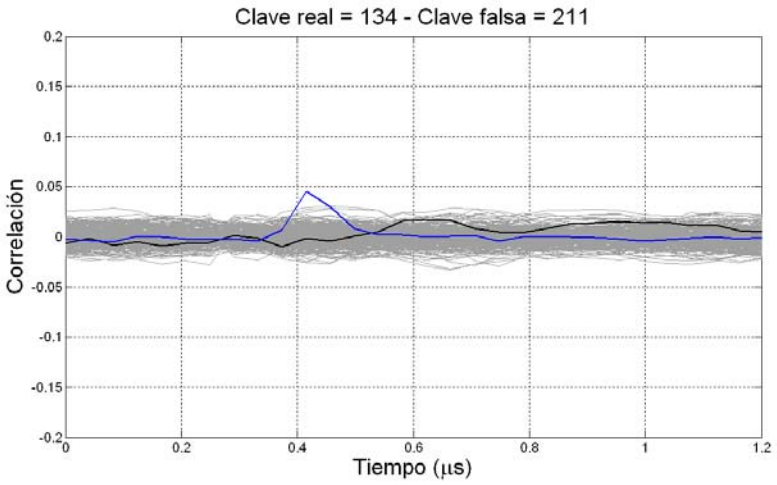

Figura 5.56:Análisis de correlación del sistema hardware protegido. Atacado el byte 0 en el registro post *SubBytes* en la ronda 0.

La clave falsa se diferencia de las demás a partir de la captura de 5000 trazas, tal y como se muestra en la Figura 5.57.

Finalmente el resultado del análisis por diferencia de medias también devuelve el falso positivo con una diferencia de medias máxima en la clave real de 0.24 mA, un 11% superior al resto (Figura 5.58).

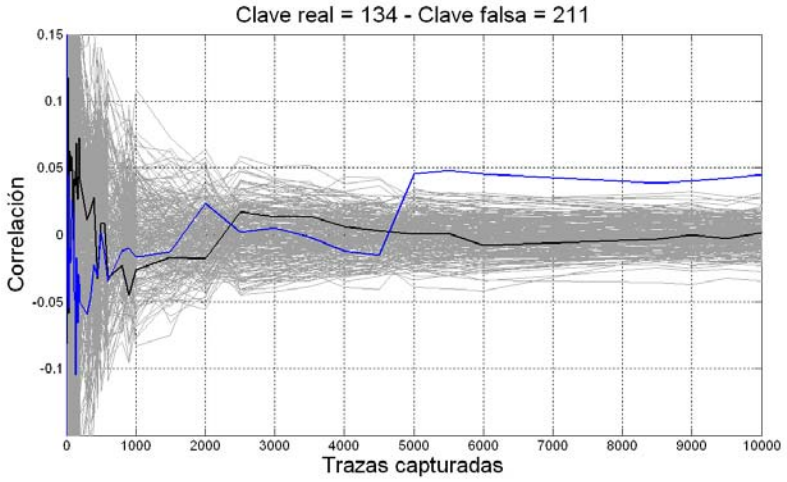

Figura 5.57:Análisis de correlación del sistema hardware protegido. Atacado el byte 0 en el registro post *SubBytes* en la ronda 0. Representación en función de las trazas capturadas.

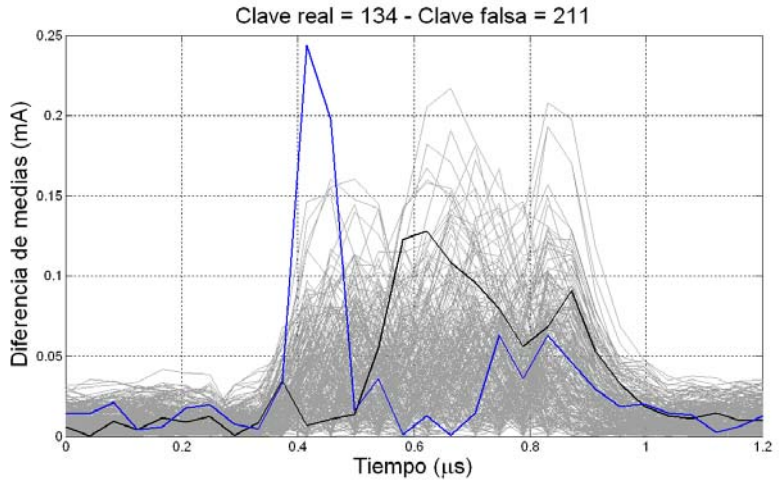

Figura 5.58:Análisis por diferencia de medias del sistema hardware protegido. Atacado el byte 0 en el registro post *SubBytes* en la ronda 0.

5.5.5. Análisis de resultados

La implementación en el caso del sistema totalmente hardware resuelve, tanto el algoritmo como el cálculo de la matriz de reenmascarado, en bloques de 128 bits y a razón de una ronda del algoritmo en cada ciclo de reloj.

La Tabla 5.13 presenta un resumen de los recursos y las prestaciones de cada uno de los sistemas hardware implementados, el de referencia y el que incorpora la contramedida. En cuanto al tiempo necesario para procesar un texto plano, no existe variación alguna dado que ambos sistemas

resuelven el algoritmo en 10 ciclos de reloj. En la plataforma de trabajo que se ha utilizado, la frecuencia de reloj es de 24 MHz. Esto implica un tiempo de procesado de 416 ns por cada texto plano, lo que significa $2.4 \cdot 10^6$ textos planos cada segundo.

Parámetro	**Sistema hardware de referencia**	**Sistema Hardware**	**Variación**
Frec. de reloj máxima	192 MHz	166 MHz	-13.5 %
"Slices" consumidos	1193 (15 %)	1535 (21 %)	27.7 %
"LUTS" consumidos	2684 (9.3 %)	3985 (13 %)	48.5 %
"FF" consumidos	2610 (9 %)	2750 (9.5 %)	5.4 %
36K BRAMS	0 (--)	18 (33.3 %)	--

Tabla 5.13:Comparación de recursos consumidos por el sistema hardware de referencia y el sistema hardware con la contramedida implementada. Entre paréntesis el porcentaje respecto al total de la FPGA.

La adición de la contramedida penaliza el funcionamiento, tanto en recursos consumidos como en velocidad máxima de trabajo.

Es posible realizar la comparación con otras contramedidas a nivel de hardware. [Zen'2012] propone la implementación hardware de la función *Sbox* mediante el uso de puertas enmascaradas en un dispositivo Virtex 5. Esta solución represente un aumento de recursos de la FPGA consumidos de 482 % y una disminución de la velocidad máxima de proceso de 33%.

Utilizando WDDL [Kaz'2012] implementa al algoritmo utilizando un aumento del 100% del área consumida respecto de la versión no protegida, que él mismo implementa. Este resultado es previsible al tratarse de una contramedida que se basa en el procesado del dato y de su complementario para mantener constante el número de transiciones en cada ciclo.

El campo de aplicación de esta implementación sería aquel en que fuese necesario procesar grandes cantidades de datos en muy poco tiempo. Ejemplo de este caso son: las comunicaciones digitales a través de canales no seguros, la protección de documentos de gran volumen, la protección de contenidos multimedia o el procesado digital de señales, entre otros.

5.6. Conclusiones

Una nueva contramedida frente a los ataques por análisis de consumo se ha implementado en tres casos diferentes: un primer sistema totalmente software, un segundo sistema basado en diseño combinado software/hardware y un tercer sistema puramente hardware.

En los tres casos se ha usado un dispositivo FPGA Virtex5 para realizar las pruebas y corroborar las hipótesis planteadas en el capítulo 3.

La efectividad de la contramedida se basa en que revela una falsa clave, a diferencia de otras contramedidas que tratan de eliminar la relación entre el dato procesado y el consumo.

6. Conclusiones y futuras líneas de trabajo

Se presentan a continuación las conclusiones más relevantes que se derivan de esta tesis doctoral, así como las líneas de trabajo sobre las que orientar una investigación futura.

6.1. Conclusiones generales

En esta tesis doctoral se expone un nuevo paradigma en el ámbito de la protección hardware de dispositivos ante ataques por análisis de consumo. El trabajo de investigación se ha dirigido a conseguir que un dispositivo bajo ataque retorne una falsa clave, haciendo inútiles los esfuerzos del atacante por desvelar la clave real de trabajo del criptoprocesador.

Con este objetivo, en el capítulo 2 se muestran las bases teóricas que permiten realizar los ataques por análisis de consumo en dispositivos criptográficos. Para ello, se describen dos modelos de consumo, distancia y altura de Hamming, que han demostrado ser validos para sistemas implementados a partir de celdas CMOS. Posteriormente, se introducen los análisis simple y diferencial del consumo. Dentro del análisis diferencial se distingue entre el análisis de correlación lineal y el análisis por diferencia de medias. Estos dos últimos tipos de ataque serán las herramientas utilizadas para verificar el funcionamiento de la contramedida en la parte práctica.

En este mismo capítulo se muestran las generalidades del "Masking y del "Hiding". Estas son las contramedidas habituales que se aplican en los sistemas criptográficos para protegerlos ante los ataques por análisis de consumo. También se describen cuáles son los puntos débiles de cada uno de estos métodos de protección, y el modo en que un atacante puede utilizarlos para descubrir la clave de trabajo.

Es en el capítulo 3 donde se ha desarrollado el fundamento teórico de la contramedida propuesta, a la que se ha denominado "Faking". Ésta consiste en hacer que el algoritmo AES procese el texto plano con una clave falsa, en lugar de utilizar la clave real. El hecho de usar una clave falsa para cifrar o descifrar datos provoca que el resultado no sea el esperado. Por esta razón, el sistema debe realizar algunas operaciones adicionales para conseguir que al resultado sea el correcto.

Finalmente, en este capítulo se exponen las particularidades del "Faking" en función de los tres escenarios que se han planteado para su implementación. En primer lugar se ejecuta al algoritmo AES y la contramedida en un procesador de 32 bits, lo que se ha denominado solución software. En el segundo caso, se recurre a un sistema basado en un diseño hardware/software y en el que el algoritmo se ejecuta en el procesador, mientras que la contramedida se resuelve en un coprocesador hardware diseñado a tal efecto. En último lugar, tanto el algoritmo como la contramedida se han implementando en un sistema totalmente Hardware. Para las tres soluciones se ha utilizado una FPGA Virtex 5 de Xilinx.

A la luz del planteamiento teórico hay que señalar que, la protección de un dispositivo criptográfico mediante "Faking" no excluye la implementación simultánea de otras contramedidas que actúen tanto en el eje temporal como en la amplitud. Por ejemplo, este método puede combinarse con un enmascaramiento (Apartado 2.9.2), de forma que un ataque de segundo orden (Apartado 2.10) retornase una clave falsa. Del mismo modo, puede combinarse con cualquier contramedida basada en estructuras de procesamiento dual.

Existen otras publicaciones que presentan sistemas que tienen por objeto introducir ruido correlacionado conjuntamente con el consumo generado por el sistema criptográfico. En esta línea [Naj'2009] propone añadir

un generador de ruido correlacionado que opere de forma concurrente con el algoritmo a proteger. En cambio, este sistema no elimina la información explotable de la traza de consumo, en tanto en cuanto, el algoritmo continúa ejecutándose con su clave correcta. Además, la protección es únicamente efectiva si se ataca la salida de la función correlacionada con el ruido añadido y, se da la circunstancia de que el resultado obtenido depende de la cantidad de trazas tomadas.

Contrariamente a lo anterior, la propuesta de esta tesis elimina la información explotable de la traza de consumo, al no utilizar la clave real en el procesado del algoritmo y al enmascarar aquellos puntos del sistema en los que pueda quedar expuesta. Además, el sistema resulta protegido con independencia del punto elegido para el ataque.

Es importante hacer notar que, el aumento de las trazas capturadas no hace más que fortalecer la correlación en la hipótesis de la clave falsa por encima de las demás. Puesto que la correlación sobre la clave real se oculta entre el resto de hipótesis, aumentar la cantidad de capturas hace aún más difícil la diferenciación de la clave real y, por ende, el sistema es más seguro.

Una de las características más destacables de esta contramedida es que su particular estructura le permite acciones que dificultan el trabajo del atacante. Fruto de ellas pueden mencionarse las siguientes:

a) Es posible modificar la clave real de trabajo sin modificar la clave falsa, de modo que el atacante continúa obteniendo la correlación máxima en la misma clave falsa aunque se haya modificado la clave de encriptado.

b) También se puede modificar la clave falsa manteniendo la clave real. En este caso el atacante puede llegar a la conclusión errónea de que se ha modificado la clave de trabajo.

c) O bien, es posible modificar ambas claves.

Ante esta casuística, un supuesto atacante no tiene elementos para determinar si se ha producido uno u otro de los casos anteriores.

En el capítulo 4 se describe el banco de trabajo diseñado y desarrollado para la captura automática de las trazas de consumo. Este proceso se realiza enviando textos al dispositivo a testear, para ser encriptados, y capturando el consumo eléctrico producido mediante un osciloscopio digital y una sonda de corriente. Los valores que forman cada una de las trazas se almacenan en un ordenador a la espera de ser procesadas.

La plataforma de trabajo elegida (SAEBO-GII) ha demostrado ser un entorno de trabajo ideal para este tipo de análisis, dado que su particular diseño permite capturar trazas con una muy alta relación señal ruido. Esta característica hace que sea posible comprobar el funcionamiento de la contramedida mediante la captura de un reducido número de trazas.

El capítulo 5 se centra en verificar experimentalmente el funcionamiento de la contramedida. Para ello se implementan, por un lado sistemas sin contramedidas para usarlos de referencia y, por otro lado, sistemas con "Faking". Las pruebas se han realizado sobre los tres escenarios diferentes introducidos en el capítulo 3: totalmente software, diseño software/hardware y totalmente hardware. Estos escenarios son representativos de una gran variedad de sistemas criptográficos presentes en múltiples dispositivos, tales como: tarjetas inteligentes, dispositivos encriptados de almacenamiento, sistemas de identificación biométrica, sistemas de control de acceso, etc.

Los resultados experimentales demuestran que la contramedida consigue su objetivo principal, es decir, oculta la correlación relativa a la hipótesis planteada sobre la clave real entre las correlaciones relativas al resto de hipótesis. Hay que tener en cuenta que, el hecho de que el sistema muestre una correlación alta en una hipótesis falsa, consigue confundir al atacante en un primer momento. Una vez que éste haya comprobado que la clave obtenida no es la correcta será consciente de que se está falseando el resultado.

De la parte experimental se extrae que la contramedida propuesta no requiere una gran cantidad de recursos adicionales, en comparación con los recursos necesarios para implementar las contramedidas "Masking" y "Hiding". En el sistema software, el tiempo y la memoria necesarios para resolver el "Faking" están en la línea de lo que representa implementar las

otras contramedidas (Apartado 5.3.4). En el segundo caso, cuando la contramedida se implemente en un coprocesador hardware (Apartado 5.4.6), resulta ventajoso implementar la contramedida propuesta en esta tesis respecto de las otras dos. Finalmente, en la implementación totalmente hardware (Apartado 5.5.5), la nueva contramedida también resulta ventajosa respecto a la implementación de “Masking” o “Hiding”.

6.2. Futuras líneas de trabajo.

Para complementar la investigación iniciada en esta tesis doctoral se enuncian algunas propuestas de futuros trabajos en el área de la protección de dispositivos criptográficos:

a) **Evaluación del funcionamiento de la contramedida ante otros tipos de ataques.** El ataque por análisis de consumo no es la única forma de llegar a la obtención fraudulenta de la clave de un sistema criptográfico. En la literatura científica se proponen ataques (EMA) [Mar'2013] en los que se explota la radiación electromagnética emitida por el dispositivo. También se realizan ataque por inducción de fallos [Bar'2010] [Ber'2005]; en ellos se observa la evolución del dispositivo tras la inducción de un error controlado en un punto del algoritmo con la intención de limitar los posibles valores de la clave. [Agr’2003] Propone la realización de ataques multicanal en los que se explota simultáneamente información procedente de diversos canales o bien dispone de una caracterización muy precisa del consumo del dispositivo.

b) **Implementación de la contramedida en tarjetas inteligentes y evaluación de su funcionamiento.** Las tarjetas inteligentes, o “Smart Card”, representan una herramienta ampliamente utilizada para la identificación de personas en el control de acceso, la realización de transacciones bancarias o el acceso a servicios personales, entre otros [Ide'2015]. Estas tarjetas almacenan información del usuario que es sensible de ser atacada con fines delictivos y, por tanto, se convierten en objetivo de ataques por análisis de consumo.

Las tarjetas inteligentes pueden ser meros contenedores de memoria en las que almacenar datos encriptados a los que se puede acceder cuando la información es leída por algún otro dispositivo. Este dispositivo es el encargado de ejecutar los algoritmos criptográficos para codificar la información y, por tanto, es el dispositivo que debe ser protegido.

Otros tipos de tarjetas inteligentes incorporan microprocesadores embebidos, de hasta 32 bits, junto con memoria RAM, ROM y EEPROM para almacenar datos temporales, programa y/o datos de identificación. Estas tarjetas tienen capacidad para ejecutar algoritmos criptográficos y, por tanto, son susceptibles de ataque.

Finalmente las tarjetas inteligentes criptográficas incorporan, a parte de un microprocesador, módulos hardware para resolver cifrados y gestionar firmas digitales, como por ejemplo las que emite la FNMT[54] española.

c) **Verificación del funcionamiento de la contramedida en otras implementaciones del algoritmo AES**. El algoritmo "Advanced Encription Standard" puede ser implementado de múltiples formas, a parte de las ya documentadas en esta tesis doctoral. La casuística es muy variada y depende de las necesidades de trabajo o de la cantidad de datos que deban ser procesados. Desde procesadores PIC®[55] de 8 bits asociados a pequeñas aplicaciones industriales hasta complejos dispositivos ASIC diseñados para aplicaciones específicas pueden incorporar esta contramedida.

También es posible realizar la implementación mediante estructuras pipeline cuyo objetivo es aprovechar al máximo los recursos en aplicaciones hardware.

Por otro lado, Según la norma publicada en [NIST'2001], este algoritmo admite longitudes de claves de 128, 192 o 256 bits. La investigación realizada en esta tesis se ha centrado en la versión de 128

[54] Fábrica Nacional de Moneda y Timbre.

[55] ® Microchip Technology Inc.

bits y sería interesante analizar el funcionamiento de la contramedida en las versiones de 196 y 256 bits de clave.

Bibliografía

[AGI'2008] Agilent Technologies **"Osciloscopios de la serie 1000 de Agilent, guia de usuario"**, 2008

[AGI'2009] Agilent Technologies **"Agilent E2094S IO Libraries Suite 15.5 Data Sheet"**, 2009

[Agr'2003] Dakshi Agrawal, Josyula R. Rao, and Pankaj Rohatgi **"Multi-channel Attacks"** Cryptographic Hardware and Embedded Systems - CHES 2003
Lecture Notes in Computer Science Volume 2779, 2003, pp 2-16

[AIST'2009] Research Center for Information Security, National Institute of Advanced Industrial Science and Technology **"Side-channel Attack Standard Evaluation Board SASEBO-GII Specification"**

[Bae'2005] Yoo-Jin Baek, Mi-Jung Noh. SoC R&D Center, Samsung Electronics, Korea **"Differential Power Attack And Masking Method"** Trends in Mathematics, Information Center for Mathematical Sciences, Volume 8, Number 1, June, 2005, Pages 1–15

[Bar'2010] Alessandro Barenghi, Guido M. Bertonit, Luca Breveglieri, Mauro Pellicioli and Gerardo Pelosi **"Fault Attack on AES with Single-Bit Induced Faults"** Information Assurance and Security (IAS), 2010 Sixth International Conference on. pp 167 - 172. Agosto 2010

[Ben'2012] Noura Benhadjyoussef , wajih El hadj youssef, Mohsen Machhout , Rached Tourki **"A compact 32-Bit AES design for embedded system"** Design & Technology of Integrated Systems in Nanoscale Era (DTIS), 2012 7th International Conference on pags. 1-4

[Ber'2005] Guido Bertoni, Vittorio Zaccaria, Luca Breveglieri, Matteo Monchiero, Gianluca Palermo **"AES Power Attack Based on Induced Cache Miss and Countermeasure"** Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on. 586 - 591 Vol. 1 Abril 2005

[Bri'2004] Eric Brier, Christophe Clavier, and Francis Olivier **"Correlation Power Analysis with a Leakage Model"** In prceedings of CHES 2004 LNCS3156, pp 16-29, Springer-Verlag 2004

[Buc'2011] Marco Bucci, Luca Giancane, Raimondo Luzzi, Giuseppe Scotti, Alessandro Trifiletti **"Delay-Based Dual-Rail Precharge Logic"** Very Large Scale Integration (VLSI) Systems, IEEE Transactions on (Volume:19 , Issue: 7) pp.1147 - 1153. 2011

[Cha'1999] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, Pankaj Rohatgi **"Towards Sound Approaches to Counteract Power-Analysis Attacks"** Advances in Cryptology CRYPTO' 99 Lecture Notes in Computer Science Volume 1666, 1999, pp 398-412

[Cor'2000] J.S.Coron, P. Kocher, D.Naccache: "**Statistics and Secret Leakage**". In proceedings of Financial Cryptography, LNCS 1962, pp 157-173, Sprinter-Verlag, 2000.

[Dae'1999] Joan Daemen, Vincent Rijmen, **"AES proposal: Rijndael"** http://csrc.nist.gov/archive/aes/rijndael/Rijndaelammended.pdf

[Dai'2004] Daisuke Suzuki, Minoru Saeki, and Tetsuya Ichikawa **"Random Switching Logic: A Countermeasure against DPA based on Transition Probability"** International Association for Cryptologic Research, 2004.

[Dai'2006] Daisuke Suzuki, Minoru Saeki, **"Security Evaluation of DPA Countermeasures Using Dual-Rail Pre-charge Logic Style"** Cryptographic Hardware and Embedded Systems - CHES 2006, pp 255-269

[Dar'2014] Waltenegus Dargie; "**A Stochastic Model for Estimating the Power Consumption of a Processor**" Computers, IEEE Transactions on (Volume:PP , Issue: 99) pp 1.

[Dee'2014] Deevi Radha Rani, S. Venkateswarlu **"Implementation of Power Analysis Attack using SASEBO-W"** Deevi Radha Rani et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (3) , 2014, 3994-3997

[Deg'2005] Degalahal, V. ;"**Methodology for High Level Estimation of FPGA Power Consumption** "Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific (Volume:1) pp 657-660

[Des'2012] Deschamps, Jean-Pierre, Sutter, Gustavo D., Cantó, Enrique **"Guide to FPGA Implementation of Arithmetic Functions"** Lecture Notes in Electrical Engineering, Volumen 149. Springer Netherlands 2012

[Dif'1976] Whitfield Diffie and Martin E. Hellman **"New Directions in Cryptography"** Information Theory, IEEE Transactions on (Volume:22 , Issue: 6) 644-654 Nov 1976

[FTDI'2010] Future Technology Devices International Ltd **"FT2232D Dual USB to Serial UART/FIFO IC Data Sheet"**, 2010

[Goo'2005] Tim Good and Mohammed Benaissa **"AES on FPGA from the Fastest to the Smallest "** Cryptographic Hardware and Embedded Systems – CHES 2005 pags. 427-440

[Gra'2006] Gibrán Granados Paredes **"Introduccióon a la criptografía"** Revista Digital Universitaria, 10 de julio 2006 • Volumen 7 Número 7 http://www.revista.unam.mx/vol.7/num7/art55/int55.htm (Abril 2015)

[Gun'2011] Tim Güneysu and Amir Moradi. "**Generic Side-Channel Countermeasures for Reconfigurable Devices**" B.Preneel and T.Takagi (Eds.) CHES 2011, LNCS 6917, pp 33-48, 2011. © International Association for Cryptologic Research 2011.

[Han'2008] Yu HAN, Xuecheng ZOU, Zhenglin LIU, Yicheng CHEN "**Efficient DPA Attacks on AES Hardware Implementations**" I. J. Communications, Network and System Sciences. 2008; 1: 1-103

[He'2014] Wei He **"Side-Channel Attack Protection Techniques in FPGA Systems using Enhanced Dual-Rail Solutions"** Tesis doctoral - Universidad Politécnica de Madrid - 2014

[Her'2011] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard **"An AES Smart Card Implementation Resistant to Power Analysis Attacks"** Cryptographic Hardware and Embedded Systems – CHES 2011 Lecture Notes in Computer Science Volume 6917, 2011, pp 95-107

[Hiu'2011] Liu Huiying, Wang Tao, Zhao Xinjie, Wu Kehui. "**Algebraic Side-Channel Attack on SMS4 Key Schedule**" Instrumentation, Measurement, Computer, Communication and Control, 2011 First International Conference on pp 553 556

[Hok'2008] Chun Hok Ho, Philip H.W. Leong, Wayne Luk, Steven J.E. Wilton, "**RAPID ESTIMATION OF POWER CONSUMPTION FOR HYBRID FPGAS**" Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on pp. 227-232

[Hua'2007] Chi-Wu Huang, Chi-Jeng Chang, Mao-Yuan Lin and Hung-Yun Tai **"The FPGA Implementation of 128-bits AES Algorithm Based on Four 32-bits Parallel Operation"** Data, Privacy, and E-Commerce, 2007. ISDPE 2007. The First International Symposium on pags. 462 - 464

[Hwa'2006] Hwang, D.D. ; Tiri, K. ; Hodjat, A. ; Bo-Cheng Lai, more authors **"AES-Based Security Coprocessor IC in 0.18-µm CMOS With Resistance to Differential Power Analysis Side-Channel Attacks"** Solid-State Circuits, IEEE Journal of (Volume:41 , Issue: 4) pp.781 - 792. 2006

[Ide'2015] **"Idegamili S.A. Tecnologia y seguridad en identificación"** http://www.idgalimi.com.ar/ (Última consulta Abril 2015)

[Ito'2012] Hiroki Ito, Mitsuru Shiozaki, Anh-Tuan Hoang, Takeshi Fujino. "**Efficient DPA-Resistance Verification Method with Smaller Number of Power Traces on AES Cryptographic Circuit**" Digital System Design (DSD), 2012 15th Euromicro Conference on pp 735-738

[Jen'2007] Chi-Jeng Chang, Chi-Wu Huang, Hung-Yun Tai, Mao-Yuan Lin and Teng-Kuei Hu **"8-bit AES FPGA Implementation using Block RAM"** Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE. Pág. 2654 - 2659

[Jev'2011] Jevtic, R., Carreras, C.: "**Power Measurement Methodology for FPGA Devices**" Instrumentation and Measurement, IEEE Transactions on (Volume:60 , Issue: 1) pp. 237 – 247. Enero 2011.

[Kad'2011] Kadir, S.A. ; Electr. Eng., Bandung Inst. of Technol., Bandung, Indonesia ; Sasongko, A. ; Zulkifli, M."**Simple Power Analysis Attack Against Elliptic Curve Cryptography Processor on FPGA Implementation**" Electrical Engineering and Informatics (ICEEI), 2011 International Conference on pp. 1-4

[Kap'2010] Kaps J., ; Velegalati R. **"DPA Resistant AES on FPGA Using Partial DDL"** Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on pp. 273 - 280. 2-4 May 2010

[Kaz'2012] Kazuyuki Tanimura, Nikil D. Dutt,**"HDRL: Homogeneous Dual-Rail Logic for DPA Attack Resistive Secure Circuit Design"** Embedded Systems Letters, IEEE (Volume:4 , Issue: 3) pp 57 - 60. Sept. 2012

[Koc'1996] P. Kocher **"Timing Attacks on Implementations of Di e-Hellman, RSA, DSS, and Other Systems"**. In proceedings of CRYPTO 1996, LNCS 1109, pp 104-113, 1996.

[Koc'1999] Paul Kocher, Joshua Jaffe, Benjamin Jun "**Differential Power Analysis**" Advances in Cryptology — CRYPTO' 99 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings pp. 388 - 397

[Koe'2005] François Koeune and François-Xavier Standaert "**A Tutorial on Physical Security and Side-Channel Attacks**" FOSAD 2004/2005, LNCS 3655, pp. 78-108, Sprinter-Verlag, 2005.

[Kul'2004] Konrad J. Kulikowski, Mark G. Karpovsky, Alexander Taubin, **"Power Attacks on Secure Hardware Based on Early Propagation of Data",** Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on (Volume:1) pp I-261 - I-268.

[Li'2008] Huiyun Li, Keke Wu, Bo Peng, Yiwei Zhang, Xinjian Zheng and Fengqi Yu"**Enhanced Correlation Power Analysis Attack on Smart Card**" Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for pp.2143 - 2148

[Li'2010] Huiyun Li, Keke Wu and Fengqi Yu "**Enhanced correlation power analysis attack against trusted systems**" Security and Communication Networks Volume 4, Issue 1, Article first published online: 13 JAN 2010

[Li'2011] Huiyun Li ; Shenzhen Institutes of Adv. Technol., Chinese Univ. of Hong Kong, Shenzhen, China ; Keke Wu ; Guoqing Xu ; Hai Yuan and more authors "**Simple Power Analysis Attacks Using Chosen Message against ECC Hardware Implementations**" Internet Security (WorldCIS), 2011 World Congress on pp 68-72

[Lu'2009] Jiquiang Lu, Ling Pan and jerry den Hartog **"Security of AES Against First and Second-Order Differential Power Analysis"** 4th Benelux Workshop on Information and System Security. 2009.

[Lum'2013] Rubén Lumbiarres-López, Mariano López-García, Enrique F. Cantó-Navarro. "**Ataques por canal lateral sobre el algoritmo de encriptación AES implementado en MicroBlaze**""Actas de las XIII Jornadas de Computación Reconfigurable y Aplicaciones JCRA 2013". Madrid: 2013, p. 105-112.

[Lum'2014] Rubén Lumbiarres-López, Mariano López-García, Enrique F. Cantó-Navarro.**"Implementation on MicroBlaze of AES Algorithm to Reveal Fake Keys Against Side-Channel Attacks"** Industrial Electronics (ISIE), 2014 IEEE 23rd International Symposium on pp.1882 - 1887

[Man'2003] Stefan Mangard "**A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion**" Information Security and Cryptology — ICISC 2002 Lecture Notes in Computer Science Volume 2587, 2003, pp 343-358

[Man'2007] S. Mangard, E. Oswald, T. Popp "**Power Analysis Attacks - Revealing the secrets of Smart Cards**". Springer Science+Busines Media, 2007 ISBN-13:978-0387-30857-9

[Mar'2013] Roberto Martínez Bejarano, **"Evaluación de la seguridad de sistemas embebidos ante ataques EMA"** Tedis doctoral publicada en el departamento de tecnologia electrònica de la Universidad Carlos III de Madrid. Noviembre 2013

[Mas'2010] Masoomi, M.; Masoumi, M. ; Ahmadian, M. "**A Practical Differential Power Analysis Attack against an FPGA Implementation of AES Cryptosystem**" Information Society (i-Society), 2010 International Conference on pp 308-312

[Mat'2014] The MathWorks, Inc. **"MATLAB The Language of Technical Computing Documentation" ,** 1994-2015

[Mes'2000] Thomas S. Messerges. **"Using Second-Order Power Analysis to Attack DPA Resistant Software"** Cryptographic Hardware and Embedded Systems — CHES 2000 Lecture Notes in Computer Science Volume 1965, 2000, pp 238-251.

[Mey'2010] Meynard, O. "**Far Correlation-based EMA with a Precharacterized Leakage Model**" Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010 pp 977-980

[Mok'2012] Mokhov, A.; Khomenko, V. ; Sokolov, D. ; Yakovlev, A. **"On Dual-Rail Control Logic for Enhanced Circuit Robustness"**Application of Concurrency to System Design (ACSD), 2012 12th International Conference on pp. 112 - 121

[Naj'2009] Najeh Kamoun, Lilian Bossuet, and Adel Ghazel, **"Correlated Power Noise Generator as a Low Cost DPA Countermeasure to Secure Hardware AES Cipher"** in Proceeding of the 3rd IEEE International Conference on Signals, circuits and Systems, SCS 2009, pp. 1-6, Djerba, Tunisa, November 2009., Tunisia (2009)"

[NIST'1977] U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology "**DATA ENCRYPTION STANDARD (DES)**" (FIPS PUB 46)

[NIST'2001] U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology "**ADVANCED ENCRYPTION STANDARD (AES)**" (FIPS PUB 197)

[Niy'2007] Niyaz PK **"Advanced Encryption Standard (AES) Implementation in C/C++ with comments."** Hoozi Resources website of free downloads, source code, technical articles and resources. http://www.hoozi.com/. Última visita Enero 2015.

[Pee'2007] Eric Peeters , François-xavier St , Jean-jacques Quisquater "**Power and Electromagnetic Analysis: Improved Model, Consequences and Comparisons**" In Integration, the VLSI journal Vol.40, pp. 52-60, 2007.

[Pop' 2002] Thomas Popp and Stefan Mangard **"Masked Dual-Rail Pre-Charge Logic DPA-Resistance without Routing Constraints"** In prcoeedings CHES 2005 vol 3659

[Pro'2009] Emmanuel Prouff, Matthieu Rivain, Regis Bévan **"Statistical Analysis of Second Order Differential Power Analysis"** Computers, IEEE Transactions on (Volume:58 , Issue: 6) pp.799 - 811

[Qia'2013] Qiang Liu, Zhenyu Xu, and Ye Yuan **"A 66.1 Gbps Single-pipeline AES on FPGA"** Field-Programmable Technology (FPT), 2013 International Conference on pp 378 - 381

[Qui'2005] Jean-Jacques Quisquater & François-Xavier Standaert: "**Exhaustive Key Search of the DES: Updates and Refinements**". SHARCS'05. http://2005.sharcs.org/ Ultima visita 09/2014

[Ren'2009] Mathieu Renauld, Fran¸cois-Xavier Standaert "**Algebraic Side-Channel Attacks**", Cryptology ePrint Archive, report 2009/179, http://eprint.iacr.org/2009/279 última consulta 4-10-14

[Sam'2002] David Samyde, Sergei Skorobogatov, Ross Anderson and Jean-Jacques Quisquater “On a New Way to Read Data from Memory”Security in Storage Workshop, 2002. Proceedings. First International IEEE. Pp 65-69.

[San'2012] Enrique Sánchez Acosta **"Criptoanálisis más utilizados en la actualidad"** Universidad Francisco Vitoria

[Sok'2005] Danil Sokolov, Julian Murphy,Alexander Bystrov, Alex Yakovlev, **"Design and Analysis of Dual-Rail Circuits for Security Applications"** Computers, IEEE Transactions on (Volume:54 , Issue: 4) pp 449-460. 2005

[Song'2008] Song Sun, Zijun Yan, Zambreno, J. "**Experiments in attacking FPGA-based embedded systems using differential power analysis**" Electro/Information Technology, 2008. EIT 2008. IEEE International Conference on. Pp 7-12

[Sri'2012] P.V. Sriniwas Shastry, Amruta Kulkarni, Mukul S.Sutaone **"ASIC Implementation of AES"** India Conference (INDICON), 2012 Annual IEEE. Pp.1255 - 1259

[Sta'2005] William Stallings **Cryptography and network security: principles and practice - fifth edition"** Editorial Prentice Hall. 2005, ISBN: 978-0-13-609704-4

[Suz'2006] Daisuke Suzuki and Minoru Saeki Mitsubishi Electric Corporation, Information Technology R&D Center "**Security Evaluation of DPA Countermeasures Using Dual-Rail Pre-charge Logic Style**" In CHESS 2006

[TEK'2014] TEKTRONIX, INC - **"AC Current Probes CT1, CT2, CT6 Data Sheet"**, 2014

[Tia'2012] Qizhi Tian, Sorin A. Huss "**On the Attack of Misaligned Traces by Power Analysis Methods**" Computer Engineering & Systems (ICCES), 2012 Seventh International Conference on pp. 28 - 34

[Tir'2004] Kris Tiri and Ingrid Verbauwhede, **"A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation"** Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings (Volume:1) 246 - 251

[Tir'2004b] Kris Tiri and Ingrid Verbauwhede **"Secure Logic Synthesis"** Field Programmable Logic and Application, 14 th International conference, FPL 2004, pp. 1052-1056.

[Tir'2005] Tiri K., Hwang D.D., Hodjat, A, Bo-Cheng Lai et al **"AES-based cryptographic and biometric security coprocessor IC in 0.18-µm CMOS resistant to side-channel power analysis attacks"** VLSI Circuits, 2005. Digest of Technical Papers. 2005 Symposium on 216 - 219. 16-18 June 2005.

[Tos'2010] Toshihiro Katashita, Akashi Satoh, Katsuya Kikuchi, Hiroshi Nakagawa, Masahiro Aoyagi. **"Evaluation of DPA Characteristics of SASEBO for Board Level Simulations"** COSADE 2010 - First International Workshop on Constructive Side-Channel Analysis and Secure Design

[Vel'2008] Rajesh Velegalati, Panasayya S V V K Yalla. "**Differential Power Analysis Attack on FPGA Implementation of AES**"Technical Reports, George Mason University - ECE Department, May, 2008

[Wan'2012] Chenxu Wang, Mingyan Yu, Jinxiang Wang, Peihe Jiang, Xiaochen Tang. "**A more practical CPA attack against present hardware implementation**" Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on (Volume:03) pp 1248 - 1253.

[Wan'2014] Yi Wang and Yajun Ha **"A Performance and Area Efficient ASIP for Higher-Order DPA-Resistant AES"** Emerging and Selected Topics in Circuits and Systems, IEEE Journal on (Volume:4 , Issue: 2) pp 190-202

[XIL'2001] XILINX tecnical specifications **"32-bit Processor Local Bus Architecture Specifications Version 2.9"** 1996-2001

[XIL'2010] XILINX tecnical specifications **"LogiCORE IP Processor Local Bus (PLB) v4.6 (v1.05a) Product Specification"**, 2010

[XIL'2010b] XILINX tecnical specifications **"XPS UART Lite (v1.01a) Product Specification"** April, 2010

[XIL'2010c] XILINX tecnical specifications **"Virtex-5 Libraries Guide for HDL Designs",** Diciembre 2010

[XIL'2011] XILINX tecnical specifications **"MicroBlaze Processor Reference Guide UG081 (v12.0)"** Xilinx EDK 13.1 release. 2011

[XIL'2011b] XILINX tecnical specifications **"LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11d)"** Marzo, 2011

[XIL'2011c] XILINX tecnical specifications **"IP Processor Block RAM (BRAM) Block (v1.00a)"** Marzo 2011

[Yoh'2013] Yohei Hori, Toshihiro Katashita, Akihiko Sasaki and Akashi Satoh **"A First Report on Electromagnetic and Power Analysis Attacks against a 28-nm FPGA Device"** Information--An International Interdisciplinary Journal, Vol.16, No.8(B), pp.5993-6006, 2013

[Yu'2007] Pengyuan Yu, **"Implementation of DPA-Resistant Circuit for FPGA"** Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering, Abril-2007

[Yua'2010] Zheng Yuanyuan, Wang Tao "**Research in Correlation Algebraic Attack on AES First Round**" Intelligence Information Processing and Trusted Computing (IPTC), 2010 International Symposium on pp 320-323

[Yue'2009] Daheng Vue, Van Sun, Minxuan Zhang, Shaoqing Li, and Yutong Dai **"A Look-Up-Table Based Differential Logic to counteract DPA attacks"** ASIC, 2009. ASICON '09. IEEE 8th International Conference on pp 855-858

[Zen'2012] Juanli Zeng, Yi Wang, Cheng Xu, Renfa Li1, **"Improvement on Masked S-box Hardware Implementation"** Innovations in Information Technology (IIT), 2012 International Conference on. Pp 113 - 116. 18-20 March 2012

[Zha'2011] Yuyu Zhang, Guoxi Wang, Yufeng Ma, Jingwen Li1, **"A Comprehensive Design Method Based on WDDL and Dynamic Cryptosystem to Resist DPA Attack.pdf"** Intelligence Science and Information Engineering (ISIE), 2011 International Conference on pp.333 - 336