FACULTAD DE FÍSICA

Grado en Física

Trabajo de fin de grado

Procesado de señales eléctricas para la optimización de ataques laterales en circuitos criptográficos

José Manuel Domínguez Begines

Tutores

Antonio José Acosta Jiménez

Erica Tena-Sánchez

Junio de 2016

Agradecimientos

A los tutores de este trabajo: Antonio Acosta Jiménez y Erica Tena-Sánchez.

Por elaborar los scripts que hacen posible los ataques, a Irene Durán Menor de Gaspar (que también tomó las trazas de Trivium) y nuevamente a Erica (que es además responsable de la ejecución de dichos ataques).

Este trabajo se ha realizado en el ámbito de los proyectos 201450E034 y 201550E039 del CSIC y TEC2013-45523-R y TEC2016-80549-R del Ministerio de Economía, Industria y Competitividad, realizados en el Instituto de Microelectrónica de Sevilla, como complemento a las tareas de investigación llevado por el equipo de investigación del grupo TIC180, al que pertenecen los tutores.

Resumen

Existen diversas formas de romper la seguridad de un sistema criptográfico. Una de ellas son los ataques de canal lateral o alternativo, que se aprovechan de la posibilidad de medir, durante el proceso de cifrado, las magnitudes físicas relacionadas con el dispositivo que implementa el algoritmo criptográfico.

Una de las magnitudes físicas que puede aprovecharse son las trazas del consumo de potencia eléctrica del dispositivo durante el proceso de cifrado. La adquisición de dichas trazas, junto con el conocimiento de los datos a crifrar, permite ejecutar ataques de análisis diferencial de potencia, que a través de la dependencia del consumo de potencia del dispositivo con los bits de la clave secreta, permiten que ésta sea revelada al completo.

La capacidad de un atacante para llevar a cabo este tipo de ataques puede verse limitada por, entre otros factores, el poder computacional necesario para ejecutarlos, o también por la presencia de una gran cantidad de ruido en las trazas de consumo de potencia medidas.

En este trabajo, se prueban potenciales soluciones a los dos problemas. La carga computacional de los ataques se va a reducir mediante técnicas simples para eliminar la información redundante de las trazas de consumo, con resultados prometedores. El ruido en la señal se ha intentado reducir mediante un procesado digital de la señal de consumo, aunque los resultados no han sido favorables.

Índice

1. Introducción

Tradicionalmente, la ciberseguridad se ha basado en la **"seguridad matemática"** de los algoritmos criptográficos. Para proteger la información mediante criptografía se crea un algoritmo que actúa sobre la información a proteger (texto plano) y sobre una **clave secreta** y produce un texto cifrado [1]. El objetivo, es que las transformaciones efectuadas por el algoritmo hagan del descifrado de los datos un problema matemático lo suficientemente complejo como para requerir una cantidad de poder computacional tan grande que haga inviable su resolución. El algoritmo normalmente es conocido públicamente, así que toda su seguridad depende de la protección de dicha **clave secreta**. Entonces, los algoritmos deben ser diseñados para que esta clave no pueda ser descubierta a partir del conocimiento de los textos planos o los textos cifrados. Para intentar romper la "seguridad matemática", existen, a grandes rasgos, dos opciones.

Por un lado se puede realizar un **ataque de fuerza bruta**, en el que simplemente se prueban todas las posibles claves que puede utilizar el algoritmo criptográfico, hasta que se obtiene un resultado positivo con alguna. El problema de este método es, que para un dispositivo que funciona con una clave de n bits, existen 2^n posibles claves distintas que se pueden configurar en el mismo. Con el poder de computación que proporciona la tecnología actual, se considera que una clave de 128 bits es suficiente como para que este ataque sea completamente inútil, ya que se podrían tardar millones de años en obtener la clave.

Otra opción es encontrar una solución al problema matemático que requiera un poder computaciónal mucho menor (**una vulnerabilidad**). Si el algoritmo no es suficientemente complejo, o utiliza números aleatorios y se ha descuidado su generación, puede ser una tarea no extremadamente compleja.

Sin embargo, a medida que han ido evolucionado los algoritmos, se ha ido haciendo cada vez más complicado atacar los sistemas criptográficos de esta manera, así que se han ideado métodos alternativos de ataque, que no están basados en violar la "seguridad matemática" del algoritmo, sino en aprovecharse del hecho de que el algoritmo debe ser ejecutado por un dispositivo. Este dispositivo es naturalmente un sistema físico, del que se puede extraer información durante su funcionamiento normal utilizando instrumentos de medida. Este tipo de ataque se conoce como **"ataque de canal lateral o ataque de canal alternativo"**, y puede explotar diferentes tipos de magnitudes físicas medibles, como el tiempo que tarda el algoritmo en completar su ejecución [2], la emisión electromagnética [3], o el consumo de potencia eléctrica del dispositivo [4]. Este tipo de ataques puede clasificarse en ataques invasivos/no invasivos y activos/pasivos [5].

Los **ataques invasivos** involucran modificar el dispositivo o desmontarlo para obtener información de canales que no son las interfaces de entrada/salida que se incluyeron en el dispositivo para interactuar con el exterior. Los **no invasivos**, utilizan exclusivamente la información provista por estas interfaces.

En los **ataques pasivos** se intenta obtener la clave secreta haciendo funcionar al dispositivo en condiciones normales, mientras que en un **ataque activo** se manipulan

de forma específica las entradas, la fuente de corriente o cualquier otro factor externo al mismo, pero que pueda influir en su funcionamiento y hacer que manifieste información relevante para el ataque.

Aunque **este trabajo no trata sobre como llevar a cabo este tipo de ataques**, va a proporcionar una metodología relevante a la hora de optimizar los ataques existentes, ya que el trabajo está orientado a procesar la información obtenida durante estos ataques para hacerlos más fructíferos. En este trabajo nos centramos en un ataque en concreto, el llamado **ataque de "análisis diferencial de potencia"** (en inglés, "differential power analysis attack" o DPA, como podremos referirnos al mismo a partir de ahora). Este ataque está clasificado como un ataque pasivo no invasivo ya que explota la información obtenida del consumo de potencia del dispositivo atacado durante la encriptación. Este tipo de ataques es uno de los más potentes que se han presentado hasta ahora debido a que el atacante no necesita información precisa del dispostivo a atacar y el equipamiento utilizado para el ataque es común y accesible, por lo que supone una gran amenaza para la seguridad. Su funcionamiento está descrito de forma básica en la sección 2.2.

Como ya se ha dicho, la ventaja de este vector de ataque es que puede resultar más sencillo romper la seguridad del dispositivo de esta manera que por fuerza bruta o buscando vulnerabilidades en el algoritmo. Aun así, el ataque aún puede requerir de un poder computacional considerable, debido a la gran cantidad de datos que hay que almacenar y procesar. Otro problema que puede surgir, es que el consumo de potencia no contenga información lo suficientemente clara como para llevar a cabo el ataque con éxito.

Por ello, en este trabajo se estudian diferentes **técnicas de procesado** que se pueden aplicar a una señal de consumo de potencia de un dispositivo criptográfico para intentar abordar estos dos problemas. De aquí en adelante, se llamará "traza" a una señal de este tipo.

El primer problema (reducir la necesidad computacional), se intenta aliviar mediante técnicas de **compresión de la señal**, es decir, eliminando información que puede ser redundante para el ataque. En el segundo caso, se intenta mejorar la razón señal/ruido mediante el **filtrado** de una fuente de ruido que suele estar presente en cualquier dispositivo criptográfico: la **señal de reloj del circuito criptográfico**.

2. Objetivos y metodología

Uno de los objetivos de este trabajo es, aplicar técnicas de compresión a una señal de consumo de potencia de un dispositivo criptográfico para eliminar la información redundante que pueda contener, tratando de permitir así la realización de un ataque con menos recursos computacionales. Para ello, el Instituto de Microelectrónica de Sevilla (IMSE), ha proporcionado trazas de consumo de potencia de dos dispositivos. En un caso, son trazas obtenidas por simulación eléctrica de la implementación de un circuito S-Box [6], constitutivo de numerosos algoritmos criptográficos. El otro es un dispositivo físico implementado sobre una plataforma FPGA (Field Programmable Gate Array) que ejecuta el algoritmo AES [7].

El segundo objetivo es, conseguir obtener la clave de un dispositivo ASIC (Application Specific Integrated Circuit) que ejecuta el algoritmo criptográfico Trivium, que se ha intentado atacar numerosas veces en el IMSE sin resultados favorables [8], debido al alto nivel de ruido contenido en sus trazas. En consecuencia se va a intentar filtrar el posible ruido introducido por la señal de reloj del dispostivo en la traza de consumo de potencia. Al igual que antes, las trazas de consumo de potencia son proporcionadas por el IMSE

La S-Box es un bloque relativamente simple, constitutivo de gran cantidad de algoritmos criptográficos implementados como cifradores de bloque [7], [9],[10]. Sirve normalmente como bloque básico para montajes mas complejos. Simplemente es un bloque que para cada palabra de entrada, genera una palabra de salida diferente, conforme a una función combinacional. Normalmente la función que de salida depende no linealmente de la entrada, para proteger la información lo máximo posible. Una serie de bits de entrada pertenecen a la clave y otros a un texto plano. En la implementación utilizada en este trabajo, la S-Box emplea entradas y claves de 9 bits, de modo que a veces se le llama "S-Box". En la figura 6, aparece un ejemplo de traza de consumo de potencia de la S-Box

El segundo dispositivo implementa el algoritmo AES (Advanced Encryption Standard), implementado en una FPGA, que es extremadamente popular, siendo por ejemplo el empleado para el cifrado en sistemas Windows, Android o iOS en la electrónica de consumo. En la figura 7, aparece un ejemplo de traza de consumo de potencia para el dispositivo que ejecuta AES.

Por último, está el dispositivo ASIC que ejecuta el algoritmo Trivium. Trivium es, de acuerdo con sus creadores [11], un cifrador de flujo que pretende ofrecer un equilibro flexible entre velocidad y superficie ocupada. En consecuencia, es ideal si se quiere implementar un cifrador de flujo en un dispositivo pequeño. Más adelante (figura 10), se muestra un fragmento de una traza de consumo de potencia de este dispositivo.

2.1. Flujo de trabajo

Para poder llevar a cabo el procesado de las trazas de consumo de potencia y analizar los resultados de los ataques, se ha seguido el siguiente flujo de trabajo:

1. Adquisición las trazas de consumo de potencia de los dispositivos.
2. Aplicación las técnicas de procesado pertinentes.
3. Realización del ataque de análisis diferencial de potencia con las trazas procesadas.
4. Análisis de resultados.

Sólo el segundo y el cuarto paso son objeto de estudio del trabajo. El resto ha sido realizado por personal del Instituto de Microelectrónica de Sevilla (IMSE). En consecuencia, el IMSE, tras la realizacióin de cada ataque, ha devuelto una cantidad limitada de

información sobre los mismos. Los datos sobre el resultado del ataque que se devuelven desde el IMSE aparecen en la sección 2.2.

Utilizando ese conjunto de datos, se puede obtener una serie de conclusiones sobre la efectividad de los ataques. Un ataque efectivo implicará que el procesado aplicado a la traza de consumo de potencia cumple su cometido. Si por el contrario, el ataque no tiene éxito, con los datos recibidos se puede intentar encontrar una explicación de por qué no ha funcionado. En la sección 2.2, se explica también como se realizan los ataques.

2.2. Ataque de análisis diferencial de potencia

Como ya se ha comentado, el tipo de ataque que se emplea en este trabajo para encontrar la clave de los dispositivos es el análisis diferencial de potencia. Puede encontrarse una descripción muy detallada de su funcionamiento en [5]. Aquí sólo se extrae de dicha fuente una descripción básica del proceso, para que puedan entenderse los resultados de los mismos. En la figura 1, se muestra el montaje experimental requerido para llevar a cabo estos ataques.

Este tipo de ataque analiza el consumo de potencia en un instante de tiempo fijo de los muestreados en la traza. Este consumo es dependiente de los datos (clave y texto plano) que procesa el dispositivo, y por tanto revela información sobre ambos. Las diferencias de consumo entre los distintos valores de la clave y del texto plano son normalmente pequeñas, como se muestra en las figuras 2 y 3. Esto lleva a que normalmente sea necesario realizar muchas medidas, para contrarrestar los efectos del ruido que pueda tener la señal, cuya magnitud puede ser comparable a la magnitud de las diferencias de consumo de potencia entre las diferentes medidas.

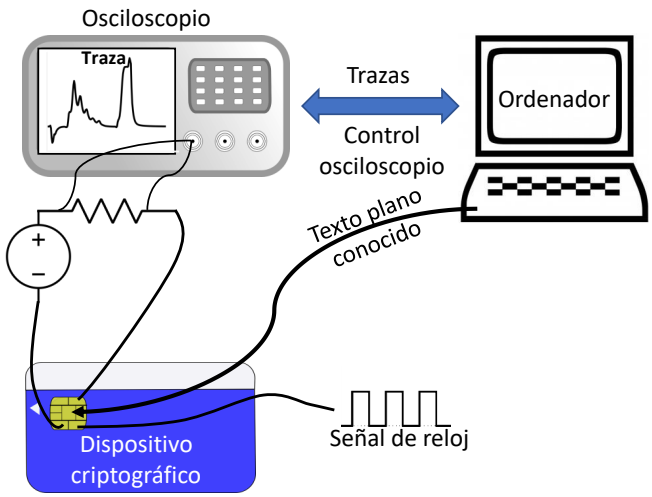


Figura 1: Esquema del montaje experimental necesario para la adquisición de trazas. Un osciloscopio es controlado mediante un ordenador, que recibe las trazas de consumo de potencia medidas. Para adquirir las trazas, uno de los métodos posibles es medir la caída de potencial en una pequeña resistencia conectada en serie con la fuente del dispositivo criptográfico [5, (sec 3.4)].

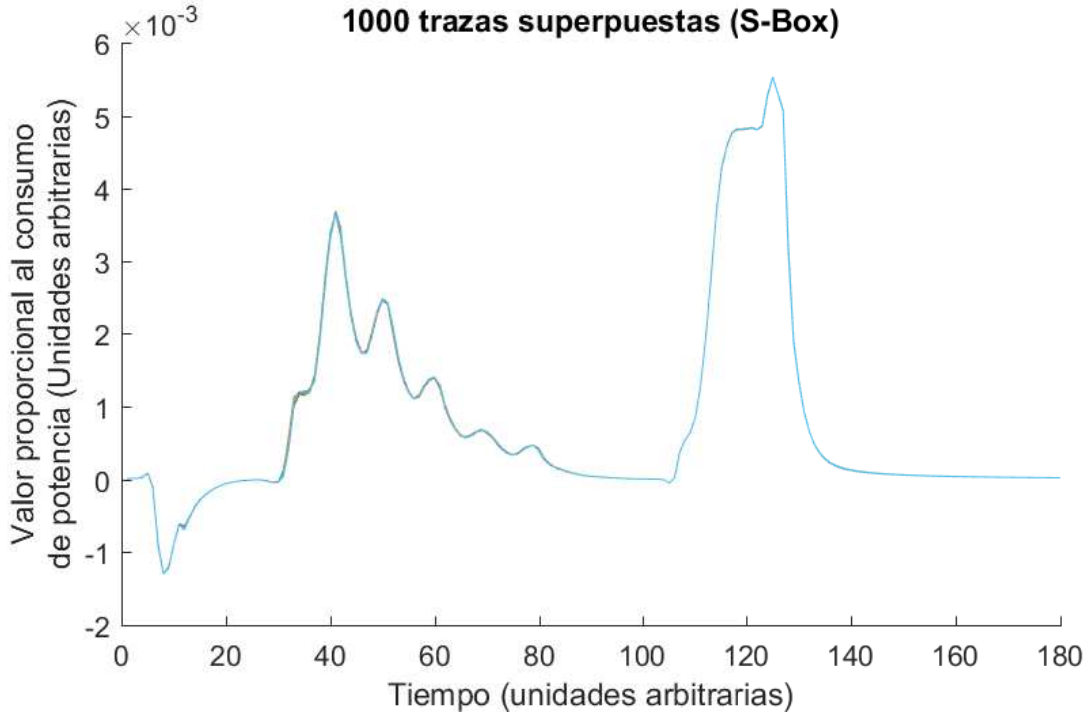


Figura 2: Conjunto de 1000 transiciones superpuestas pertenecientes a una misma clave para uno de los dispositivos estudiados. El tiempo y la potencia se muestran en unidades arbitrarias porque dichas unidades son irrelevantes para el estudio. Con esta escala no se observa prácticamente ninguna diferencia entre las transiciones.

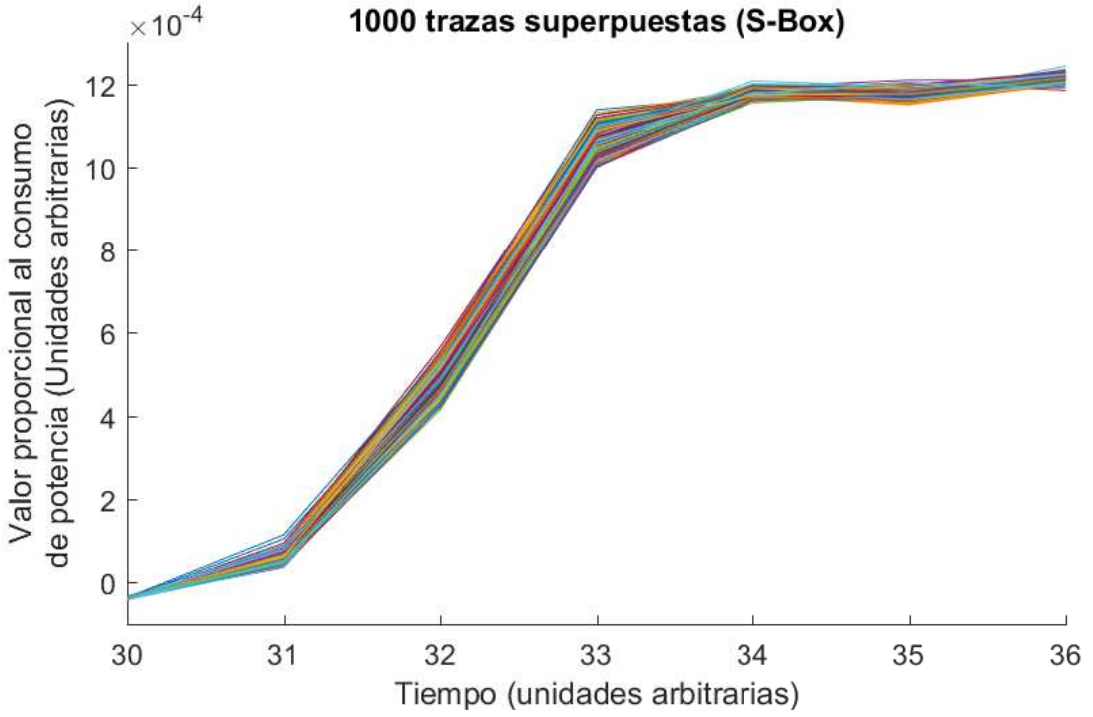


Figura 3: Conjunto de 1000 transiciones superpuestas pertenecientes a una misma clave para uno de los dispositivos estudiados. El tiempo y la potencia se muestran en unidades arbitrarias porque dichas unidades son irrelevantes para el estudio. Esta escala permite apreciar perfectamente la dependencia del consumo de potencia con los datos procesados, ya que se ven diferentes patrones.

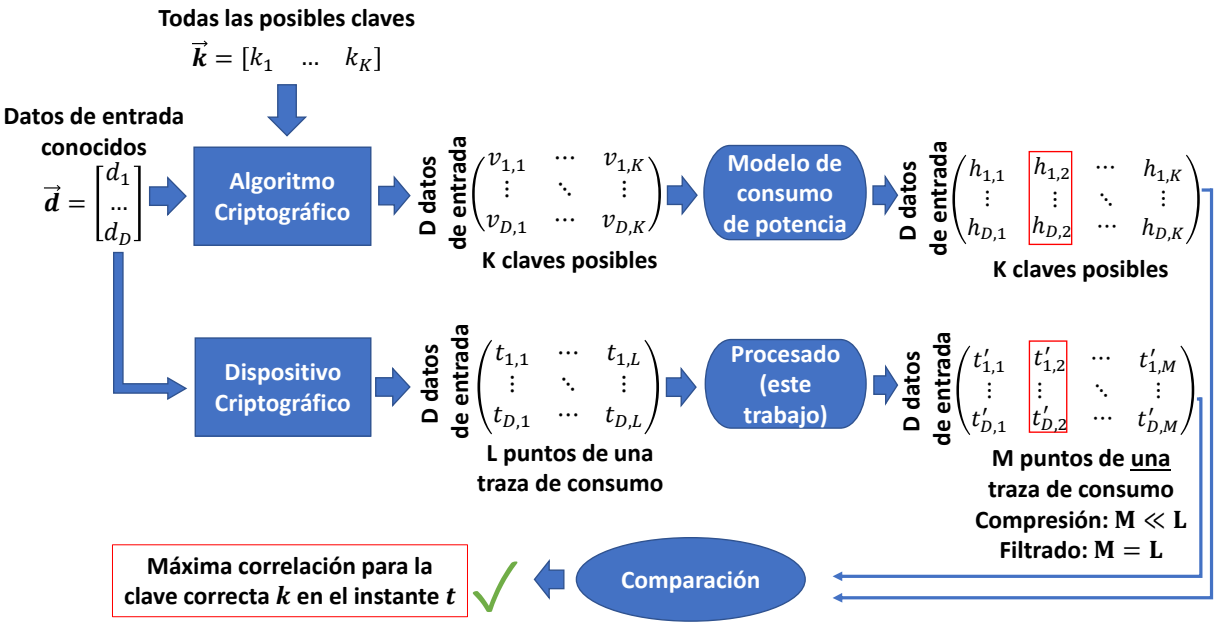


Figura 4: Esquema de los elementos que intervienen en el ataque y cómo están interrelacionados.

Para realizar este tipo de ataque, es fundamental tener en mano el dispositivo que va a atacarse, ya que se necesita hacer que cifre una serie de textos planos conocidos que impone el atacante (para separar la dependencia del consumo en clave y texto plano, al cuál **nos vamos a referir a partir de ahora como "datos"**) y medir el consumo de potencia del mismo. Además, hace falta conocimiento sobre el algoritmo criptográfico que se está ejecutando, aunque por lo general no es necesario tener detalles profundos sobre el dispositivo que lo implementa, ya que este conocimiento tan sólo se emplea para realizar un modelo estimativo del consumo de potencia del dispositivo, que puede ser suficiente para realizar el ataque.

De acuerdo con [5, (sec 6.1)], podría describirse el proceso de ataque en los cinco pasos que se explican a continuación. En la figura 4, se muestra un esquema explicativo del proceso de ataque.

1. **Seleccionar un valor intermedio del algoritmo criptográfico.** Durante la ejecución del algoritmo, tanto los datos como la clave son procesados y se generan valores intermedios a partir de ellos, por lo que hay que seleccionar un valor intermedio que dependa tanto de la clave como de los datos, que puede representarse por una función $f(d, k)$, donde d representa un valor de los datos, que van cambiando, y k un fragmento de la clave.

2. **Medir el consumo de potencia.** A continuación, se mide el consumo de potencia del dispositivo mientras cifra D conjuntos de datos (d_i) distintos. El valor intermedio elegido anteriormente debe procesar algún dato del conjunto de datos d_i considerado. Cada subíndice i, corresponde a un conjunto de datos distinto, y por tanto a un proceso de cifrado distinto. El conjunto de todos los valores de datos empleados puede recogerse en un vector columna $\vec{d} = [d_1, ..., d_D]^T$. En cada proceso de cifrado, se mide una traza de consumo de potencia, que puede recogerse en un vector fila

$\vec{\boldsymbol{t_i}} = [t_{i,1}, ..., t_{i,L}]$, donde L es el número de instantes de tiempo medidos en la traza. Al final, las trazas de consumo pueden recogerse en una matriz $\boldsymbol{T}$, de tramaño $D \,\mathrm{x}\, L$, donde cada fila está asociada a un proceso de cifrado con un conjunto de datos d_i determinado. En este trabajo, además se procesan las trazas medidas, transformando la matriz $\boldsymbol{T}$ en una matriz $\boldsymbol{T'}$, con un nuevo número de columnas, M. Si el procesado es una compresión de trazas, $M \ll L$. Si es un filtrado de la señal de reloj, entonces $M = L$.

3. **Cálculo del valor intermedio.** Conociendo el algoritmo criptográfico, se puede calcular el valor intermedio escogido anteriormente para todas las claves posibles que puedan configurarse en el dispositivo. Estas claves pueden agruparse en un vector fila de claves (con longitud K, donde K es el número total de claves posibles) $\vec{\boldsymbol{k}} = [k_1, ..., k_K]$. Una vez se tienen los vectores $\vec{\boldsymbol{d}}$ y $\vec{\boldsymbol{k}}$, se puede calcular el valor intermedio escogido $f(d_i, k_j)$ para todos los elementos en $\vec{\boldsymbol{d}}$ y en $\vec{\boldsymbol{k}}$, es decir, para $i = 1, ..., D$ y $j = 1, ..., K$. Como resultado se obtiene una matriz de tamaño $D \,\mathrm{x}\, K$, que llamaremos $\boldsymbol{V}$, cuyos elementos vienen dados por $v_{i,j} = f(d_i, k_j)$.

   Entonces, las columnas de esta matriz representan los valores intermedios para la clave k_j y las filas, los valores intermedios correspondientes a los datos d_i para todas las claves.

4. **Asignación de valores de consumo de potencia a los valores intermedios.** Una vez se tienen los valores intermedios, sólo queda hacerles corresponder a cada uno un valor de consumo de potencia. Aquí es donde entra en juego el modelo de consumo de potencia mencionado anteriormente, que es necesario elaborar previamente. Cuanto más exacto sea este modelo, mejores resultados puede dar el ataque.

   En resumen, lo que se hace es calcular a partir de la matriz $\boldsymbol{V}$ de valores intermedios, otra matriz $\boldsymbol{H}$ de valores de consumo de potencia dados por el modelo.

5. **Comparación de los valores de consumo de potencia del modelo con las trazas medidas en el laboratorio.** Ahora se compara cada columna de la matriz $\boldsymbol{H}$, que representa el consumo de potencia para una clave concreta y para el valor intermedio escogido (para todos los conjuntos de datos), con cada columna de la matriz $\boldsymbol{T'}$, que representa el consumo de potencia (procesado) para un instante de tiempo concreto (en el que no tiene por qué procesarse el valor intermedio escogido), para todos los conjuntos de datos. De la comparación se obtiene una matriz $\boldsymbol{R}$ de tamaño $K \,\mathrm{x}\, M$, que contiene unos coeficientes $r_{i,j}$, que indican cómo de relacionadas están las columnas de las matrices $\boldsymbol{H}$ y $\boldsymbol{T'}$. Cada fila de la matriz $\boldsymbol{R}$ representa una clave y cada columna un instante de tiempo concreto en las trazas de consumo procesadas.

   En los ataques sobre los sistemas que conciernen a este trabajo, los coeficientes $r_{i,j}$ son coeficientes de correlación. En consecuencia, cuanto mayor sea el valor absoluto de $r_{i,j}$, mayor es la correlación del consumo de potencia medido en el instante $t_j^{'}$ con el consumo de potencia dado por el modelo para ese mismo instante. Muchos de los elementos de una fila de $\boldsymbol{R}$ tendrán valores muy parecidos, porque no en todos los instantes de tiempo se procesa el valor intermedio seleccionado. Además, si esa fila no corresponde a la clave configurada en el dispositivo, los valores de $r_{i,j}$ serán pequeños en general. Sin embargo, si el ataque tiene éxito, en alguna fila de la matriz tiene que aparecer un valor notablemente mayor (en valor absoluto) del coeficiente

$r_{i,j}$ que en el resto de filas. **La fila en que aparece este valor indica la clave que se ha configurado en el dispositivo**. También es posible que aparezcan varios valores notablemente mayores si el valor intermedio se procesa varias veces.

Una vez realizado el ataque, son devueltas una serie de informaciones sobre los resultados, a saber:

Éxito o fracaso del ataque. Si se ha acertado o no la clave configurada en el dispositivo.

Clave configurada en el dispositivo. Es la que se debe obtener al atacar el dispositivo si el ataque funciona correctamente.

Clave devuelta por el ataque. Es la clave que el ataque ha dado como resultado. Puede corresponderse o no corresponderse con la clave configurada en el dispositivo (en este último caso se dice que el ataque ha fallado).

Máximo valor de correlación. Máximo (en valor absoluto) de los valores de los coeficientes de correlación en la matriz de correlación ($\boldsymbol{R}$) durante la realización del ataque. Este valor es una medida directa de lo aprovechable que es la información de consumo de potencia de la que se dispone. Si valiese 1, teóricamente se obtendría la clave correcta directamente. Si es menor que uno, seleccionar como solución la clave a la que corresponde este valor de correlación no garantiza obtener la clave configurada en el dispositivo. Aun así, naturalmente cuanto más se acerque a uno mayor será la posibilidad de que la clave relacionada con este valor sea la configurada en el dispositivo.

Diferencia del valor máximo de la correlación con segundo valor máximo de la correlación. Si se obtienen el valor máximo de correlación y el segundo valor máximo (en valor absoluto) de la matriz de correlación, y se calcula su diferencia, se obtiene este valor. Este valor mide, cómo de seguro es que la clave correcta se corresponda con la clave relacionada con el máximo valor de la correlación y no con el siguiente. Si esta diferencia es grande, al elegir la clave asociada al máximo valor habrá menos riesgo de equivocarse. Si la diferencia es pequeña, hay un riesgo mayor de que la clave correcta sea la relacionada con el segundo valor máximo de correlación.

Posición en el ranking. Al realizar el ataque, en base a estos valores de correlación, se puede elaborar un "Ranking" de posibles claves correctas. La primera del ranking sería la que el ataque devuelve, es decir la que el ataque considera correcta (aquella asociada con el valor máximo de la correlación). La segunda, la que según el ataque se debería probar si falla la primera (aquella con el segundo valor máximo de la correlación), y así sucesivamente. Evidentemente, el objetivo del ataque es que la primera clave del ranking se corresponda con la clave configurada en el dispositivo. Si esto no es posible, al menos interesa que esté entre las primeras posiciones.

MTD. Measurements to disclose the key, o el número de patrones o datos diferentes a cifrar mínimo necesario para que el ataque sea fructífero. Es una medida de lo vulnerable o segura que es una implementación frente a un ataque. Mientras mayor sea, más seguro es el dispositivo. Mientras menor sea, más vulnerable, porque se requiere menos esfuerzo computacional para obtener la clave [12].

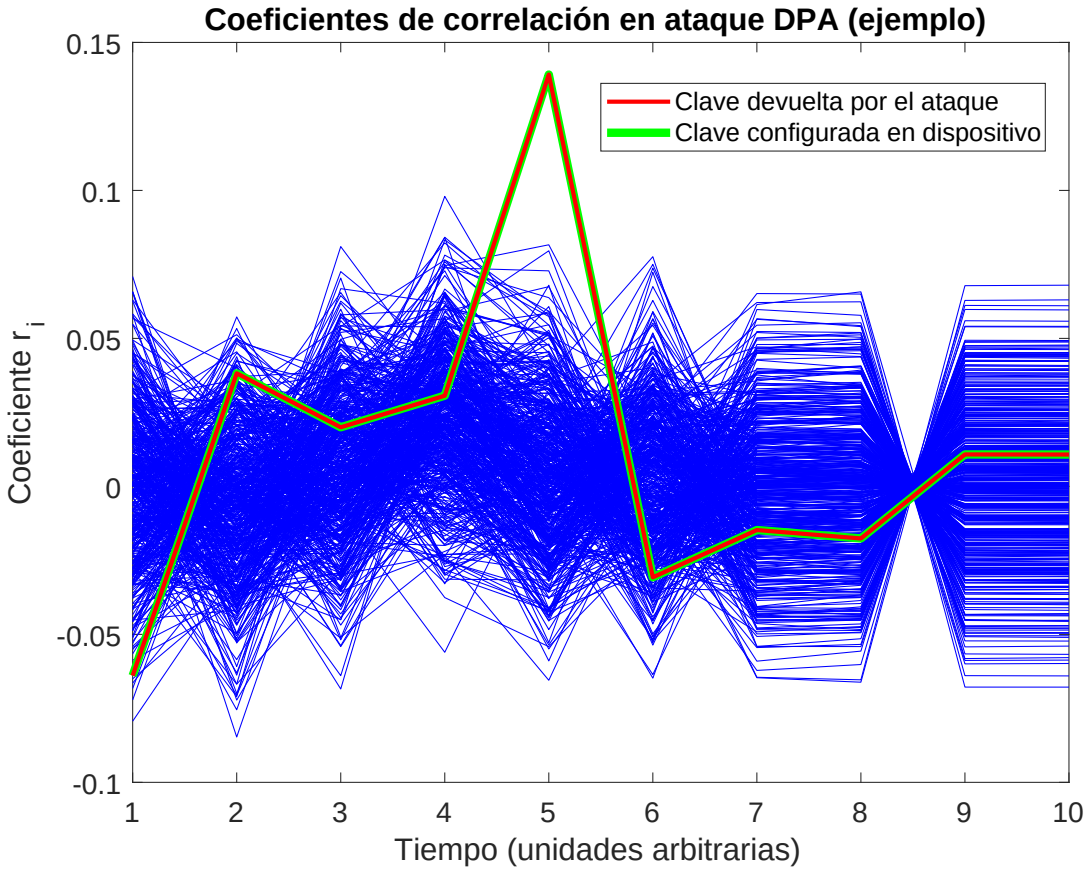


Figura 5: Resumen de un ataque exitoso al dispositivo S-Box utilizando una traza comprimida. En el eje horizontal se representan los puntos de ataque, mientras que en el vertical se muestra simultáneamente el valor del coeficiente r para todas posibles claves. La línea roja representa la clave que el ataque ha considerado correcta y la verde la configurada en el dispositivo. Como el ataque ha sido exitoso en este caso, ambas coinciden. Nótese cómo la clave que ha elegido el ataque es aquella que presenta picos de correlación que la hacen sobresalir sobre la representación para el resto de claves.

A partir de los resultados del ataque, se obtienen unas representaciones gráficas que son comunes en la literatura. En dichas gráficas, se refiere el eje horizontal a los instante de tiempo medidos t_j, mientras que el vertical se refiere a la correlación $r_{i,j}$ obtenida para dichos instantes de tiempo para la clave k_i. A continuación se representan estos datos para todas las claves, superpuestos unos sobre otros. Esto permite determinar en un vistazo rápido, en primer lugar, si el ataque ha tenido éxito, ya que entonces debe aparecer un pico de gran magnitud que sobresale sobre el "continuo" de valores $r_{i,j}$ similares. En segundo lugar, se puede saber además en qué punto (o puntos si hay más de un pico) de la traza se procesa el valor intermedio elegido. En la figura 5 puede verse un ejemplo de este tipo de gráfica para un ataque exitoso en el que se ha empleado una de las trazas comprimidas producidas en este trabajo.

Para el análisis de los resultados del ataque realizado con las trazas comprimidas, no se necesitan realmente este tipo de representaciones, pero para el análisis de los resultados del ataque realizado con las trazas filtradas sobre Trivium, sí que van a resultar útiles.

2.3. Compresión de trazas

Una traza contiene información sobre el consumo de potencia del dispositivo en un lapso de tiempo determinado. Este período puede comprender varios ciclos de reloj (por ejemplo, el cifrado de diferentes textos planos), o sólo el tiempo necesario para procesar un texto plano. Cada ciclo de reloj perteneciente a una traza de consumo se conoce como transición. En el caso de la S-Box, el tiempo necesario para procesar un texto plano es de un ciclo de reloj, mientras que en el caso de AES es de diez ciclos de reloj.

La información que contiene la transición corresponde a todos los instantes de tiempos muestreados que la componen. Dado que en el ataque sólo interesan aquellos instantes en los que se procesan determinados valores intermedios que se han seleccionado previamente, por lo que la parte de la transición que no corresponde a dichos valores intermedios es prescindible y su inclusión en el ataque no proporciona ninguna información útil.

Hasta la realización de este trabajo, al realizar los ataques, estos instantes de tiempo redundantes también se estaban considerando, incrementando el tamaño de las matrices que son procesadas y en consecuencia aumentando también tanto el esfuerzo computacional necesario para completar el ataque, así como el espacio ocupado por las trazas en memoria. La respuesta a este problema es eliminar la información redundante de las trazas tratando de dañar lo mínimo posible la información aprovechable contenida en las mismas.

Naturalmente, la forma más eficaz de hacer esto sería identificar qué puntos son inútiles y qué puntos contienen la información relevante, y a continuación eliminar los primeros. Pero no siempre se tiene el conocimiento necesario para identificar estos puntos. En ese caso, habría que usar un método "ciego" (o al menos parcialmente ciego), para comprimir las trazas. Tres de estos métodos se proponen en [5, (sec 4.5.2)], pero no se estudian en dicha publicación. Esos tres métodos son los que se tratan en este trabajo.

Cualquiera de estos tres métodos de compresión considerados actúa como una función, que recibe como entrada el consumo de potencia del dispositivo en todos los instantes de tiempo medidos y produce como salida un escalar, o visto de otra forma, a efectos de la realización del ataque, una "nueva traza" que consta solamente de un único "instante de tiempo".

A continuación, se enumeran los tres métodos estudiados, que van acompañados de un símbolo entre paréntesis (...). A partir de ahora podrá utilizarse indistintamente o bien su símbolo o bien su nombre para hacerles referencia.

Suma simple ($\sum x_i$) Consiste en sumar todos los valores del consumo de potencia presentes en la traza.

Suma de valores absolutos ($\sum |x_i|$) Igual que la suma simple pero en este caso se suma el valor absoluto del consumo de potencia en cada instante de tiempo.

Suma de cuadrados ($\sum x_i^2$) Se suma el cuadrado de todos los valores de consumo de potencia.

Aunque originalmente en [5], se considera la integración de los valores en lugar de su suma, debido a la forma en que funciona el ataque, no importa si se hace una cosa o la otra, ya que como la traza es un conjunto de puntos discretos, al integrar se obtiene un valor proporcional al que se obtiene al sumar, y ambos son válidos para la realización de los ataques.

Además, no se ha limitado el estudio a simplemente aplicar estos métodos. Ante la sospecha de que estos métodos puedan ser demasiado drásticos, y eliminen no sólo la información redundante, sino también gran parte de la necesaria para encontrar la clave correcta, se ha optado por implementar también otras soluciones intermedias empleando técnicas de "bracketing".

El bracketing consiste en dividir cada traza en varios intervalos de instantes de tiempo, y aplicar cada uno de los tres métodos anteriores a cada uno de los intervalos por separado. En los casos estudiados, se ha utilizado bracketing de dos, cinco y diez intervalos de tiempo. En este caso, no se obtiene como resultado un escalar, sino que, si por ejemplo se divide la traza en cinco intervalos, se obtiene una "nueva traza" compuesta por cinco escalares o "instantes de tiempo".

Para estudiar la efectividad de los métodos, se ha considerado una implementación simulada de una S-Box (substitution box, o "caja de sustitución") y una implementación en un dispositivo físico del popular algoritmo AES, que ya han sido descritas en detalle en la sección 2.

En ambos casos, se ha medido el consumo de potencia (en el primero más bien se ha simulado), y se han realizado ataques DPA para los tres métodos considerados, además de para los varios tipos de bracketing distintos (dos, cinco y diez intervalos). La selección del punto de la traza donde comienzan y acaban dichos intervalos se ha realizado en principio de manera que a cada intervalo le corresponda el mismo número de puntos (a partes iguales), con la excepción de un caso, el de la S-Box, en el que para el bracketing de dos intervalos se han elegido los intervalos de instantes de tiempo [1,100] y [101,180], porque en dicho dispositivo se conoce que el primero corresponde a la fase de precarga y el segundo a la de evaluación. Este conocimiento previo de dónde puede filtrarse información podría ayudar a mejorar los resultados.

Naturalmente, no se han sumado a mano los valores para los cientos de trazas de las que se disponen, sino que se ha automatizado el proceso mediante MATLAB (ver código 1 en la sección 7). En las figuras 8 y 9, pueden verse ejemplos de nuevas trazas producidas al utilizar el método de la suma simple para S-Box y AES respectivamente. Los otros dos métodos producen resultados visualmente parecidos. Si se desea observar la diferencia entre los métodos, en las conclusiones generales sobre la compresión de trazas (sección 3.3), aparece una comparación de los tres métodos de compresión con un bracketing de 10 intervalos para S-Box y AES (figuras 29 y 30).

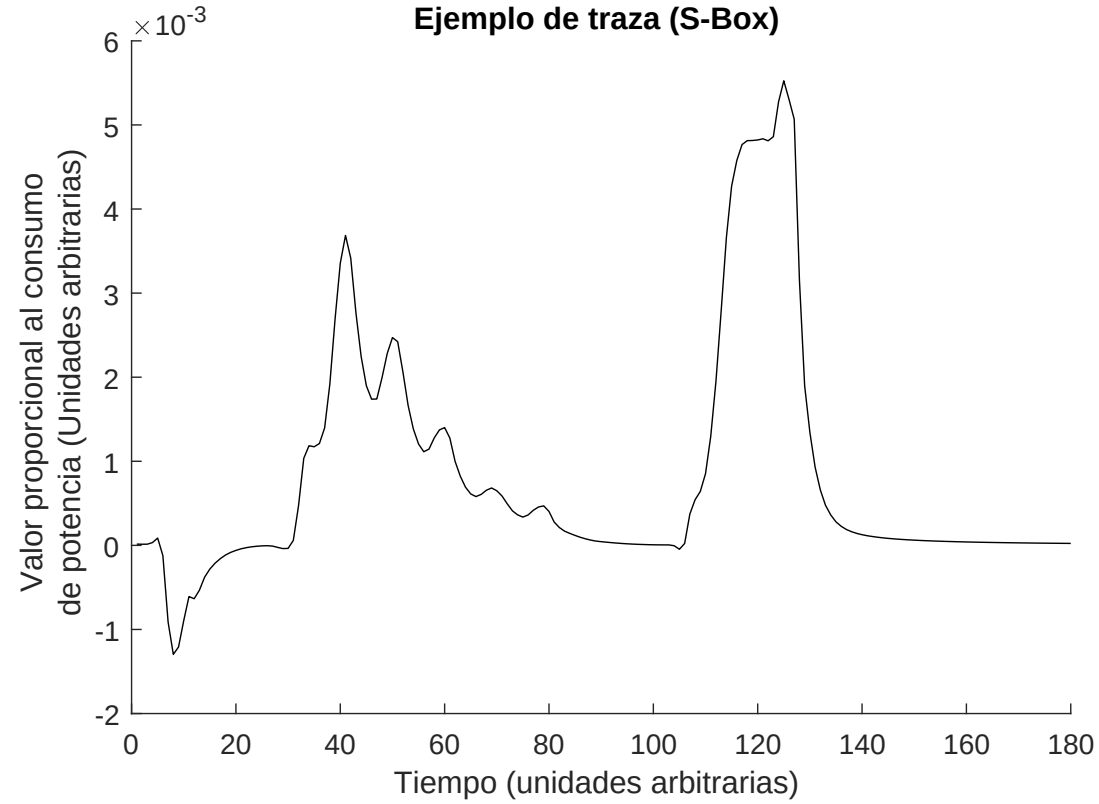


Figura 6: Traza de ejemplo para S-Box. El tiempo y la potencia se muestran en unidades arbitrarias porque dichas unidades son irrelevantes para el estudio.

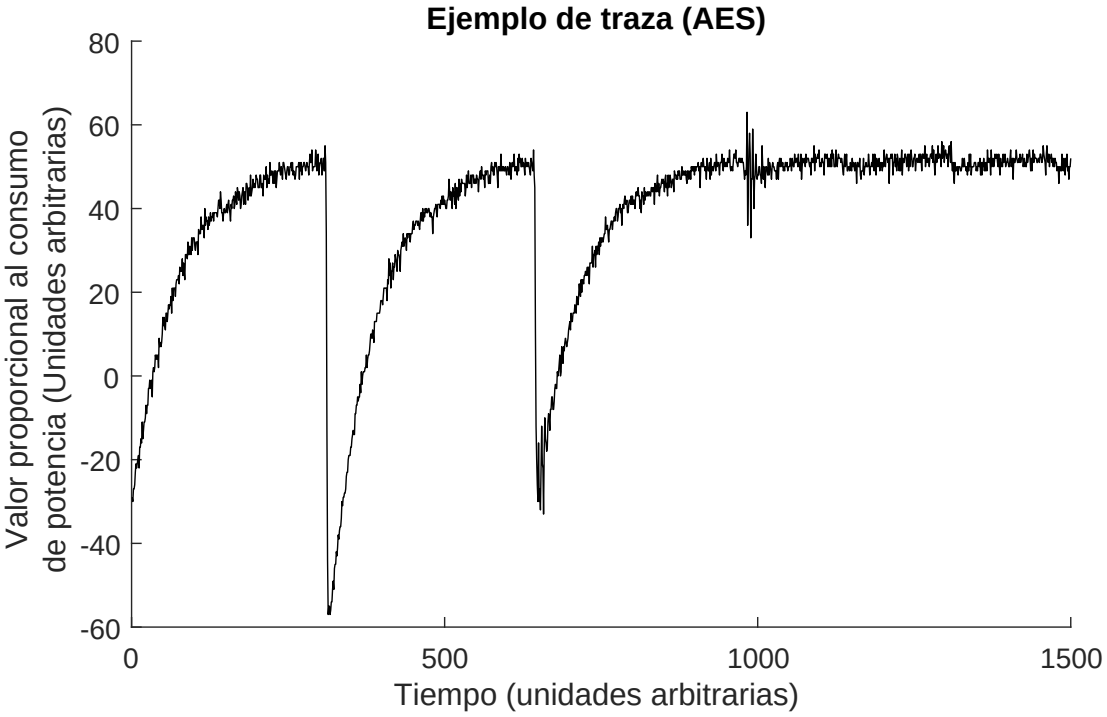


Figura 7: Traza de ejemplo para AES, donde se observan dos picos correspondientes a las dos últimas rondas de encriptación. El tiempo y la potencia se muestran en unidades arbitrarias porque dichas unidades son irrelevantes para el estudio.

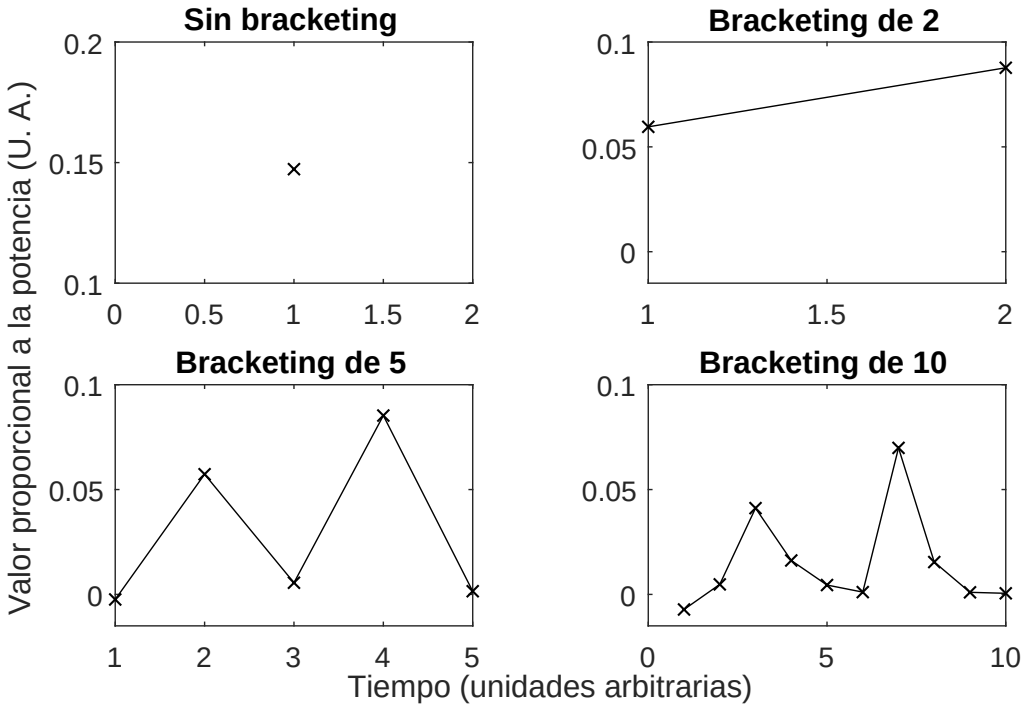


Figura 8: Resultados de aplicar la técnica de suma simple a una traza ejemplo de S-Box. Se observa cómo a medida que aumenta el número de intervalos de bracketing, la señal comprimida se va pareciendo a la original (figura 6) cada vez más.

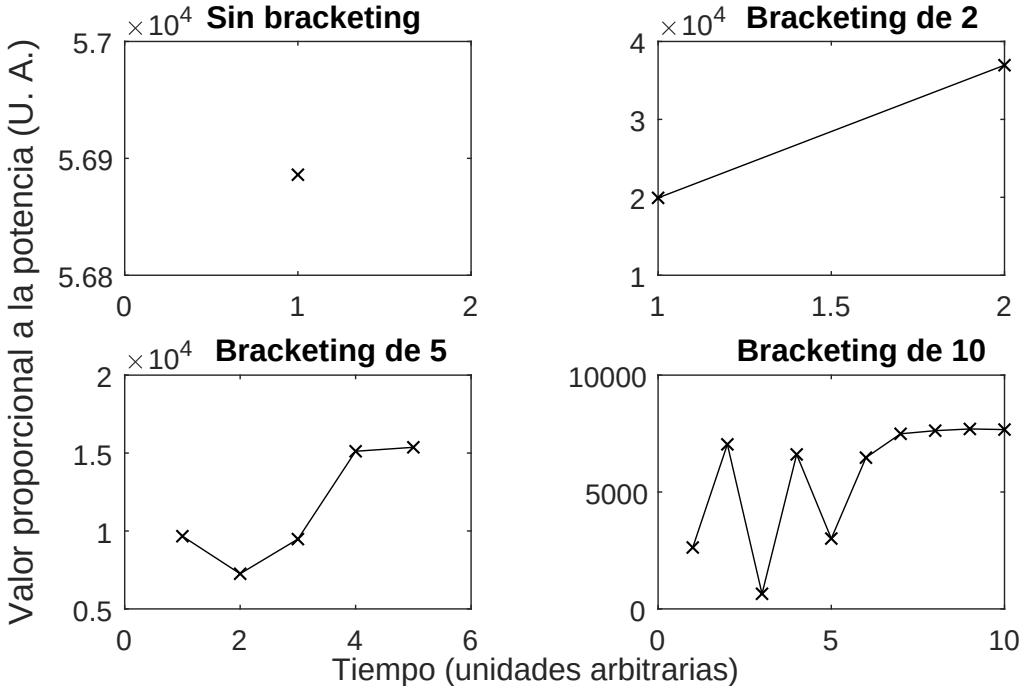


Figura 9: Resultados de aplicar la técnica de suma simple a una traza ejemplo de AES. Se observa cómo a medida que aumenta el número de intervalos de bracketing, la señal comprimida se va pareciendo a la original (figura 7) cada vez más.

2.4. Filtrado de la señal de reloj

Con el fin de eliminar información no aprovechable, se ha procedido a eliminar el efecto de la señal de reloj en la traza de consumo. Dicho efecto no contiene información útil, por lo que es interesante eliminar su influencia.

Para el filtrado de la señal de reloj de las trazas de Trivium, se ha decidido utilizar una transformada de Fourier que se ha realizado mediante el algoritmo Fast Fourier Transform. En concreto se ha utilizado la versión de dicho algoritmo implementada en el software MATLAB. La justificación de esta decisión, es el carácter periódico de dicha señal, y el hecho de que en una traza aparece suficientes veces (96), como para que aplicar la transformada de Fourier en una señal de duración finita pueda dar un resultado aceptable.

La señal de reloj del dispositivo es de frecuencia conocida, concretamente 2MHz. Como es una señal de tipo pulso cuadrado, hay que tener en cuenta que esta señal no se compone únicamente de un seno a la frecuencia de 2MHz, sino que tendrá aportación de información en un gran número de frecuencias, cuanto más cuadrada se genere la señal, más componentes de alta frecuencia tendrá. Así que no basta con hacer la transformada de Fourier y filtrar la componente de 2MHz, si no que es necesario tener en cuenta que van a aparecer también armónicos superiores en aquellas frecuencias que sean múltiplos enteros de la misma. El filtrado ha sido automatizado mediante un script de MATLAB (ver código 2 en la sección 7), de forma que las 10000 trazas con 120001 valores muestreados de las que se disponía se procesan en alrededor de 20 minutos[1]. A continuación, se muestra paso por paso cómo se ha realizado dicho filtrado.

Se parte de la señal original (figura 10). En la figura se puede observar tanto un fragmento de dicha señal como una señal sinusoidal con la misma frecuencia que la señal de reloj, sobre la que se realizarán las mismas operaciones que se realizan sobre la señal a filtrar. El fin de esta señal sinusoidal es ser usada como referencia gráfica de la frecuencia de reloj.

Si ahora se realiza una transformada de Fourier discreta mediante el algoritmo FFT integrado en MATLAB, se obtiene lo mostrado en la figura 11. Se ve que hay una componente con una intensidad notable en una frecuencia que coincide exactamente con la frecuencia de referencia (frecuencia de reloj). Pero como ya se ha comentado, esta no es la única componente de la que hay que preocuparse, sino que también pueden aparecer armónicos superiores de esta frecuencia. Para comprobarlo, se modifica la señal de referencia por una que es la suma de sinusoides cuyas frecuencias son la frecuencia de reloj y sus múltiplos enteros.

Una vez hecho esto, ahora se puede ver la localización de todos esos armónicos no deseados en el dominio de la frecuencia. En la figura 12, se muestra una versión ampliada de la figura 11 en torno a la frecuencia de reloj, pero que además incluye las posiciones de estos armónicos.

Muchos de estos armónicos también tienen una intensidad apreciable. Se ha observado

[1]Equipo: Corei7 720QM, 4GB RAM

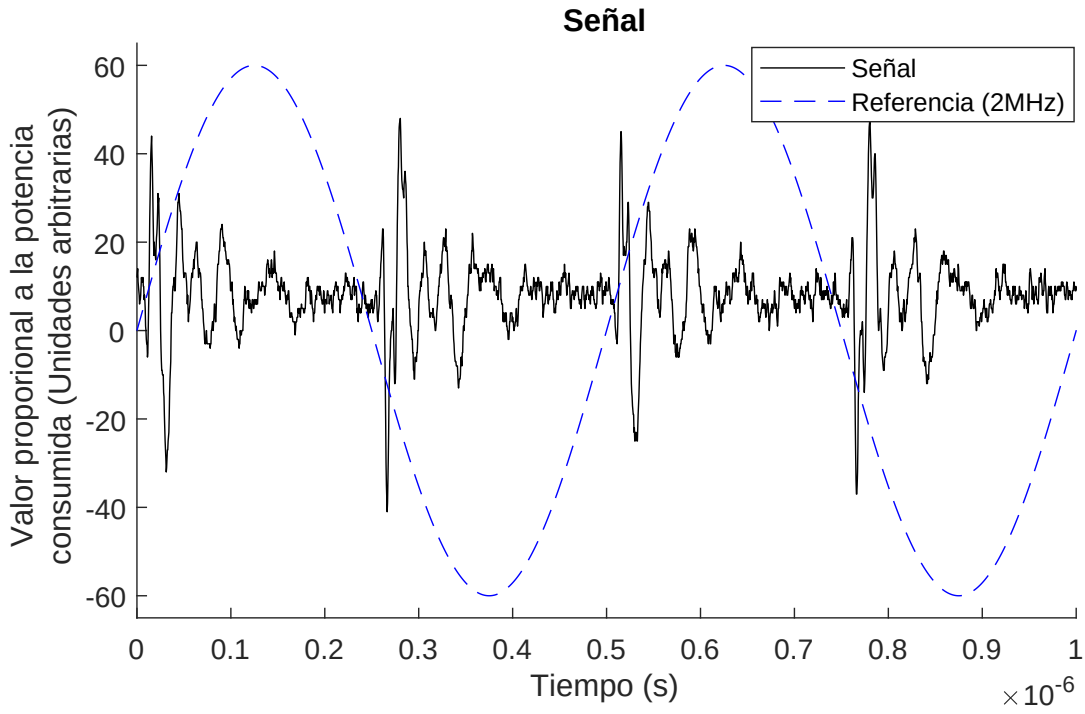


Figura 10: Fragmento de la señal original extraída del dispositivo, junto con una señal sinusoidal de referencia con la misma frecuencia que la señal de reloj.

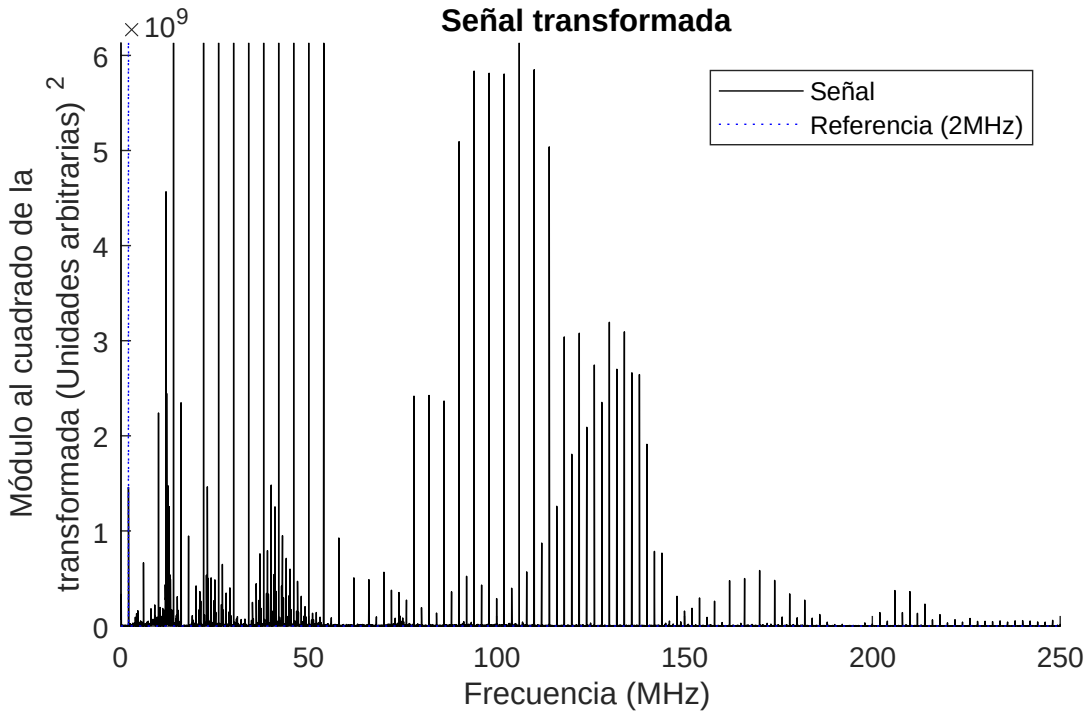


Figura 11: Señal original (sin filtrar) en el dominio de la frecuencia (cuadrado del módulo de la transformada), junto con la transformada de la señal sinusoidal de referencia. Se han omitido las frecuencias para las cuales el cuadrado del módulo de la transformada es casi nula, además de la parte simétrica de la transformada.

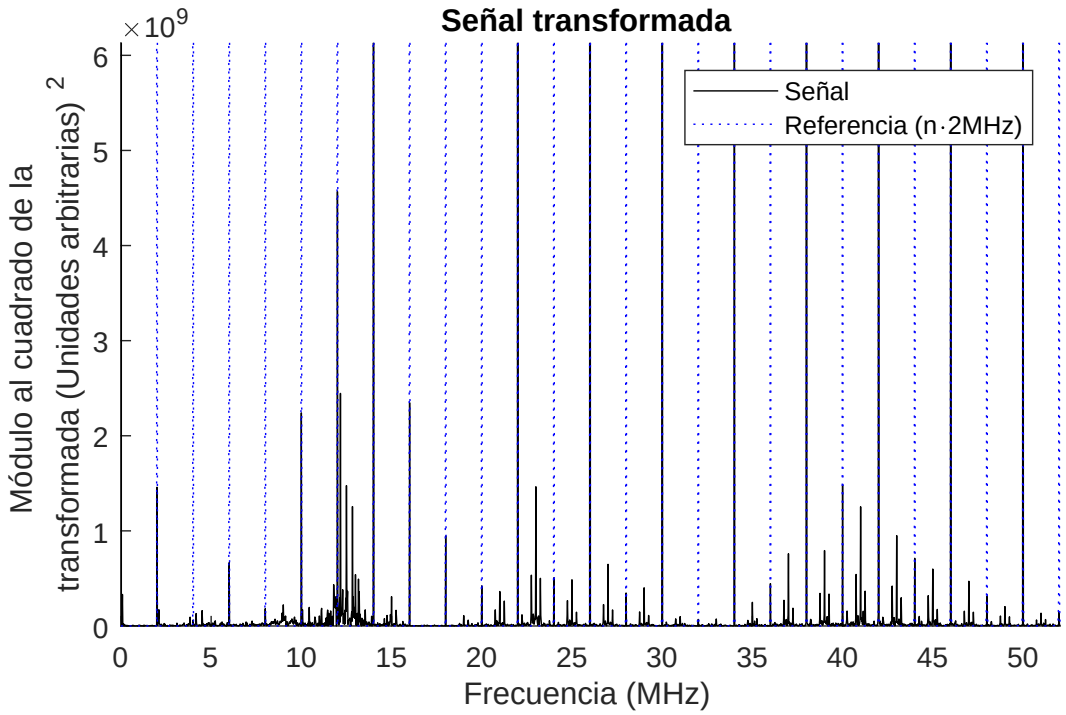


Figura 12: Señal original (sin filtrar) en el dominio de la frecuencia (cuadrado del módulo de la transformada), junto con la transformada de la señal sinusoidal de referencia y sus armónicos superiores. Se han omitido las frecuencias para las cuales el cuadrado del módulo de la transformada es casi nula, además de la parte simétrica de la transformada.

(no se muestra en la figura) que aún pueden distinguirse las intensidades de los armónicos frente al resto de componentes que les rodean hasta la frecuencia del armónico número 150, que corresponde a una frecuencia de 300MHz (aunque como ya se ha visto en la figura 11, realmente empiezan a ser despreciables mucho antes).

Sabiendo esto, hay que escoger un método para filtrar las componentes de la transformada de Fourier asociadas con las frecuencias correspondientes. Como observando las figuras queda claro que la señal de reloj y sus armónicos se manifiestan de forma similar a deltas de Dirac (picos de gran altura), como método de filtrado se ha empleado simplemente la sustitución de la componente de la transformada correspondiente por el valor cero.

En todas las figuras que se muestran en lo que queda de esta sección, se muestra el resultado de filtrar únicamente 10 armónicos de la señal de reloj (es decir, la frecuencia de 2MHz Y 9 armónicos de frecuencia superior). Sin embargo, **se ha realizado también el filtrado eliminando 35, 75 y 150 armónicos**, números que, como se observa en la figura 11, son armónicos que están ubicados en frecuencias que separan partes notablemente distintas del espectro (o en el caso de los 150 armónicos, el fin de la manifestación apreciable de los armónicos). Si se observasen diferencias en los resultados con cada tipo de filtrado, probablemente esta selección las maximize.

En la figura 13 se puede ver el resultado de la operación, filtrando **10 armónicos**. Como en esta figura los cambios pueden no ser suficientemente evidentes, se ha añadido

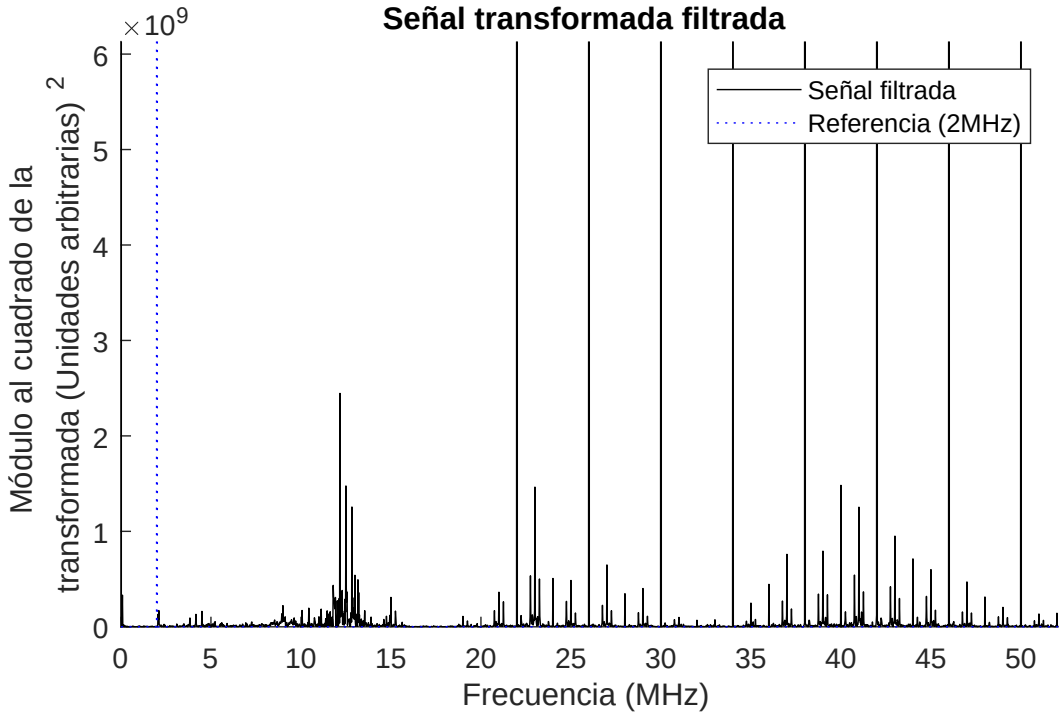


Figura 13: Señal filtrada en el dominio de la frecuencia, junto con la señal de referencia. Nótese como a partir de 20 MHz (armónico 10) no se ha modificado la señal (para ello comparar con la figura 12).

otra figura (figura 14) en la que se muestra el resultado de substraer a la señal original la señal filtrada. Como se puede ver, algunas de las componentes importantes de la señal estaban localizadas en los lugares donde se esperan encontrar los armónicos de la señal de reloj.

Después del filtrado, realizando la transformada de Fourier inversa de la señal filtrada (en el dominio de la frecuencia), se obtiene la señal filtrada en el dominio del tiempo, que tiene un aspecto tal como el que se muestra en la figura 15. En la figura 16, se muestra la diferencia (el resultado de substraer a la señal original la filtrada) entre las dos señales en el dominio del tiempo.

Una vez se tiene la señal filtrada en el dominio del tiempo, se puede ejecutar el ataque utilizando esta señal en lugar de la original. Para consultar los resultados obtenidos, véase la sección 4.

Si se utiliza el código 2 para llevar a cabo el procesado, el fichero obtenido con las trazas filtradas tiene un tamaño aproximadamente 10 veces superior al original. Esto hace que el tiempo de ataque se incremente, pasando de 1 hora y 20 minutos a aproximadamente 114 horas. El motivo de ello es que las trazas originales sólo contienen valores proporcionales al consumo de potencia que son números enteros, mientras que al aplicar este filtrado aparecen cifras decimales en el fichero. Redondear estas cifras permitiría obtener un tamaño de fichero similar al original, y conseguir que el ataque fuese igual de rápido que en el caso original. La contrapartida es que se perdería información que podría ser necesaria para averiguar la clave. En consecuencia, en este trabajo se ha optado por realizar el ataque

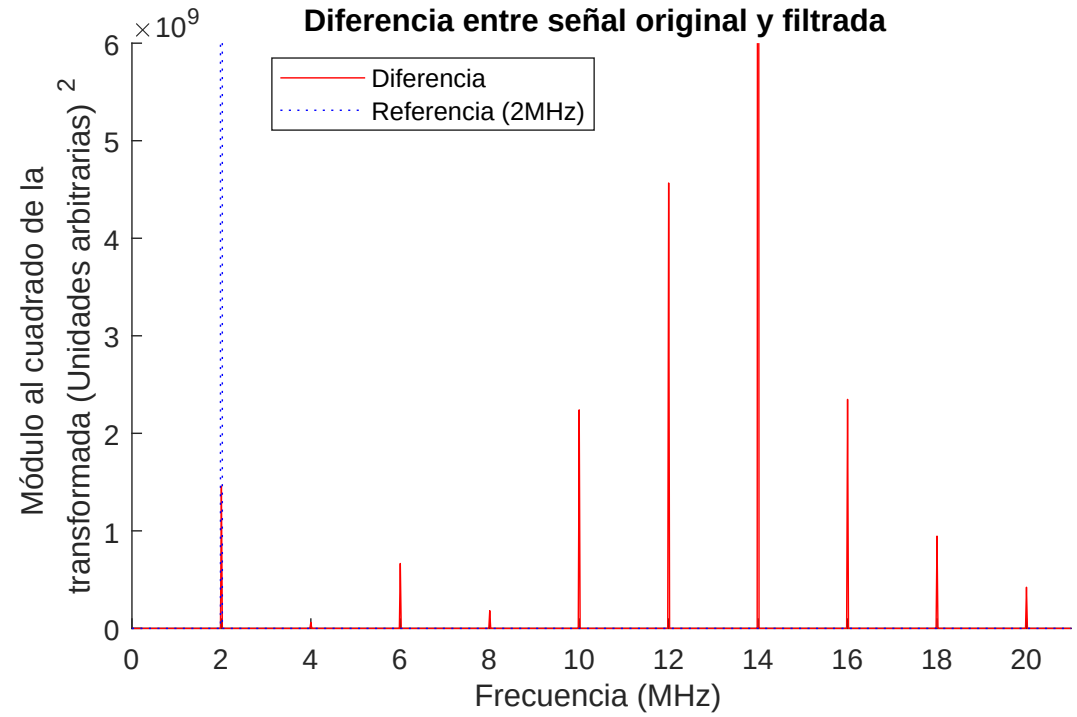


Figura 14: Resultado de substraer a la señal original la señal filtrada (filtrado de 10 armónicos) en el dominio de la frecuencia. Se muestran frecuencias sólo hasta el límite de frecuencia del filtrado (20MHz).

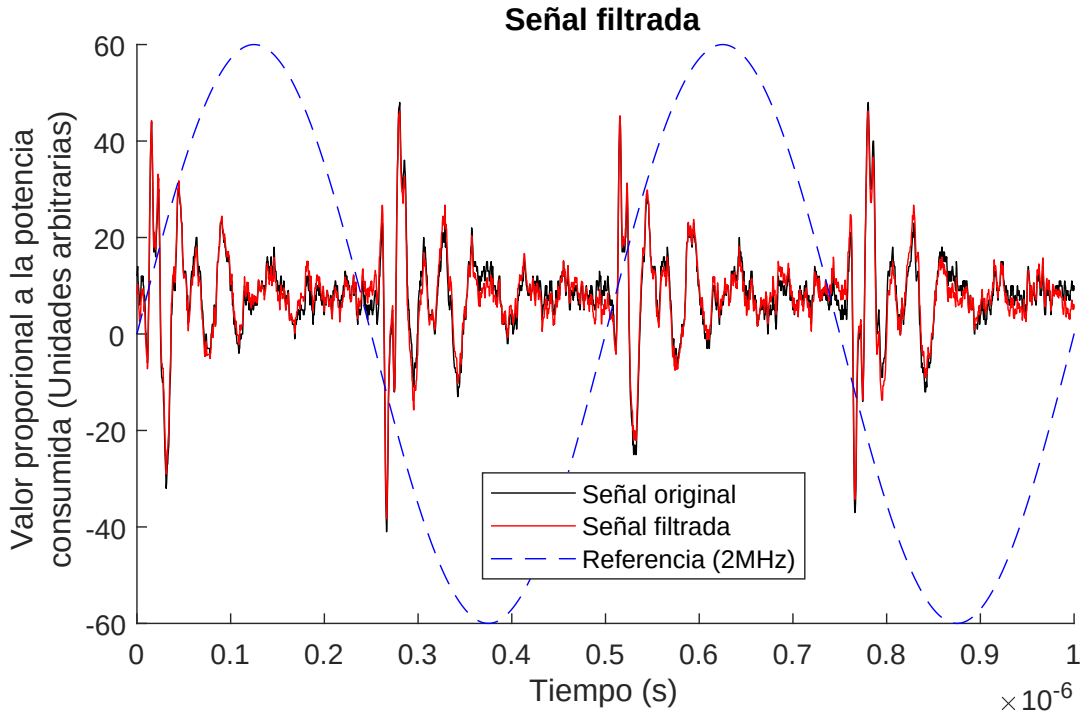


Figura 15: Fragmento de la señal filtrada (filtrado de 10 armónicos) en el dominio del tiempo junto con la señal original y la señal sinusoidal de referencia de 2 MHz.

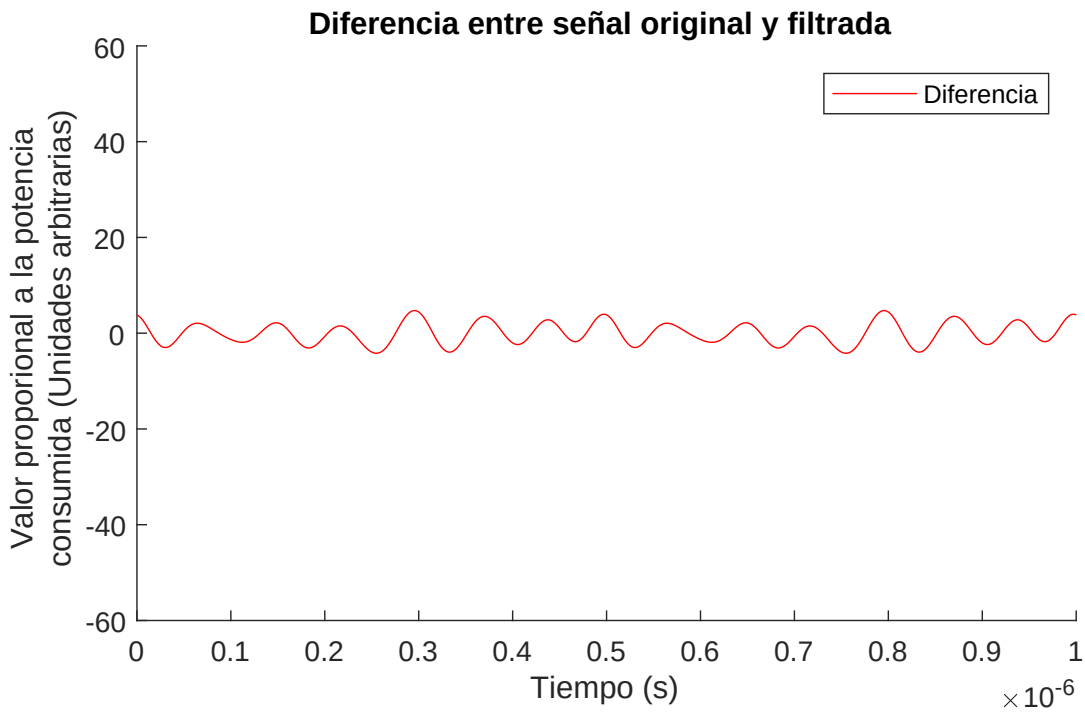


Figura 16: Resultado de substraer a la señal sin filtrar la señal filtrada (filtrado de 10 armónicos) en el dominio del tiempo.

conservando las cifras decimales.

3. Compresión de trazas

3.1. Compresión de trazas de S-Box

Para el dispositivo S-Box, se han obtenido trazas para 10 claves de cifrado distintas. Para cada clave se han utilizado los tres métodos, aplicados sobre la traza completa (sin bracketing), y también con bracketing, dividiendo la traza en dos, cinco y diez intervalos de instantes de tiempo. Nótese que aplicar el método sobre la traza completa (sin bracketing), es equivalente a decir que se ha aplicado bracketing dividiendo la traza en un único intervalo.

Si se tienen en cuenta todas formas de compresión, en total se obtienen 120 formas de comprimir las trazas (10 claves x 3 métodos x 4 tipos de bracketing). De esas 120 formas, 40 de ellas pertenecen al método de suma simple, 40 al método de suma de valores absolutos y otras 40 al método de suma de cuadrados.

En primer lugar, cabe preguntarse, cuál de los tres métodos puede ser el más exitoso, considerando éxito que el ataque devuelva como resultado la clave correcta. Si se cuenta en cuántos experimentos de los 40 que corresponden a cada método de compresión se ha logrado obtener la clave correcta, se obtiene:

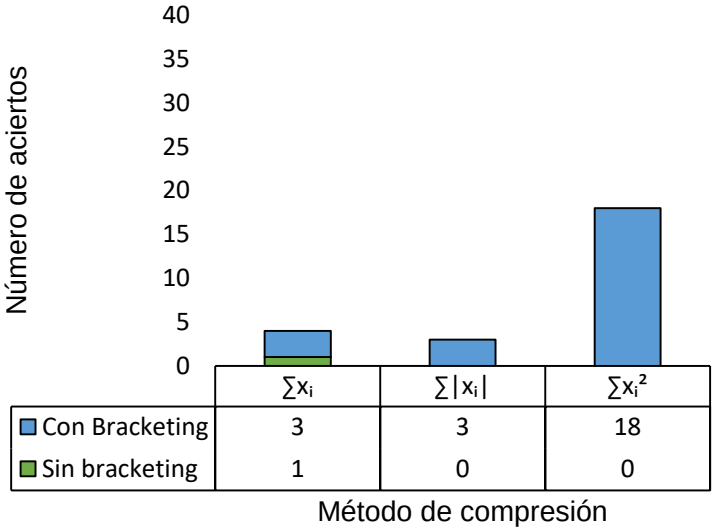


Figura 17: Número de aciertos en función del método utilizado y de si se ha empleado bracketing o no. Recuérdese que 40 es el número máximo de aciertos para este experimento (que coincide con el número de pruebas).

- Suma simple: 4 de 40 (10 % de las veces).
- Suma de valores absolutos: 3 de 40 (7,5 % de las veces).
- Suma de cuadrados: 18 de 40 (45 % de las veces).

Está claro que el método que consigue mayor éxito es el de la suma de cuadrados. Los métodos de suma simple y suma de valores absolutos fracasan casi en la totalidad de los casos.

El siguiente paso es ver, dentro de cada uno de los tres métodos, que es más provechoso, si aplicarlo sobre toda la traza o dividirla en distintos intervalos de instantes de tiempo (bracketing). Esta vez contaremos directamente la cantidad de aciertos en porcentaje. La figura 17 aporta de forma gráfica dicha información.

Un vistazo a dicha figura muestra que la abrumadora mayoría de los aciertos se consiguen cuando se emplea la estrategia de bracketing. De hecho, tan sólo se consigue encontrar la clave correcta cuando no se emplea bracketing una sola vez. En las otras 24 ocasiones se ha empleado bracketing.

A la vista de este resultado se puede concluir, que aplicar cualquiera de los tres métodos sin emplear bracketing es bastante inútil. El bracketing es un requisito indispensable. De todas formas, debe tenerse también en cuenta que el bracketing funciona, pero que incluso con el método más favorable (suma de cuadrados), aproximadamente un tercio de los intentos de ataque fracasan (recordar que de los 40 intentos por método, 10 se realizan sin bracketing). Hay algo en la técnica de compresión de las trazas que todavía falla.

Para intentar esclarecer esta cuestión, se separaron los datos anteriores por estrategia de bracketing aplicada (2, 5 o 10 intervalos). En la figura 18, se “desdoblan” los resultados

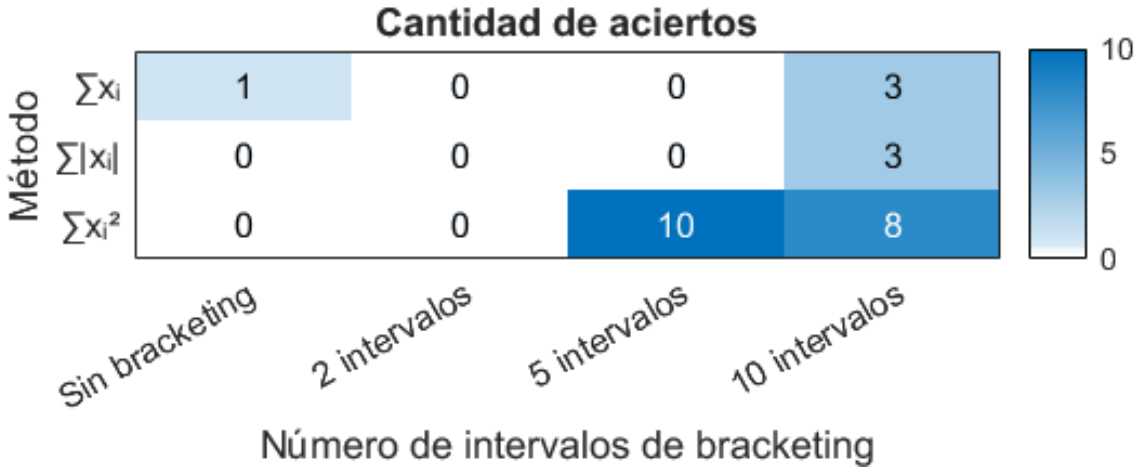


Figura 18: Aciertos según el método de compresión aplicado y el número de intervalos de bracketing elegidos. Las zonas más oscuras representan mayor número de aciertos, y las más claras menos aciertos. En este caso el valor máximo de aciertos es 10. Nótese que al desplazarse hacia la derecha (aumenta el número de intervalos) y hacia abajo (hacia el método de suma de cuadrados) aumenta la eficacia en general.

de la figura 17. Ahora no sólo se distingue entre aplicar o no aplicar bracketing, si no que además aparecen las tres estrategias.

En dicha figura se puede ver, atendiendo a la dirección horizontal (relacionada con el número de intervalos de bracketing), que la cantidad de aciertos conseguida aumenta notablemente al aumentar el número de intervalos de bracketing. Para el bracketing de 10 intervalos hay en total bastantes más aciertos que para el bracketing de cinco, y con el bracketing de dos intervalos no se consigue acertar la clave nunca.

En segundo lugar, atendiendo a la dirección vertical, relacionada con el método de compresión aplicado, se observa que los métodos de suma simple y suma de valores absolutos tienen un desempeño bastante similar en todos los casos (algo que ya se ha puesto de manifiesto anteriormente en la figura 17). Sin embargo, al examinar los datos para el método de suma de cuadrados, se encuentra nueva información: hace falta un número mínimo de intervalos para poder encontrar la clave (aparentemente mayor que dos para este dispositivo), a partir del cuál la relación entre los aciertos conseguidos y los intentos realizados (10 en cada caso), es muy elevada.

Entonces, si se quiere aprovechar la compresión de trazas (al menos para esta S-Box), no sólo hay que usar el método de suma de cuadrados si no que además hay que emplear un número de intervalos de bracketing suficiente. Recordemos que de momento, al menos para el dispositivo actual (S-Box), es mayor que dos.

Aunque ya se ha obtenido una información bastante útil acerca de cada forma de comprimir las trazas, todavía se puede intentar obtener más información. Para ello se pueden considerar todos los datos adicionales que devuelve el ataque, que se recuerdan aquí (para una descripción más detallada consultar la sección 2.2):

- Clave configurada en el dispositivo

- Clave devuelta por el ataque
- Máximo valor de correlación
- Diferencia entre valor máximo de correlación y segundo valor máximo de correlación
- Posición en el ranking
- MTD

Analizando estos datos adicionales quizás se consiga explicar mejor por qué ocurre esto, y además explorar la causa de algunos fenómenos contraintuitivos a la vista de los datos, como por ejemplo que la suma simple sin bracketing consiga dar una clave correcta, o que para el método de suma de cuadrados se obtengan más aciertos para el bracketing de 5 intervalos que para el bracketing de 10 intervalos.

Antes de comenzar, hay una cosa bastante importante que considerar. Los resultados no son los mismos ni se obtiene la misma tendencia para cada clave, aunque se fije el mismo método de bracketing y el mismo método de compresión. Quizás bien porque en función de los bits de las claves se obtiene más información o menos en el consumo de potencia del ataque, o por alguna peculiaridad en la forma en que se procesen los datos para el ataque.

En consecuencia, no es útil controlar el bracketing y el método de compresión e ir cambiando la clave. En su lugar lo que se ha hecho es, para cada conjunto de bracketing y método, obtener un valor medio de las magnitudes para todas las claves, y además una desviación típica de ese valor medio. El valor medio indica qué esperar en general al aplicar el método a una clave desconocida, y la desviación típica, en qué medida los resultados del ataque van a depender de la clave configurada en el dispositivo. Esta desviación típica aparece en forma de barras de error o acompañada del símbolo $\pm$.

Comencemos por la información que puede dar el hecho de saber la clave correcta y la clave resultado del ataque, es decir, la información relativa al **número de bits que no se consigue acertar** en el ataque. Se emplea para ello la **distancia de Hamming**, que se define como la suma del número de casos en los que un bit de una de las claves es distinto del bit correspondiente en la otra clave [5, (sec 3.2.2)]. Por ejemplo, en este dispositivo, que funciona con claves de 9 bits, el valor máximo de esta distancia será 9 y ocurrirá cuando una clave sea “111111111” y la otra “000000000”. Esta distancia de Hamming indica entonces la medida en que la clave devuelta por el ataque está cerca de la configurada en el dispositivo.

En la figura 19, se muestra la distancia de Hamming entre la clave configurada en el dispositivo y la obtenida en el ataque en función del número de intervalos de bracketing. Para la suma simple y la suma de valores absolutos se observa una ligera disminución con el aumento de intervalos, aunque como la dependencia con la clave es muy fuerte y el cambio pequeño, tampoco queda muy claro qué ocurriría si siguiésemos aumentando el número de intervalos de bracketing. De momento lo único que se sabe es que siempre está aproximadamente entre 3 y 5. Para la suma de cuadrados, sí que se observa que una vez superado el mínimo de dos intervalos de bracketing, la distancia de Hamming baja

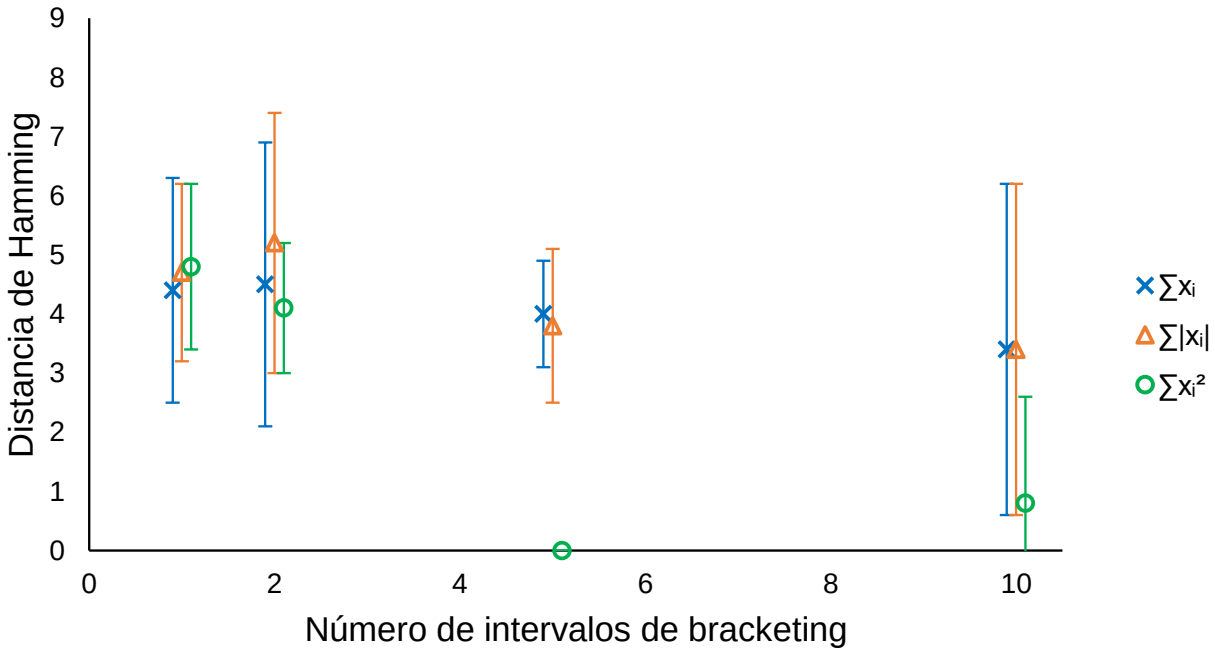


Figura 19: Distancia de Hamming (media) entre la clave configurada en el dispositivo y la devuelta por el ataque, en función del número de intervalos de bracketing, para cada uno de los métodos de compresión. Aunque técnicamente, los valores para cada método deberían aparecer en el mismo lugar del eje horizontal, como los datos son discretos, se han desplazado 0,1 unidades unos de otros, para evitar que se solapen y facilitar su entendimiento.

de un valor entre 4 y 5 a un valor de entre 0 y 1. Nótese que para el bracketing de 5 no aparece ninguna dependencia con la clave porque se consigue acertar en todos los casos (desviación típica cero). En el caso del bracketing de 10, el dato nos indica que pese a la disminución de la distancia de Hamming, la dependencia con la clave se mantiene elevada. Quizás si hubiese fallado algún intento para el bracketing de 5 se hubiese podido observar el mismo comportamiento.

Dado que la distancia de Hamming se relaciona con lo cerca que se está de la clave configurada en el dispositivo, cabe esperar que tenga algún tipo de relación con la **posición en el ranking de la clave dada por el ataque**, que a fin de cuentas es una medida distinta del mismo factor. En la figura 20 se muestran los resultados.

En este caso se obtiene una tendencia no muy clara y con una desviación típica elevada para la suma simple y la suma de valores absolutos, mientras que para la suma de cuadrados se obtiene una tendencia similar a la mostrada por la distancia de Hamming, con la salvedad de que en este caso la medida del nivel de acercamiento a la clave correcta es más fina, ya que además la dependencia con la clave va disminuyendo al aumentar el número de intervalos de bracketing (aunque de nuevo, como todos aciertan para cinco intervalos de bracketing en el caso de la suma de cuadrados, tenemos una anomalía).

El análisis de la siguiente magnitud, el **valor máximo de la correlación con el número de intervalos de bracketing**, sí es capaz de mostrar alguna tendencia en el bracketing de cinco intervalos para la suma de cuadrados. Como se dijo cuándo se presentó esta magnitud, esta va a ser la que potencialmente puede arrojar más luz sobre

el asunto, ya que muestra la utilidad de la información que se ha conservado después de la compresión. Se puede ver la dependencia mencionada en la figura 21.

El valor medio del máximo valor de la correlación sube al aumentar el número de

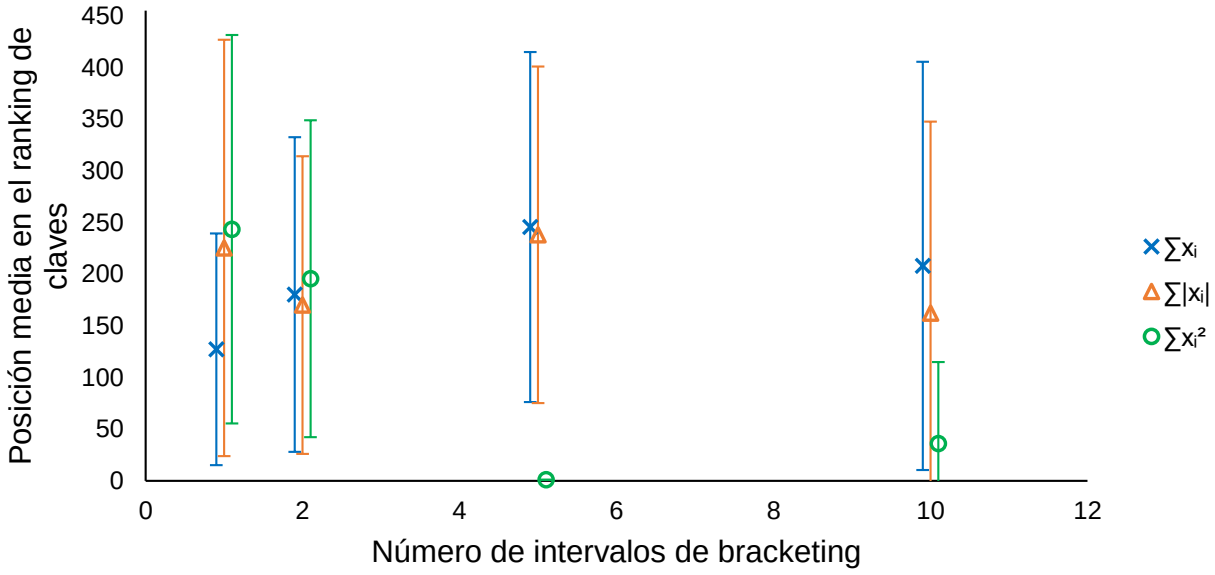


Figura 20: Posición media en el ranking de claves frente al número de intervalos de bracketing. Aunque los valores de los tres métodos para cada intervalo deberían solapar, se han separado para facilitar la lectura, aprovechando que en el eje horizontal los valores deben ser discretos y no hay lugar a confusión.

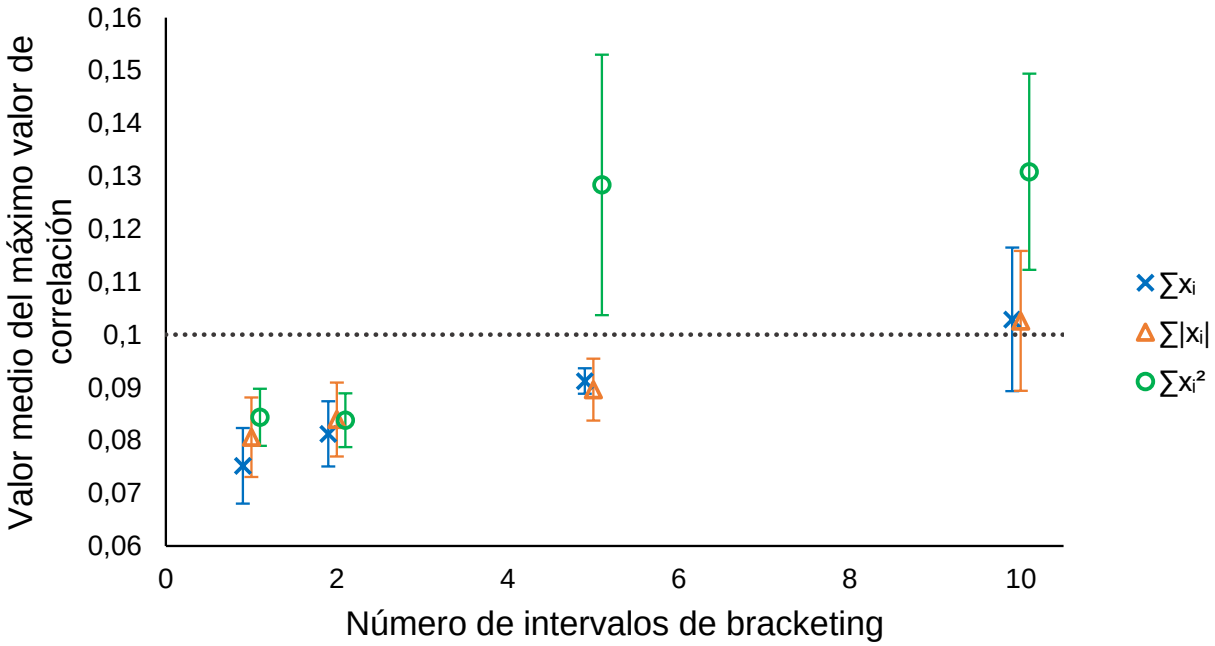


Figura 21: Valor medio del máximo valor de la correlación conseguido en el ataque frente al número de intervalos de bracketing utilizados. Al igual que en las figuras 19 y 20, se ha evitado el solapamiento de los valores para mejorar la claridad. Se ha incluido además una recta paralela al eje horizontal, que representa una suposición acerca del valor hipotético a partir del cual los ataques pueden empezar a resultar satisfactorios.

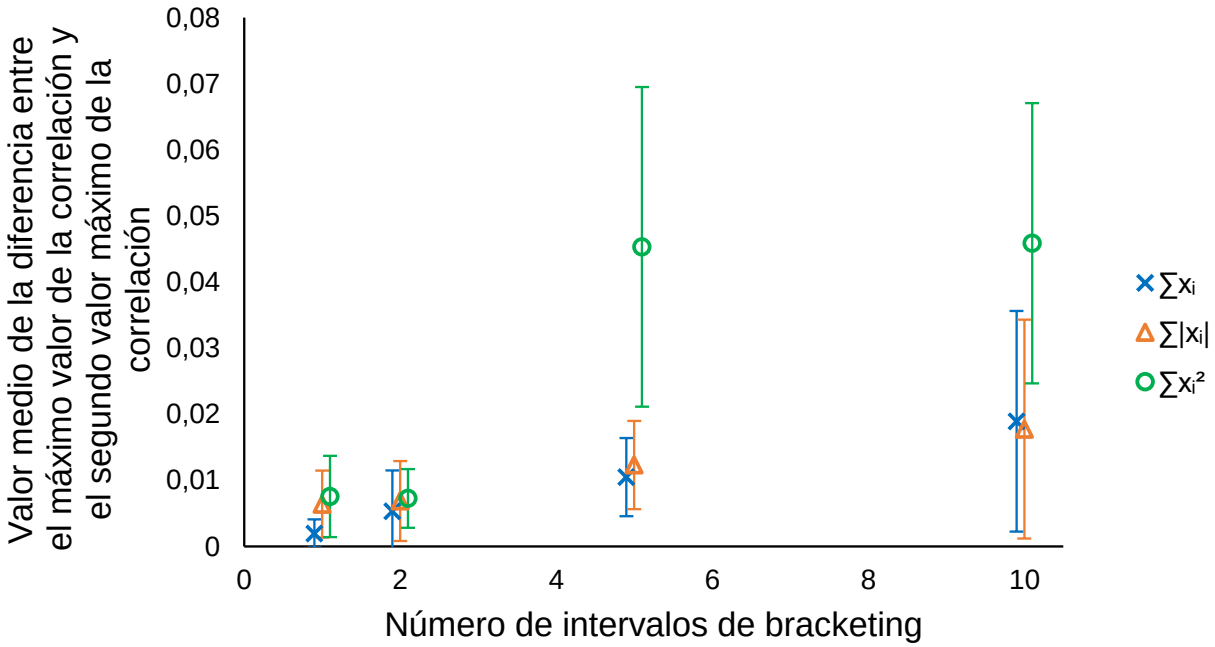


Figura 22: Valor medio de la diferencia entre el máximo valor de la correlación y el segundo valor máximo de la correlación frente al número de intervalos de bracketing utilizados. Al igual que en las figuras 19, 20 o 21, se ha evitado el solapamiento de los valores para mejorar la claridad.

intervalos de bracketing en todos los casos, lo cual indica que teóricamente, aunque no se obtenga la clave, la cantidad de información útil sobre la misma contenida en la traza comprimida debería aumentar al aumentar el número de intervalos de bracketing. Como cabe esperar debido a la mayor eficacia del método, para la suma de cuadrados este crecimiento es mucho más rápido que para la suma simple y la suma de valores absolutos. Además, como en la figura 18 se vio que los resultados también empezaban a mejorar para estos dos últimos métodos, se podría pensar que el hecho de que haga falta un número mínimo de intervalos de bracketing para que comiencen a verse resultados, puede traducirse en que hace falta un mínimo valor medio del máximo valor de la correlación para que empiecen a verse resultados. Como suposición, tan sólo fundada en la mera observación los escasos datos disponibles, sugiero un valor de 0,1 (marcado en la figura con una recta), como valor mínimo. Por último, cabe comentar que, aunque el valor medio del máximo de la correlación se incremente al aumentar el número de intervalos, también lo hace la dependencia de este valor con la clave, que luego parece mantenerse más o menos constante a falta de más datos. Esto no debe ser necesariamente negativo si con un hipotético mayor número de intervalos no considerado en el estudio no sigue aumentando.

Otro factor a considerar muy relacionado con el anterior es el valor medio de la diferencia entre el valor máximo de la correlación y el segundo valor máximo de la correlación obtenidos. O dicho en otras palabras, en qué medida está uno lejos de equivocarse si apuesta por la clave que corresponde al valor máximo de la correlación en lugar de la que corresponde al segundo valor máximo. Su análisis se muestra en la figura 22.

Se puede ver que el comportamiento es prácticamente idéntico al del valor medio del máximo valor de la correlación, lo cuál es bueno, ya que además de incrementarse la probabilidad de acertar la clave (máximo valor de la correlación), disminuye la probabilidad

	$\sum x_i$	$\sum \lvert x_i \rvert$	$\sum x_i^2$
Bracketing de 5	-	-	800 ± 400
Bracketing de 10	900 ± 500	900 ± 500	700 ± 300

Tabla 1: MTD para los distintos métodos de compresión e intervalos de bracketing en los que se acertó la clave.

de equivocarse al elegir entre las primeras claves.

El último factor que queda por considerar es el **MTD**. En esta ocasión, dado que no se pueden estudiar los casos en los que ha fallado la obtención de la clave, hay pocos valores del MTD que analizar, así que una representación gráfica quizás no pueda competir con la representación en forma de tabla (tabla 1) que se ha elegido. De ella se ha omitido el caso de acierto para la suma simple sin bracketing, ya que como sólo ha funcionado para una clave no se puede obtener la desviación típica, y no se sabe cuánto puede oscilar para una clave distinta.

Se puede ver que el número de patrones necesarios es mayor para la suma simple y la suma de valores absolutos que para la suma de cuadrados (y su dependencia con la clave también). Además, en los dos casos de la suma de cuadrados, al aumentar el número de intervalos decrece tanto el número de patrones requeridos como su dependencia con la clave.

3.2. Compresión de trazas de AES

En este caso, se han medido trazas para una sola clave de cifrado. Se han aplicado los tres métodos, tanto sobre la traza completa (sin bracketing), y como con bracketing, dividiendo la traza en dos, cinco y diez intervalos de instantes de tiempo.

Ahora en total se obtienen sólo 12 formas de compresión (1 clave x 3 métodos x 4 tipos de bracketing). De esas formas, 4 pertenecen al método de suma simple, 4 al método de suma de valores absolutos y 4 al método de suma de cuadrados.

Así el número de aciertos es:

- Suma simple: 4 de 4 (100 % de las veces).
- Suma de valores absolutos: 2 de 4 (50 % de las veces).
- Suma de cuadrados: 2 de 4 (50 % de las veces).

Esta vez, el método que consigue mayor número de aciertos es la suma simple (acierta siempre). La suma de cuadrados y la suma de valores absolutos presentan comportamientos similares, acertando en la mitad de las ocasiones.

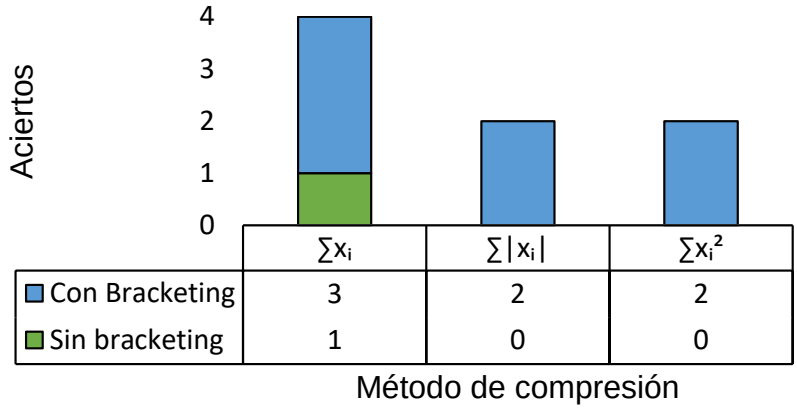


Figura 23: Número de aciertos en función del método utilizado y de si se ha empleado bracketing o no. Recuérdese que 4 es el número máximo de aciertos para este experimento (que coincide con el número de pruebas).

La separación entre los casos en los que se emplea bracketing de los que no (figura 23), muestra que en el caso de la suma simple, la separación ha resultado indiferente (se consigue el máximo número de aciertos sin bracketing que es uno), mientras que en el caso de la suma de valores absolutos y la suma de cuadrados, el bracketing es imprescindible.

Si se piensa en los resultados obtenidos con la S-Box, resulta bastante sorprendente, por un lado, que el método de la suma simple que antes apenas daba ningún resultado, ahora siempre funcione, y por otro lado, que la eficacia del método de la suma de valores absolutos se haya igualado a la del método de suma de cuadrados.

En la figura 24, se puede ver además cómo influye el número de intervalos de bracketing en la eficacia de cada método. En este caso, salvo en la suma simple (donde se mantiene constante porque el ataque siempre funciona), el aumento de la eficacia con el número de intervalos permanece.

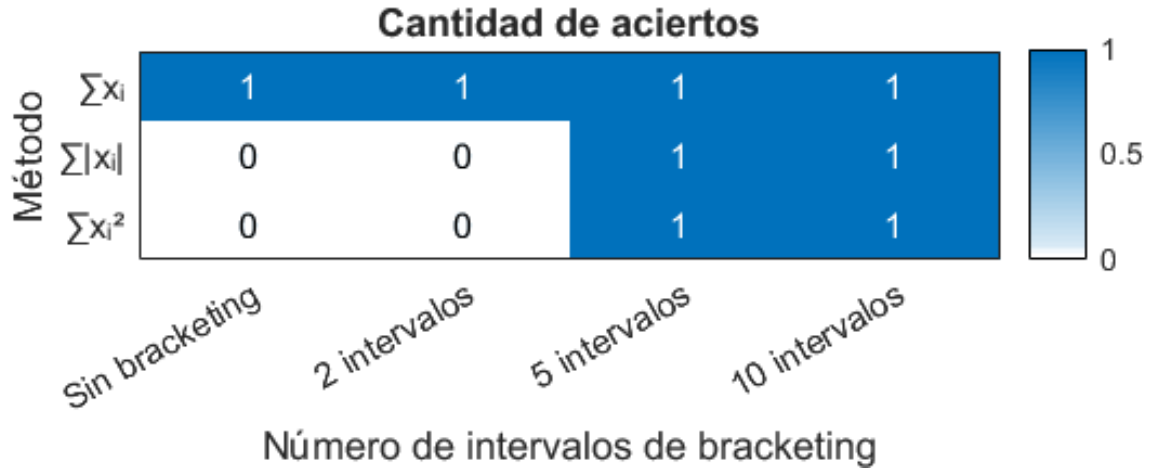


Figura 24: Aciertos según el método de compresión aplicado y el número de intervalos de bracketing elegidos. Las zonas más oscuras representan mayor número de aciertos, y las más claras menos aciertos. En este caso el valor máximo de aciertos es 1.

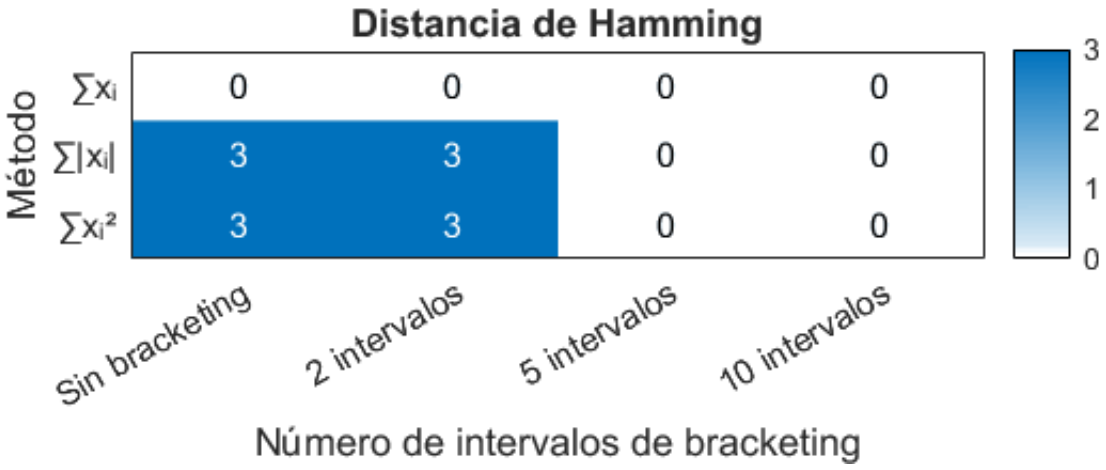


Figura 25: Distancia de Hamming según el método de compresión aplicado y el número de intervalos de bracketing elegidos. Las zonas más oscuras representan mayor distancia de Hamming, y las más claras menos distancia.

Nuevamente se puede usar el resto de información que devuelve el ataque para intentar averiguar más datos. Al igual que en el caso de la S-Box, esta información es:

- Clave configurada en el dispositivo
- Clave devuelta por el ataque
- Máximo valor de correlación
- Diferencia entre valor máximo de correlación y segundo valor máximo de correlación
- Posición en el ranking
- MTD

A diferencia del caso anterior (S-Box), para el caso del AES sólo se utiliza una clave, así que no tiene sentido definir una desviación típica.

El análisis de la distancia de Hamming entre la clave devuelta por el ataque y la configurada en el dispositivo, arroja los resultados de la figura 25. Cuando el ataque falla, la distancia de Hamming siempre es igual a 3. De hecho, se trata siempre de la misma clave. Para este dispositivo la longitud de la parte atacada de la clave es de 8 bits.

Sin embargo, el ranking de claves dado por el ataque no siempre es el mismo (ver tabla 2). A medida que aumenta el número de intervalos de bracketing, la posición en el ranking de la clave correcta va acercándose a la primera posición.

	$\sum \lvert x_i \rvert$	$\sum x_i^2$
Bracketing de 1	39	42
Bracketing de 2	10	14

Tabla 2: Posición en el ranking de la clave correcta en cada uno de los ataques fallidos.

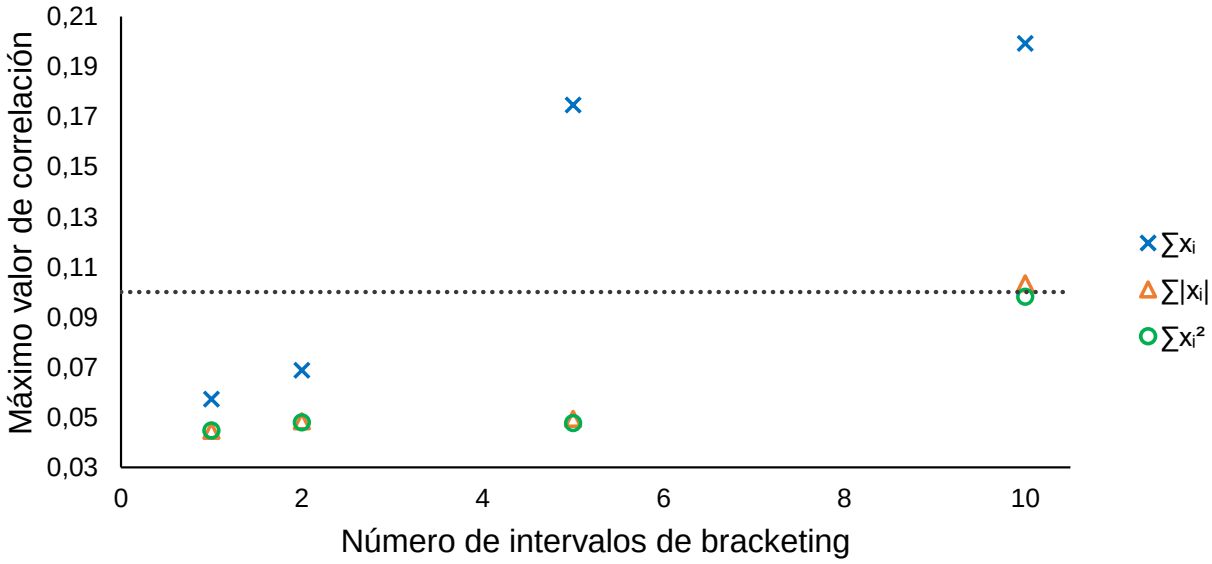


Figura 26: Máximo valor de la correlación conseguido en el ataque frente al número de intervalos de bracketing utilizados. Se ha dibujado una recta paralela al eje horizontal, que representa una la suposición hecha anteriormente (figura 21) acerca del valor hipotético a partir del cual los ataques pueden empezar a resultar satisfactorios.

De nuevo, el comportamiento del valor máximo de la correlación con el número de intervalos de bracketing aclara la situación. Los resultados se muestran en la figura 26, junto con la misma suposición sobre el nivel de correlación mínimo que se hizo para S-Box.

En dicha figura vuelve a observarse el aumento del valor máximo de la correlación con el número de intervalos de bracketing, que en este caso es mucho más rápido para el método de la suma simple que para los otros dos métodos, que tienen un comportamiento parecido. Sin embargo, el límite que se estableció arbitrariamente en el caso anterior (S-Box), no parece ser válido en este caso, ya que se obtienen varios resultados correctos cuando el valor máximo de la correlación está bastante por debajo de este límite.

De nuevo, conviene analizar la diferencia entre el valor máximo de la correlación y el segundo valor máximo de la correlación obtenidos, o en qué medida está uno lejos de equivocarse si apuesta por la clave que corresponde al valor máximo de la correlación en lugar de la que corresponde al segundo valor máximo. La figura 27 responde a esta cuestión.

Se puede ver nuevamente que el comportamiento es prácticamente idéntico al del máximo valor de la correlación, así que también disminuye la probabilidad de equivocarse al elegir entre las primeras claves. Sin embargo se observa una diferencia con la S-Box. Para el método más efectivo, en la S-Box ambos valores tocan techo simultáneamente. Sin embargo, para el AES, la diferencia entre el valor máximo de la correlación y el segundo valor máximo de la correlación toca techo antes de que lo haga el valor máximo de la correlación.

Por último queda por considerar el **MTD**. Los valores para el MTD aparecen en la figura 28. El número de patrones necesarios es menor para la suma simple que para la suma de valores absolutos y la suma de cuadrados. Además, en todos los casos, al

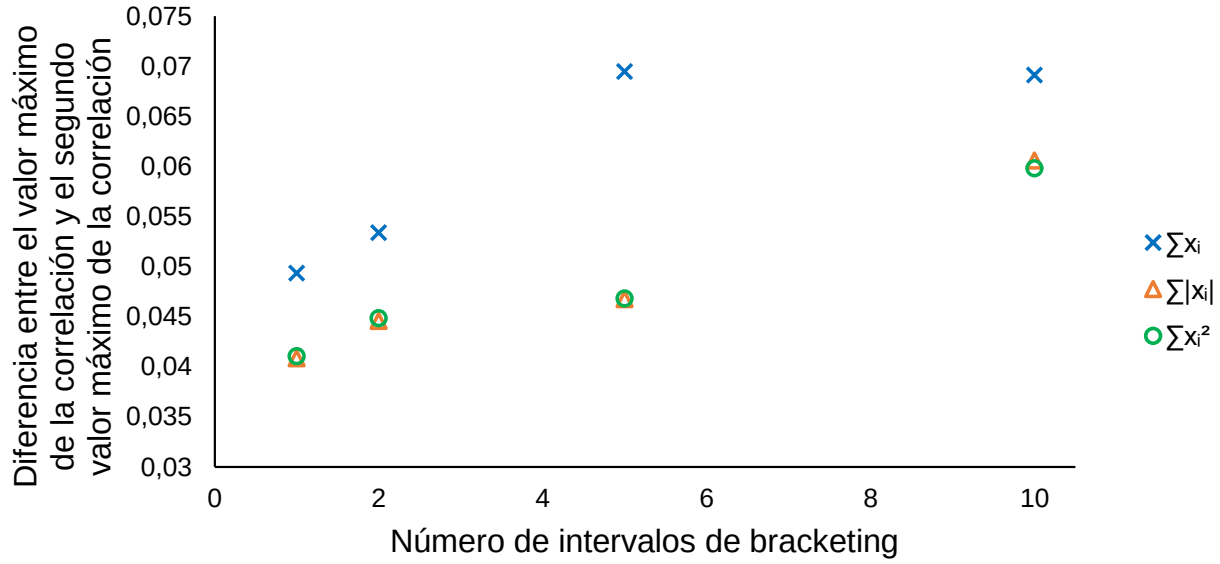


Figura 27: Diferencia entre el máximo valor de la correlación y el segundo valor máximo de la correlación frente al número de intervalos de bracketing utilizados.

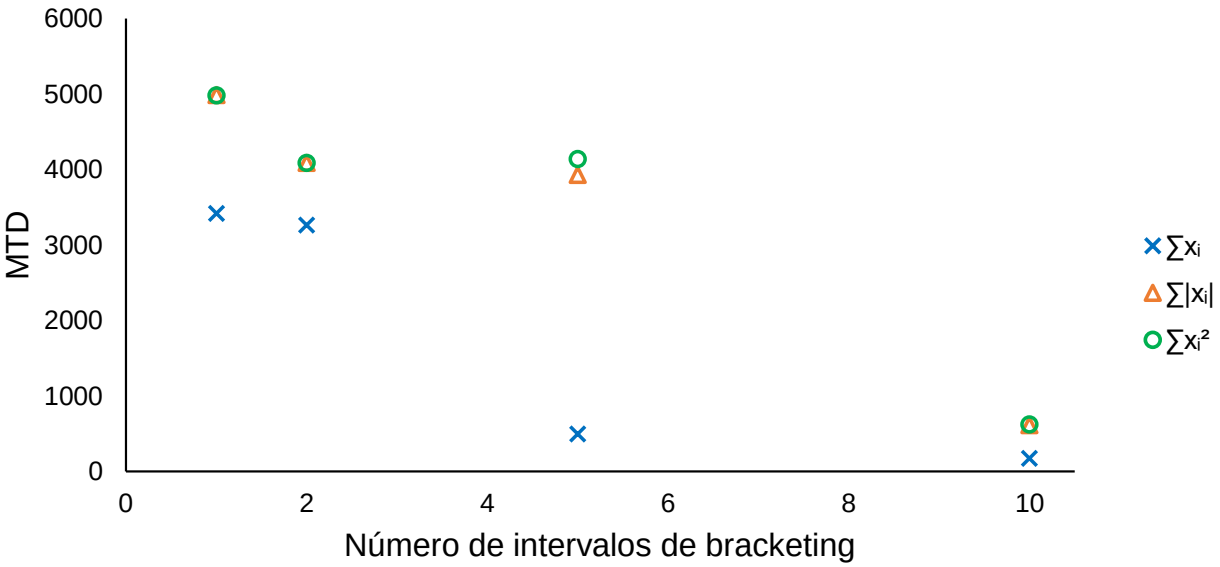


Figura 28: MTD frente al número de intervalos de bracketing utilizados.

aumentar el número de intervalos decrece el número de patrones requeridos, habiendo una caída abrupta en un alrededor de un número de intervalos que depende del método considerado.

3.3. Conclusiones sobre la compresión de trazas

Si se contemplan los resultados de la aplicación de las técnicas de compresión en ambos casos (S-Box y AES), pueden extraerse algunas conclusiones generales.

Ante todo hay que resaltar una conclusión extraída del estudio de la S-Box, que quizás es fundamental porque afecta al resto de resultados. Se trata de la **fuerte dependencia del resultado del ataque con la clave**. En casi todos los datos analizados para S-Box, una desviación típica de valor grande los acompaña. En algunos casos, esta desviación típica supone que a veces unas claves pueden **no acertarse** mientras que otras sí, para el mismo método y número de intervalos de bracketing.

Por otro lado, se ha visto que para S-Box el método que mejor funciona es la suma de cuadrados, mientras que para AES el que mejor funciona es la suma simple. Esto quiere decir que no puede afirmarse que de forma general que uno de los tres métodos sea mejor que otro, ya que dependiendo del dispositivo a atacar tienen mas éxito unos métodos u otros.

Con el fin de esclarecer esta cuestión, se pueden comparar las figuras 6 y 7 (las trazas de ejemplo para S-Box y AES), con las figuras 29 y 30 (las trazas comprimidas utilizando bracketing de 10 intervalos para los tres métodos). Se ha escogido el bracketing de 10 para la comparación porque se aprecia mejor la forma de la señal, y por tanto las diferencias entre los distintos métodos de compresión. Podría haberse escogido otro bracketing para la comparación, pero habría sido más difícil llevarla a cabo.

En el caso de AES (figura 30), se detecta con relativa facilidad la diferencia entre las trazas, que debe ser el motivo por el que la suma simple es más eficaz que la suma de valores absolutos o la suma de cuadrados. Si se atiende a la traza de ejemplo (figura 7), y luego a la figura 30, se observa que los métodos de suma de valores absolutos y suma de cuadrados no respetan una característica de la forma de la señal que sí respeta el de la suma simple, que es el hecho de que la primera caída en el consumo de potencia lo lleva por debajo del valor inicial de consumo. En el resto de la señal sí parece respetarse la forma para los tres métodos.

En el caso de S-Box, (figura 29), es más complicado detectar la diferencia que causa que la suma de cuadrados sea más eficaz que la suma de valores absolutos y esta a su vez más eficaz que la suma simple. Una hipótesis que podría explicar la diferencia de la suma de cuadrados con las otras dos es que resalta la diferencia entre los valores de los picos. Quizás si la información útil se encuentra allí, pone más fácil al algoritmo el trabajo. En cuanto a la diferencia entre la suma de valores absolutos y la suma simple, lo único que se observa es que la suma simple no respeta la pequeña caída inicial (empieza ya desde abajo), mientras que la suma de valores absolutos sí que la respeta, así que en este último caso la forma es más parecida a la de la traza original.

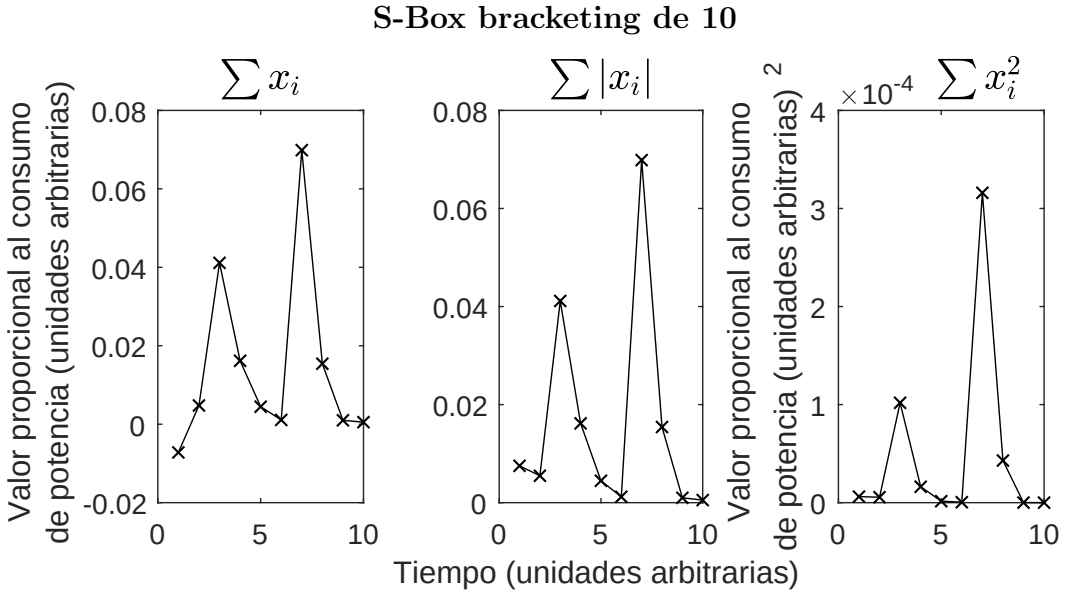


Figura 29: Señal de consumo de potencia de S-Box comprimida con cada uno de los métodos para el bracketing de 10 intervalos.

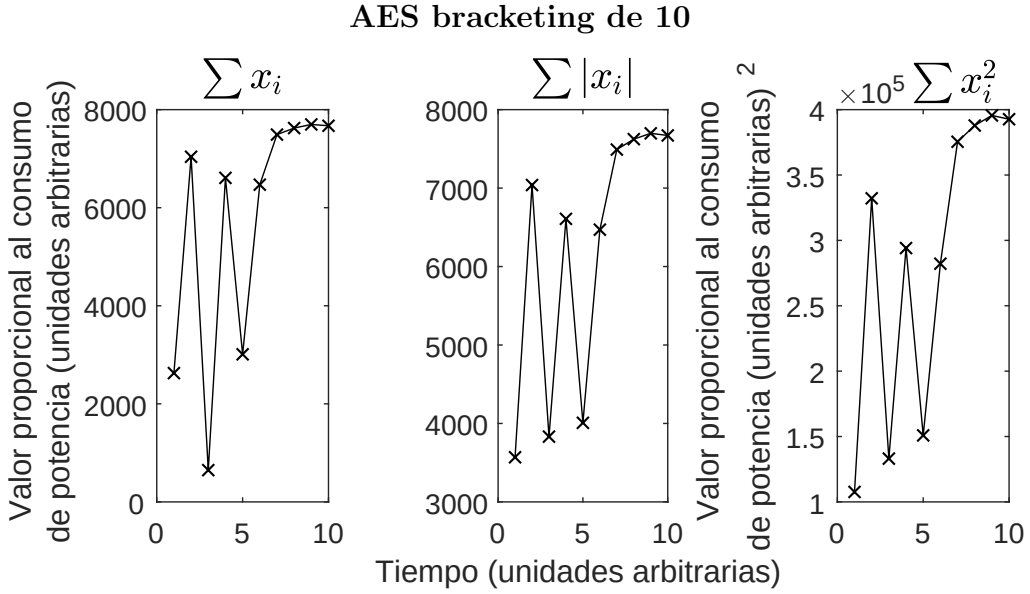


Figura 30: Señal de consumo de potencia de AES comprimida con cada uno de los métodos para el bracketing de 10 intervalos.

En cualquier caso, a partir de las diferencias entre los tres métodos para AES, se puede deducir que la deformación de la señal está causada por la eliminación del signo negativo del valor proporcional al consumo de potencia (ya que en la suma simple esto no ocurre). Esto sugiere que si se añade a las señales una componente en DC tal que no haya ningún valor negativo en el valor proporcional al consumo de potencia, quizás estos problemas desaparezcan y los tres métodos puedan tener un rendimiento más parecido.

En cuanto al bracketing, los resultados muestran que es una práctica que mejora la efectividad siempre, independientemente del método de compresión. En el caso de la S-Box, simplemente el mayor porcentaje de acierto utilizando bracketing lo hace evidente, mientras que en AES, aunque para los métodos de suma de valores absolutos y suma de cuadrados se mantiene la validez de la afirmación, para el método de la suma simple aparentemente no (porque acierta siempre), lo cuál es algo confuso.

Para comprender este suceso, hay que recordar que el ataque se basa en los **coeficientes de correlación**, que no dejan de ser magnitudes estadísticas, que **indican probabilidades y no certezas**.

Teniendo esto en cuenta, merece la pena recurrir al (valor medio en S-Box) valor máximo de la correlación. Entonces el comportamiento sí es uniforme tanto para AES como para S-Box, en el sentido de que todos los métodos incrementan su valor máximo de la correlación cuando aumenta el número de intervalos de bracketing. Además el método más eficaz incrementa su valor máximo de la correlación de manera mucho más rápida que los otros dos. Este aumento en el caso del método más eficaz es más o menos abrupto, y luego aparentemente satura. Lo cuál implica que hace falta un número mínimo de intervalos de bracketing para que el método más eficaz funcione adecuadamente, y una vez alcanzado dicho número, merece poco la pena incrementar el número de intervalos de bracketing.

Afortunadamente, la diferencia entre el valor máximo de la correlación y el segundo valor máximo de la correlación se comporta de la misma manera que el valor máximo de la correlación al aumentar el número de intervalos de bracketing, así que no sólo aumenta la cantidad de información útil escondida en la traza comprimida, sino que además aumenta la certeza de que la clave correcta es la primera en el ranking y no las siguientes.

Este análisis del valor máximo de la correlación, particularmente el de la suma simple, muestra que ha sido posible acertar la clave incluso aunque el valor máximo de la correlación está bastante por debajo del límite hipotético que se estableció en el análisis de la S-Box. Para explicar esto, hay que tener en cuenta que las trazas de S-Box provienen de una simulación eléctrica, por tanto no tienen ruido y representan el escenario perfecto para un atacante. Por el contrario, las trazas de AES se extraen de una implementación en un FPGA, y se mide no sólo el consumo del circuito, sino también ruido. Este factor puede ser el responsable de la reducción de la magnitud de los picos de máxima correlación. Aun así, realizar el experimento para AES con más de una clave, comprobando si se siguen obteniendo las claves correctas con un valor bajo de la correlación, podría clarificar esta cuestión.

El resto de factores analizados (distancia de Hamming, ranking de claves, MTD),

	Tamaño de las trazas	Efectividad del ataque	Tiempo de procesado
S-Box sin procesado	1,42 MB	95 %	-
AES sin procesado	4,38 MB	100 %	-
S-Box $\sum x_i^2$ Bracketing de 5	50 KB	100 %	$\approx$ 20 s
S-Box $\sum x_i^2$ Bracketing de 10	98 KB	80 %	$\approx$ 30 s
AES $\sum x_i$ Bracketing de 1	11 KB	100 %	$\approx$ 10 s
AES $\sum x_i$ Bracketing de 2	20 KB	100 %	$\approx$ 10 s
AES $\sum x_i$ Bracketing de 5	46 KB	100 %	$\approx$ 12 s
AES $\sum x_i$ Bracketing de 10	83 KB	100 %	$\approx$ 14 s
S-Box mejora	93,3 % a 96,6 %	-15 % a 5 %	-
AES mejora (peor caso)	98,2 %	0 %	-

Tabla 3: Comparación del tamaño en disco de las trazas (para una clave) y la efectividad del ataque (porcentaje de aciertos) entre las mismas trazas sin procesar y procesadas para varios de los métodos que mejor han funcionado. Se muestra además el tiempo requerido para procesar las trazas en un equipo con un procesador Intel Core i7 720QM y 4GB de RAM, con un script no optimizado para el uso de múltiples núcleos del procesador. Se observa que se ha reducido significativamente el tamaño de las trazas, sin disminuir apenas la efectividad de los ataques, añadiendo tan sólo la penalización del tiempo que dura el procesado. El tamaño en disco se refiere trazas almacenadas en la versión 7 del formato binario ".mat" de MATLAB.

parecen seguir el ritmo del valor máximo de la correlación, en el sentido de que si este último aumenta, dichos factores toman valores que indican una facilidad mayor o una cercanía mayor a obtener la clave.

Por último, cabe preguntarse si se ha alcanzado el objetivo que se pretendía, que es permitir la realización de un ataque con menos recursos computacionales. Teniendo en cuenta, que la matriz $\boldsymbol{T'}$ que se genera para el ataque tiene como máximo 10 columnas con los intervalos de bracketing considerados, en lugar de las 180 (longitud de una traza de S-Box) o 1500 (longitud de una traza de AES) que tenía cuando se realizaba el ataque sin un procesado previo, es de esperar que el tiempo de ataque se reduzca. Si además se observan los datos de la tabla 3 sobre el tamaño en disco de las trazas, la efectividad (porcentaje de aciertos) y el tiempo de procesado, se puede ver que se ha reducido en más de un 90 % el tamaño de las trazas en todos los casos, y la efectividad de los ataques se ha visto relativamente poco perjudicada. Por tanto, puede concluirse que se ha logrado alcanzar el objetivo deseado.

4. Filtrado de la señal de reloj en Trivium

En este caso, se sabía previamente, que con las trazas originales sin ningún tipo de procesado no se consigue acertar la clave configurada en el dispositivo, porque la señal contiene demasiado ruido (ver trabajo de fin de grado [8]). Este ha sido el motivo proncipal que ha motivado llevar a cabo el filtrado.

Sin embargo, el filtrado de la señal de reloj **no ha dado resultados positivos, porque no se ha conseguido averiguar la clave**.

Recuérdese que se ha realizado el filtrado eliminando 10, 35, 75 y 150 armónicos en cada caso. Dado que **el ataque ha sido equivalentemente infructuoso para todos los experimentos**, no tiene sentido realizar un análisis detallado de cada intento igual que el que se hizo para S-Box y AES. Sin embargo, a continuación se muestra una comparación entre lo que, de acuerdo con la literatura, debería obtenerse cuando el ataque tiene éxito y lo que se obtiene en el caso estudiado.

El método de ataque (consultar [13] para una descripción más detallada) difiere ligeramente del explicado en la sección 2.2. Trivium es un cifrador de flujo que utiliza una clave de 80 bits y produce un bit de una secuencia pseudoaleatoria (conocido como keystream, que se utiliza para realizar a continuación una operación lógica Xor con los datos a cifrar y obtener el texto cifrado) por ciclo de reloj. En este caso, lo que se ataca es directamente el generador de bits aleatorios, así que, no es suficiente con medir un ciclo de reloj para obtener un fragmento de clave, si no que hay que dejar que el aparato opere durante 81 ciclos de reloj para que proporcione suficiente información como para realizar el ataque. Esto se debe, a que en cada ciclo de reloj i-ésimo lo que se puede obtener es un valor σ_i, no un fragmento de clave directamente (que llamábamos k_i en la sección 2.2). Este valor σ_i, es función de los bits de la clave, y a través de ellos, de otros valores σ. Así, los 81 valores de σ forman un sistema de ecuaciones, y una vez se han encontrado los 81 valores se puede resolver el sistema para encontrar la clave.

El valor σ_i es un valor lógico (0 ó 1), que se utiliza en una puerta Xor del dispositivo. Así que en cada ciclo de reloj sólo hay dos posibles valores para sigma. Sin embargo, lo que se observa cuando se mide el consumo de potencia, es el cambio en el valor lógico de la salida de dicha puerta (que pueden ser los siguientes: $0\rightarrow0$, $0\rightarrow1$,$1\rightarrow0$ y $1\rightarrow1$). De modo que a la hora de hacer la correlación en el ataque, hay cuatro valores posibles de consumo con los que correlacionar, en lugar de sólo dos.

En la figura 31 (extraída directamente de la figura 4 de [13]), se muestra un ejemplo de lo que debería obtenerse en un ataque que da resultados positivos. Dicho ejemplo de ataque se ha llevado a cabo sobre un sistema simulado. Se observa que la gran mayoría de los puntos tiene una correlación similar, excepto unos pocos que tienen picos de alta correlación. Además, según el valor σ_i que se está intentando obtener, el pico con mayor valor absoluto (que indica la transición correcta, y conociendo el valor de sigma anterior, el valor de sigma correcto), se va desplazando hacia la derecha.

En la figura 32, se muestran los resultados obtenidos en este trabajo para el coefi-

Resultado esperado en un ataque exitoso sobre Trivium

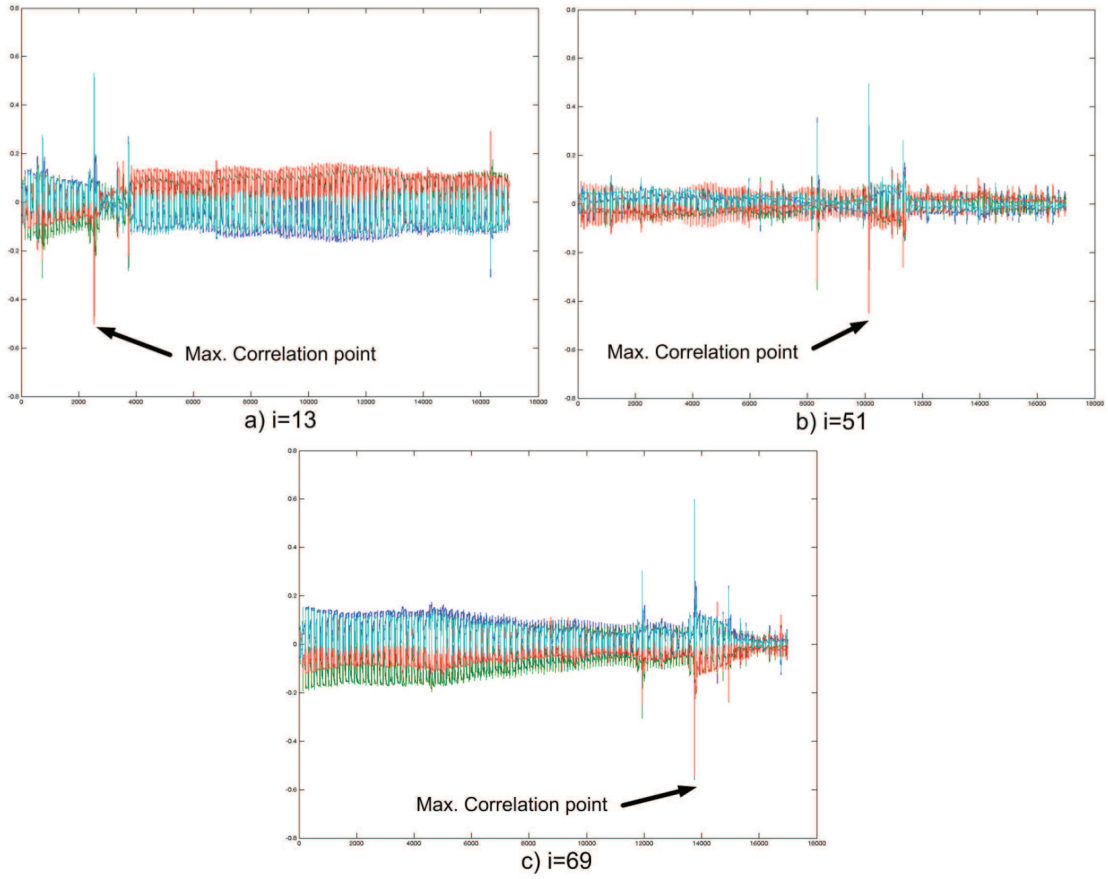
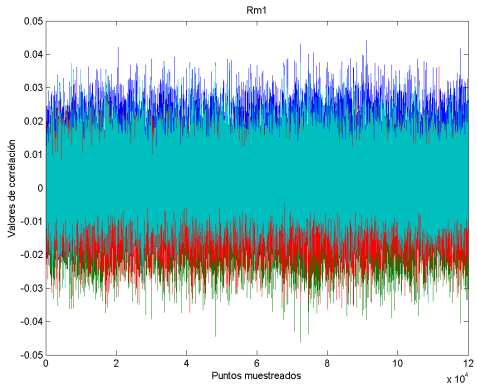
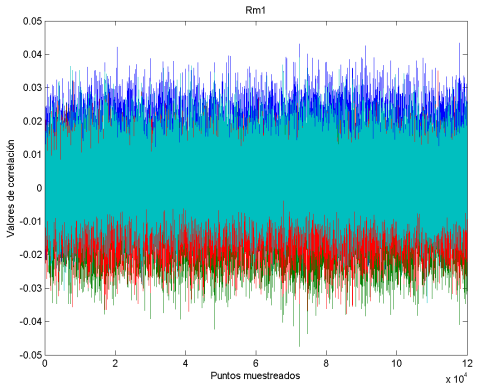


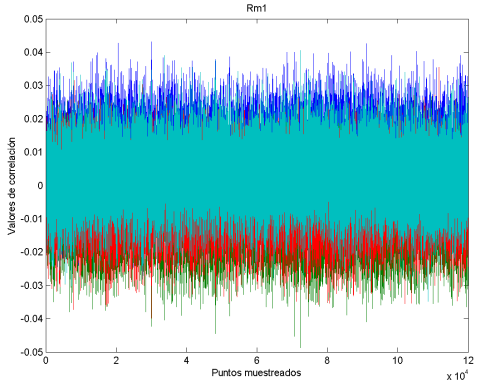
Figura 31: Ejemplo de ataque exitoso sobre Trivium (extraído de [13, figura 4]). En el eje horizontal se muestran los puntos del ciclo de reloj atacado, y en el vertical los valores de la correlación para cada punto. Aparecen cuatro curvas en cuatro colores distintos. Cada una representa la correlación con un tipo de transición ($0\to0$, $0\to1$,$1\to0$ y $1\to1$) en la puerta lógica Xor vulnerable del dispositivo. Cada una de las tres gráficas corresponde a un intento de obtener un valor σ_i distinto.



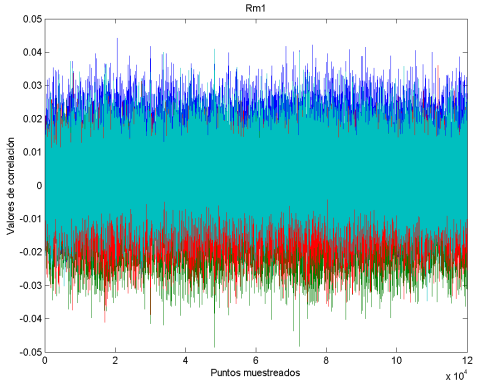
(a) Sin filtrado



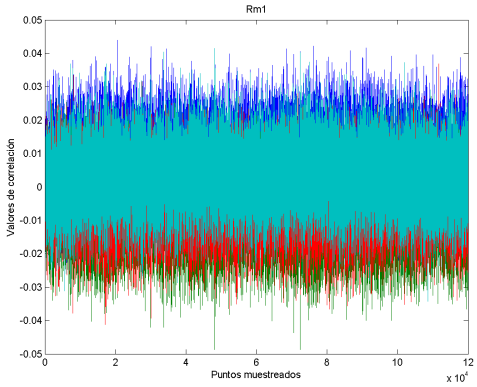
(b) 10 Armónicos



(c) 35 Armónicos



(d) 75 Armónicos



(e) 150 Armónicos

Figura 32: Resultados obtenidos en el ataque sobre Trivium para el valor σ_1. En el eje horizontal se representan los instantes de tiempo atacados, y en el eje vertical los valores de correlación para las diferentes transiciones, correspondiéndose cada color con un tipo de transición (0→0, 0→1,1→0 y 1→1) en la puerta lógica Xor vulnerable del dispositivo.

ciente σ_1. En este caso no existe ningún pico de correlación significativo, la correlación se mantiene uniforme para las cuatro posibles transiciones a lo largo de todo el ciclo de reloj y no puede apostarse por ninguna en concreto, por tanto, no se puede averiguar la clave configurada en el dispositivo. Los resultados para el resto de valores σ_i son cualitativamente idénticos, no obteniéndose ningún pico significativo de correlación en ningún caso. Esto impide escoger valores de σ, por tanto no puede resolverse el sistema de ecuaciones, y no hay forma de obtener la clave.

5. Conclusiones

En este trabajo, se ha considerado por una parte la compresión de trazas de consumo de potencia, y por otro lado el filtrado de la señal de reloj en señales de consumo de potencia de dispositivos criptográficos.

La **compresión de las trazas** se ha basado en tres técnicas diferentes (suma simple, suma de valores absolutos y suma de cuadrados), propuestas en previamente en [5]. Además se ha introducido la idea del "bracketing", es decir, dividir las trazas en varios intervalos y aplicar la compresión a cada intervalo por separado. Se han aplicado las técnicas a las trazas de dos implementaciones de algoritmos criptográficos, una simulada de una S-Box (substitution box) y otra física del algoritmo AES (implementado en un FPGA).

Tras atacar ambos sistemas con las trazas comprimidas en lugar de las originales, los resultados han mostrado que efectivamente la compresión de trazas permite reducir muy significativamente su tamaño[2] (y por tanto el poder computacional requerido en los ataques), sin destruir la información necesaria para los ataques, aunque con algunas salvedades.

La compresión destruye menos información cuantos más intervalos de bracketing se utilicen (dado que a mayor número de intervalos, mayor parecido de la traza comprimida con la original), cuando no se utiliza bracketing, es cuando se obtiene el peor resultado. Además parece que por encima de un número mínimo de intervalos de bracketing se percibe una mejoría abrupta y luego no hay una mejora significativa. La forma particular de cada tipo de traza y el hecho de que contenga valores negativos o no, influye en la efectividad de cada método. Por tanto, la compresión es un instrumento que debe utilizarse con cautela, empleando un número de intervalos de bracketing suficiente, y prestando atención a la traza particular que se está comprimiendo.

El **filtrado** de la señal de reloj, de frecuencia que se supone conocida, se ha realizado mediante una transformada de Fourier (algoritmo Fast-Fourier-Transform) de la traza original, de forma que se pueden localizar las componentes relacionadas con la señal de reloj y eliminarlas. Se ha aplicado al caso particular de una implementación del algoritmo Trivium en un dispositivo físico.

[2] Ver tabla 3.

En este caso, los resultados han mostrado que el filtrado de la frecuencia de reloj es completamente inefectivo, independientemente del numero de armónicos de la frecuencia de reloj que se filtren. No se consigue distinguir una clave cuyo consumo de potencia presente una correlación con el de la clave correcta notablemente más alta que el resto.

Para **seguir progresando** en la línea de la compresión de trazas, como se ha mencionado ya en la sección 3.3, podría estudiarse incluir un offset (sumar un valor en DC) a las señales, de forma que no contengan valores negativos, y a continuación estudiar si desaparece o no la dependencia de la eficacia de cada método con la forma de la traza. Además, para aprovechar este método con fines prácticos, como pueden ser por ejemplo el acceso a un dispositivo cifrado en una investigación policial, sería necesario establecer claramente, además de la cuestión anterior, cuántos intervalos de bracketing serían necesarios para garantizar que la clave se puede obtener con las trazas comprimidas, suponiendo que se obtiene con las trazas sin comprimir.

Con respecto a Trivium, está claro que si se quiere conseguir atacar con éxito el dispositivo, serán necesarias otras técnicas de procesado, o puede que incluso otros tipos de ataques laterales. El filtrado de la señal de reloj no es el camino.

En cualquier caso, el conocimiento tanto del resultado positivo de los ataques con las trazas comprimidas, como el del resultado negativo del ataque sobre Trivium tras el filtrado, **suponen un avance significativo** en el devenir de la investigación en el campo de la microelectrónica para la seguridad que lleva a cabo el grupo de investigación del IMSE del que forman parte los tutores de este trabajo (grupo TIC-180[3]), ya que dicho grupo tendrá la posibilidad de aprovechar la compresión de las trazas para reducir la carga computacional de sus ataques, y además podrán descartar definitivamente el filtrado de la señal de reloj como método para intentar atacar con éxito el dispositivo ASIC que ejecuta el algoritmo Trivium.

[3]Sitio web: `http://www.imse-cnm.csic.es/es/lineas/tic180_sec.php`.

6. Bibliografía

[1] Schneier Bruce. «Applied cryptography: protocols, algorithms, and source code in C». En: *New York: Wiley* (1996).

[2] Paul C Kocher. «Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems». En: *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*. Springer-Verlag. 1996, págs. 104-113.

[3] Yu-Ichi Hayashi y col. «Analysis of electromagnetic information leakage from cryptographic devices with different physical structures». En: *IEEE Transactions on Electromagnetic Compatibility* 55.3 (2013), págs. 571-580.

[4] Paul Kocher, Joshua Jaffe y Benjamin Jun. «Differential Power Analysis». En: *Annual International Cryptology Conference*. Springer. 1999, págs. 388-397.

[5] Stefan Mangard, Elisabeth Oswald y Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Vol. 31. Springer Science & Business Media, 2008.

[6] ETSI/SAGE. «3rd Generation Partnership Project, 3GPP TS 35.202 version 7.0.0 release 6, Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 2: KASUMI Specification». En: (2007).

[7] Ingrid Verbauwhede, Patrick Schaumont y Henry Kuo. «Design and performance testing of a 2.29-GB/s Rijndael processor». En: *IEEE Journal of Solid-State Circuits* 38.3 (2003), págs. 569-572.

[8] Irene Durán. «Análisis de la seguridad en dispositivos criptográficos frente a ataques laterales». Trabajo de Fin de Grado. Universidad de Sevilla, 2015.

[9] Martin R Albrecht y col. «Block ciphers–focus on the linear layer (feat. PRIDE)». En: *International Cryptology Conference*. Springer. 2014, págs. 57-76.

[10] Subhadeep Banik y col. «Midori: A block cipher for low energy». En: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2014, págs. 411-436.

[11] Christophe De Canniere y Bart Preneel. «TRIVIUM Specifications». En: *eSTREAM, ECRYPT Stream Cipher Project* 2006 ().

[12] Kris Tiri y col. «Prototype IC with WDDL and differential routing–DPA resistance assessment». En: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2005, págs. 354-365.

[13] E. Tena-Sánchez y A. J. Acosta. «DPA vulnerability analysis on Trivium stream cipher using an optimized power model». En: *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. Mayo de 2015, págs. 1846-1849. DOI: 10.1109/ISCAS.2015.7169016.

7. Anexo I. Código para el procesado

En este anexo, se muestra el código empleado para la compresión de las trazas y el filtrado de la señal de reloj.

El núcleo fundamental está formado por el código 1, que comprime cualquier traza de consumo de potencia que reciba como entrada, y el código 2, que recibe una traza como entrada y filtra tanto la componente de 2MHz de frecuencia como el número de armónicos superiores definido en la función.

Código 1: comprimir.m: Función que comprime cualquier tipo de traza.

```
function [ trazacomprimida ] = comprimir( traza, metodo, opciones )
%COMPRIMIR Comprime la traza que recibe como entrada.
%   Recibe como argumento una estructura, llamada traza. La estructura... debe
%   contener un campo llamado contenido, que debe ser una matriz de de... una
%   fila y tantas columnas como instantes muestreados tenga la traza. ... Se
%   pueden especificar límites para que se utilice sólo un fragmento ... de la
%   traza y se ignore el resto en la operación (bracketing). Si no se
%   especifica ningún límite en las opciones se comprimirá sin ... bracketing
%   por defecto.
%   Es necesario especificar también el método del compresión como ... variable
%   string en el segundo argumento, qée puede ser uno de los tres
%   soportados ('rawintegration','sumofabsolutevalues','sumofsquares')... , que
%   son los tres estudiados en el trabajo, o bien especificar ... cualquier
%   otro, en cuyo caso la función pasará la traza y las opciones a ... otra
%   función en el mismo directorio de trabajo con el nombre
%   "comprimir_metodo", que el usuario puede crear a su elección para
%   probar otros métodos.
%   Es MUY importante tener en cuenta que las trazas que devuelve esta
%   función son escalares. Para hacer bracketing de cinco intervalos ... por
%   ejemplo, hay que llamar a la función cinco veces con los límites ... adecuados.

if nargin < 3
   opciones.a = [];
end

if strcmpi(metodo,'rawintegration') || strcmpi(metodo,'... sumofabsolutevalues') || strcmpi(metodo,'sumofsquares')
    %Si el método especificado está soportado, continuar.

    %Comprobar si se debe utilizar bracketing.
    hayl=0;
    if isfield(opciones,'limites')
```

```
32          hayl=1;
33          limiteinf=opciones.limites(1);
34          limitesup=opciones.limites(2);
35      else
36          limiteinf=1;
37          limitesup=length(traza.contenido);
38      end
39
40      %Comprimir la traza según el método especificado.
41      trazacomprimida=traza; %Copiar la traza original para incluir sus ...
            metadatos.
42      suma=0;
43      for i=limiteinf:limitesup
44          if strcmpi(metodo,'rawintegration')
45              suma=suma+traza.contenido(i);
46          elseif strcmpi(metodo,'sumofabsolutevalues')
47              suma=suma+abs(traza.contenido(i));
48          elseif strcmpi(metodo,'sumofsquares')
49              suma=suma+traza.contenido(i)^2;
50          end
51
52      end
53      trazacomprimida.contenido=suma; %Introducir la suma en la traza de...
            salida.
54
55      %Escribir el método empleado en los metadatos de la traza de de ...
            salida.
56      switch metodo
57          case 'rawintegration'
58              trazacomprimida.compresion='rawintegration';
59          case 'sumofabsolutevalues'
60              trazacomprimida.compresion='sumofabsolutevalues';
61          case 'sumofsquares'
62              trazacomprimida.compresion='sumofsquares';
63      end
64
65      if (hayl) %Incluir límites en los metadatos de la traza de salida.
66          trazacomprimida.compresion=[trazacomprimida.compresion '...
                _limits(from' num2str(limiteinf) 'to' num2str(limitesup) '...
                )'];
67      end
68
69  else
70      %Si el método de compresión no está soportado, llamar a la función
71      %adecuada.
72      comprimir_metodo=str2func(['comprimir_' metodo]);
73      trazacomprimida=comprimir_metodo(traza, opciones);
74  end
75
76  end
```

Código 2: filtrar_trivium.m: Función que filtra la señal de reloj de las trazas de Trivium.

```
1  function [ trazafft ] = filtrar_trivium(traza, f)
2  %FILTRAR_TRIVIUM Elimina la señal de reloj de las trazas de Trivium
3  %   Recibe como una frecuencia de reloj y una argumento una traza,
4  %   que consiste en una matriz con una fila y tantas columnas como
```

```
5  %   instantes muestreados, y devuelve como salida una traza filtrada
6  %   EN EL DOMINIO DEL TIEMPO (no dejarse engañar por el nombre de la
7  %   variable de salida, que hace que parezca que se devuelve en el ...
       dominio
8  %   de la frecuencia). Esta traza ya puede utilizarse para el ataque. ...
       El
9  %   número de armónicos a filtrar se ha dejado dentro de la función y ...
       no
10 %   como argumento.
11
12 interv=1/(2.5*10^9); %Tiempo entre lecturas de potencia, podría ...
       calcularse simplemente a partir de la frecuencia.
13 freloj=f; clear f;
14
15 instantes=[0:(length(traza)-1)]*interv; %Instantes de tiempo ...
       correspondientes a los puntos muestreados.
16 referencia=max(traza)*sin(2*pi*freloj*instantes); % Seeñal sinusoidal ...
       de referencia con la frecuencia de la señal de reloj.
17
18 %Calcular transformada de la referencia
19 referenciafft=fft(referencia);
20 referenciafftmodulo=referenciafft.*conj(referenciafft);
21 %Calcular transformada de la señal
22 trazafft=fft(traza);
23
24 %%FILTRAR la señal de ejemplo
25 %Encontrar la frecuencia de 2MHz
26 fs0=find(referenciafftmodulo==max(referenciafftmodulo));
27
28 narmonicos=10; %Número de armónicos que van a filtrarse.
29
30 %Filtrado parte positiva
31 f0=fs0(1);
32 for armonico=1:narmonicos
33 %       trazafft(armonico*f0)=0;
34         indice=f0*armonico-(armonico-1)*1; %Esta secuencia es la que ...
             siguen los armónicos de la señal de referencia cuando se ...
             dibujan.
35         trazafft(indice)=0; %Eliminación completa de la componente ...
             correspondiente.
36 end
37
38 %Filtrado parte negativa. Parece complicado de entender por qué la
39 %parte negativa está al final de la señal, pero MATLAB hace las
40 %transformadas así. Pone la parte negativa a continuación de la ...
       positiva.
41 f0=length(trazafft)-fs0(end);
42 for armonico=1:narmonicos
43         indice=length(trazafft)-f0*armonico-(armonico-1); %Esta ...
             secuencia es la que siguen los armónicos de la señal de ...
             referencia cuando se dibujan.
44         trazafft(indice)=0; %Eliminación completa de la componente ...
             correspondiente.
45 end
46
47 %%TRANSFORMADA INVERSA
48 trazafft=ifft(trazafft);
49 %Empleo el móddulo para no perder información (como la transformada
```

```
50 %numérica no es perfecta aparece parte imaginaria al hacer la ...
      transformada
51 %inversa). Aun así no hay mucha diferencia entre considerarla de ...
      alguna
52 %manera y no hacerlo (como es de esperar).
53 for i=1:length(trazafft);
54     valor=sqrt(trazafft(i).*conj(trazafft(i)));
55     if real(trazafft(i)) > 0
56         trazafft(i)=valor;
57     else
58         trazafft(i)=-valor;
59     end
60 end
61
62 end
```

Dichas funciones por sí mismas no pueden hacer nada, necesitan de un script que las llame de manera adecuada (en especial en el caso del código 1, que no es capaz de hacer el bracketing por sí mismo, ya que produce como salida un escalar).

Para llevar a cabo todo el procesado, han sido necesarios los archivos de código que se mencionan a continuación. Dichos archivos son los responsables de las llamadas necesarias.

Compresion.m Fichero principal que comprime las trazas de S-Box o AES (según como se configure).

comprimir.m Función que se llama para comprimir cada traza individual de S-Box o AES.

leer_AES.m Función que lee las trazas de AES.

leer_sbox9.m Función que lee las trazas de S-Box.

transformar_formatoataque.m Función que transforma las trazas procesadas a un formato compatible con el de los scripts de ataque del IMSE.

textprogressbar.m Simplemente muestra una barra de progreso, tomado de `https://es.mathworks.com/matlabcentral/fileexchange/28067-text-progress-bar`.

Trivium.m Script principal que filtra las trazas de Trivium.

filtrar_trivium.m Función que se llama para filtrar cada traza individual de Trivium.

Incluir todo este código haría que se excediera el límite de páginas para este trabajo. En su lugar, se incluyen sólamente las partes relevantes del archivo "Compresion.m", en el código 3. Con los códigos 1, 2, 3 y el de los ataques del IMSE, se podría, sin mucha dificultad, reproducir los resultados.

Código 3: Compresion.m: Fichero principal que comprime las trazas de SBOX o AES (según como se configure).

```
clear

%% Configuración

config.trazas.ubicacion='.\Trazas\';
config.trazas.nombre={'ivdd_1' 'ivdd_2' 'ivdd_3' 'ivdd_4' 'ivdd_5' '...
    ivdd_6' 'ivdd_7' 'ivdd_8' 'ivdd_9' 'ivdd_10'};
config.bracketing.modo='automatico';
    %modo 'manual'
        config.bracketing.limites={[1 100] [101 180]}; %definir como ...
            cell array vacío o no definir si no se quiere hacer ...
            bracketing
    %modo 'automático'
        config.bracketing.intervalos=10;

%% Generación de variables necesarias a partir de la configuración

%Bracketing
bracketing=0;
if isfield(config, 'bracketing')
    if isfield(config.bracketing, 'modo')
        if strcmpi(config.bracketing.modo,'manual')
                if isfield(config.bracketing, 'limites')
                    zonasbracketing=length(config.bracketing.limites);
                    limitesbracketing=config.bracketing.limites;
                    bracketing=1;
                else
                    error('Error: no se han definido los límites para ...
                        el modo manual de bracketing.')
                end
        elseif strcmpi(config.bracketing.modo,'automatico')
            if isfield(config.bracketing, 'intervalos')
                zonasbracketing=config.bracketing.intervalos;
                bracketing=1;
            else
                error('Error: no se ha definido el número de ...
                    intervalos para el modo automático de bracketing.'...
                    )
            end
        else
            error('Error: modo de bracketing no soportado. Modos ...
                soportados: -manual- y -automático-.')
        end
    else
        warning('Advertencia: no se ha especificado ningún modo de ...
            bracketing, aunque se ha definido parte de su configuració...
            n. No se realizará bracketing.')
        clear config.bracketing %no se ha especificado modo, ...
            desactivar bracketing
    end
end

%% Leer trazas
textprogressbar('Leyendo trazas: ');
j=0;
for h=1:length(config.trazas.nombre)
    [ trazasleidas ] = leer_sbox9( [config.trazas.ubicacion ...
        config.trazas.nombre{h}] );
```

```
48      %Si se quiere procesar AES sustituir por leer_AES
49
50      [l,~]=size(trazasleidas);
51      for i=1:l
52          j=j+1;
53          %Cada traza va a ser una estructura en un cell array.
54          traza{j}.contenido=trazasleidas(i,:);
55          traza{j}.archivofuente=config.trazas.nombre{h};
56          traza{j}.clave=h;
57          traza{j}.compresion='ninguna';
58      end
59      textprogressbar(h/length(config.trazas.nombre)*100)
60  end
61  clear trazasleidas l h i j
62  textprogressbar('hecho');
63
64  fprintf('Salvando trazas leídas en %s... \n', [config.trazas.ubicacion...
        'Trazas.mat'])
65  save([config.trazas.ubicacion 'Trazas.mat'],'traza')
66
67  %% Comprimir trazas
68  textprogressbar('Comprimiendo trazas: ');
69  warningbracketing=0;
70
71  %Preasignación de espacio
72  trazacomprimida_rawintegration=cell(length(traza),1);
73  trazacomprimida_sumofabsolutevalues=cell(length(traza),1);
74  trazacomprimida_sumofsquares=cell(length(traza),1);
75  trazacomprimida_rawintegration_bracketing=cell(length(traza),...
        zonasbracketing);
76  trazacomprimida_sumofabsolutevalues_bracketing=cell(length(traza),...
        zonasbracketing);
77  trazacomprimida_sumofsquares_bracketing=cell(length(traza),...
        zonasbracketing);
78
79  for i=1:length(traza)
80      %No bracketing
81      %   - Raw integration
82      trazacomprimida_rawintegration{i,1}=comprimir( traza{i}, '...
            rawintegration' );
83      %   - Sum of absolute values
84      trazacomprimida_sumofabsolutevalues{i,1}=comprimir( traza{i}, '...
            sumofabsolutevalues' );
85      %   - Sum of squares
86      trazacomprimida_sumofsquares{i,1}=comprimir( traza{i}, '...
            sumofsquares' );
87
88
89      %Bracketing
90      if bracketing==1
91          if strcmpi(config.bracketing.modo,'automatico')
92              inicio=1;
93              fix=0;
94              if (mod(length(traza{i}.contenido),zonasbracketing) ~= 0)
95                  fix=1;
96                  if warningbracketing == 0
97                  warning('Advertencia: No se puede dividir este tipo de...
                        traza en el número de intervalos especificado por...
```

```
                    falta de puntos. Se restará uno al número de ...
                    intervalos.')
98                  warningbracketing=1;
99                  end
100             end
101             for k=1:(zonasbracketing-fix)
102                 limitesbracketing{k}=[inicio inicio+floor(length(traza...
                        {i}.contenido)/zonasbracketing)-1];
103                 inicio=limitesbracketing{k}(2)+1;
104             end
105
106         end
107
108
109         for k=1:zonasbracketing
110             opciones.limites=limitesbracketing{k};
111             trazacomprimida_rawintegration_bracketing{i,k}=comprimir( ...
                    traza{i}, 'rawintegration' , opciones);
112             trazacomprimida_sumofabsolutevalues_bracketing{i,k}=...
                    comprimir( traza{i}, 'sumofabsolutevalues', opciones);
113             trazacomprimida_sumofsquares_bracketing{i,k}=comprimir( ...
                    traza{i}, 'sumofsquares', opciones);
114         end
115     end
116
117     textprogressbar(i/length(traza)*100)
118 end
119 %clear limitesbracketing
120 clear inicio fix warningbracketing
121 clear i opciones
122 textprogressbar('hecho');
123
124 fprintf('Salvando trazas comprimidas en %s... \n', [...
        config.trazas.ubicacion])
125
126 [...]
127
128 %% Exportar en formato de ataque
129 fprintf('Exportando trazas en formato de ataque en %s... \n', [...
        config.trazas.ubicacion])
130
131 [...]
132
133 clear T h limitestring
134
135 %% Limpieza final
136
137 clear limitesbracketing j
138 clear traza
139 clear trazacomprimida_rawintegration ...
        trazacomprimida_sumofabsolutevalues trazacomprimida_sumofsquares ...
        trazacomprimida_rawintegration_bracketing ...
        trazacomprimida_sumofabsolutevalues_bracketing ...
        trazacomprimida_sumofsquares_bracketing zonasbracketing bracketing...
         opciones k
```