eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería de Computadores

Trabajo de Fin de Grado

Utilización de algoritmos de estimación de distribuciones para optimizar ataques de canal lateral

Autor

Asier Insausti Carmona

2023

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería de Computadores

Trabajo de Fin de Grado

Utilización de algoritmos de estimación de distribuciones para optimizar ataques de canal lateral

Autor

Asier Insausti Carmona

Director

Jose A. Pascual

Resumen

El Análisis de Canal Lateral es un conjunto de ataques que sirven para obtener las claves de cifrado de dispositivos embebidos. Dichos ataques son complejos de llevar a cabo, por lo que existe una gran posibilidad de cometer errores, además de necesitar la presencia humana para su correcto ajuste y funcionamiento. Una posible solución para minimizar esta dependencia es el uso de Algoritmos de Estimación de Distribuciones. En el presente proyecto, en colaboración con el centro tecnológico de Ikerlan, se propone crear una herramienta que implemente este tipo de algoritmo para realizar ataques de Canal Lateral, y con ello, evaluar dispositivos y comprobar su protección ante los ataques efectuados sin requerir de la presencia humana.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

La ciberseguridad es el área que engloba las diferentes prácticas y técnicas que se centran en proteger los sistemas informáticos de diversos ataques maliciosos, también conocidos como ciberataques. En los últimos años, la ciberseguridad se ha convertido en un tema muy recurrente en las noticias. Y es que se ha incrementado en gran medida el número de ciberataques en todo el mundo. Esto deja en evidencia de que es un campo muy importante en la era digital, ya que cada día más y más dispositivos comienzan a necesitar de sistemas informáticos para su funcionamiento, como por ejemplo, ascensores, electrodomésticos, contadores de luz y agua.

Actualmente, existen varios tipos de ciberataques, donde varía el objetivo del mismo. Existen ataques que buscan obtener información confidencial, como por ejemplo los ataques de tipo Phishing, que buscan engañar a la víctima usando la ingeniería social, por ejemplo, correos fraudulentos, SMS con enlaces a páginas no oficiales, como entidades bancarias o redes sociales. También existen los ataques que intentan obtener la contraseña, como los ataques de fuerza bruta, que prueban muchas combinaciones de clave para intentar obtenerla. Y luego están los ataques que son más comunes y que han ganado más relevancia, que son los ataques de denegación de servicio (Denegation of Service, DoS) y los ataques de malware.

Además de estos ataques, existen otro tipo de ataques menos conocidos, pero que también pueden dejar entredicho la seguridad de algunos dispositivos. Estos ataques son los ataques físicos, que son capaces de modificar u obtener información confidencial de un dispositivo. Entre estos ataques, podemos encontrar los ataques de canal lateral (Side

Channel Attack, SCA), que son ataques que buscan obtener información confidencial observando las propiedades físicas del dispositivo. Estas propiedades físicas pueden ser el consumo de energía, variaciones en el tiempo de ejecución al verificar una clave o la temperatura del dispositivo, entre otros.

Existen diversas medidas para mitigar estos ataques, y es por ello que en los laboratorios de evaluación suelen replicarse estos ataques para comprobar el nivel de seguridad de los dispositivos. El principal problema de estas evaluaciones está en que realizar correctamente un ataque de canal lateral es un proceso complejo y propenso a errores, ya que el éxito o fracaso de los ataques tiene una fuerte dependencia humana. En los últimos años, se han usado técnicas de aprendizaje profundo (Deep Learning, DL) para tratar de mitigar esto. Por contra, aplicar estas técnicas en el contexto de los ataques de canal lateral no es algo sencillo, y aunque algunas problemáticas de los ataques clásicos desaparecen, aparecen nuevas complicaciones relacionadas con el DL. Por otro lado, recientemente ha surgido una alternativa que promete mitigar la dependencia humana sin a penas añadir complejidad extra: los algoritmos de estimación de la distribución (Estimation of Distribution Algorithms, EDAs).

Es por eso que el centro tecnológico de Ikerlan quiere desarrollar una herramienta para realizar ataques SCA de forma automática para evaluar la seguridad de ciertos dispositivos, usando los algoritmos de estimación de distribución para la misma.

Para conseguir esta herramienta, se va a usar una herramienta sobre SCA previamente implementada en Python, que se encuentra publicada en GitHub [Bubberman et al., 2020]. Con dicha herramienta, y con un ejemplo de EDA ya implementado facilitado por Ikerlan, se va a desarrollar la herramienta que se usara para evaluar los dispositivos. Además, también sé va a desarrollar una versión paralela de la herramienta, para reducir los tiempos de espera de las pruebas a realizar.

Esta herramienta debe tener diferentes opciones a la hora de hacer la evaluación, ya que tanto los parámetros del ataque como los del EDA tienen que poder cambiarse en función del análisis que se desea hacer. Entre estos parámetros se encuentran el dataset a utilizar, el número máximo de iteraciones del EDA y el tamaño de la población del EDA.

Una vez terminado el desarrollo de la herramienta, se van a realizar las diferentes pruebas. Estas pruebas van a ser de dos tipos, unas van a estar destinadas a comparar tiempos de ejecución en función de los parámetros que se usan, y las otras para comparar como afectan los parámetros a la obtención de la clave del dispositivo a analizar, viendo así si es posible obtener la clave del dispositivo.

Por lo tanto, la memoria está estructurada de la siguiente manera. Primero, se detallan los diferentes objetivos que el trabajo debe cumplir. Después, se realiza un estudio sobre el estado del arte para obtener una base previa. A continuación, se explica la solución desarrollada con sus detalles y como funciona. Asimismo, se realizan y se analizan las diferentes pruebas realizadas con la herramienta, además de sus respectivas conclusiones. Por último, se expondrán las conclusiones del trabajo y posibles líneas de futuro trabajo.

2. CAPÍTULO

Documento de objetivos del proyecto

Como ya se ha mencionado antes, el objetivo principal del proyecto es conseguir desarrollar una herramienta para realizar ataques SCA para así poder evaluar la seguridad de ciertos dispositivos. Desde Ikerlan se han facilitado las 2 herramientas previamente implementadas, por lo que la idea es partir de esa base para desarrollar la nueva herramienta. Esto es porque desarrollar una aplicación basada en dichas herramientas no requiere de grandes cambios, y con ello podemos ahorrar una parte del trabajo y poder destinarla a desarrollar las funcionalidades que no vienen implementadas.

Para empezar con el proyecto, primero se va a realizar un estudio del estado del arte, para conocer los diferentes métodos de SCA y posibles técnicas que puedan mejorar el desempeño de las mismas. Además, también se incluirá en dicho estudio el estado del arte en cuanto a la ciberseguridad en dispositivos ciberseguros, ya que son los objetivos de los análisis que se van a realizar con la herramienta a desarrollar.

Después, se va a analizar el código implementado de las 2 herramientas dadas, y ver como integrar ambas herramientas con el código necesario. Esto es porque la herramienta de SCA incluye opciones que no serán necesarias para el desarrollo de la herramienta final, pero es posible que alguna de las funcionalidades pueda servir de ayuda para el análisis de los dispositivos, por lo que habrá que estudiar si conviene incluir alguna de estas funcionalidades. Además de esto, también se tendrá que desarrollar el código en su versión paralela, ya que esto ayudara a reducir el tiempo de espera de las pruebas.

Una vez se tenga esta de la herramienta, se tendrá que implementar las funcionalidades añadidas. Están son las aquellas funcionalidades que permitirán realizar pruebas de ren-

dimiento y de análisis cuando sean necesarias, además de funcionalidades que puedan ayudar a mejorar el análisis. Junto a estas funcionalidades, también se va a añadir la posibilidad de configurar los diferentes parámetros necesarios para los análisis, con el fin de facilitar las diferentes pruebas que se quieran poder hacer.

Por último, se va a realizar un análisis experimental, dividido en 2 partes. La primera constará de pruebas de rendimiento, donde se analizara como afectan diferentes configuraciones a los tiempos de los análisis y también se comparara el tiempo de la ejecución de los análisis en serie y en paralelo. Y la segunda constará de pruebas enfocadas al análisis de los dispositivos, donde se harán diferentes pruebas para ver si son vulnerables a los ataques SCA, y por consiguiente, si es posible obtener la clave criptográfica del dispositivo.

En resumen, las tareas a desarrollar en el proyecto son las siguientes:

- Estudiar el estado del arte sobre SCA y ciberseguridad en dispositivos ciberseguros.
- Desarrollar herramienta partiendo de las herramientas existentes.
- Añadir funcionalidades necesarias a la herramienta.
- Realizar un análisis experimental con pruebas de rendimiento y análisis de seguridad de dispositivos.

3. CAPÍTULO

Estado del arte

3.1. Criptografía

La criptografía es el ámbito donde se usan técnicas de codificación para alterar los textos o mensajes y hacerlos imposibles de leer para receptores no autorizados. El objetivo principal es obtener la confidencialidad del mensaje, de forma que solo el destinatario pueda ser capaz de leerlo.

Para esta tarea, se necesita de un sistema para modificar el mensaje original. Ese sistema puede ser de varias formas, y durante la historia ha habido diferentes métodos. El más sencillo es desplazar las letras del abecedario en un número, por ejemplo. si tenemos la letra a y un desplazamiento de 5, obtenemos la letra f. En la actualidad, los sistemas que se usan son algoritmos criptográficos, donde tenemos 2 entradas y 1 salida. Como entradas, encontramos el texto a cifrar y la llave para cifrar, y como salida, encontramos el texto ya cifrado. Estos algoritmos basan el cifrado en realizar cálculos matemáticos con ambas entradas, obteniendo así la información cifrada. Además, estos algoritmos son conmutativos, es decir, el proceso se puede repetir a la inversa, usando la información cifrada y la clave para recuperar la información original.

Dentro de los diferentes algoritmos que podemos encontrar, se pueden distinguir 2 tipos.

- *Criptografía simétrica*: estos algoritmos usan una clave secreta compartida para el intercambio de mensajes, y requieren 2 fases. La primera fase trata de compartir la clave de forma segura entre las 2 partes de la comunicación, y la segunda fase es

donde ambas partes se pueden comunicar con el envío de mensajes cifrados. Dentro de esta categoría, encontramos los algoritmos DES, que ya es considerado inseguro, y también encontramos AES, el estándar actual.

- *Criptografía asimétrica*: estos algoritmos no usan clave compartida, sino que cada parte de la comunicación tiene 2 claves: una clave pública y una clave privada. La clave pública no es necesaria que se comparta de forma segura con la otra parte de la comunicación, por lo que se evita esa fase previa a la comunicación. Dentro de esta categoría, encontramos el algoritmo RSA, que usa claves de tamaño 1024 bits o más, por lo que se convierte en un algoritmo seguro por su alto coste computacional para obtener la clave con fuerza bruta.

Comparando ambos tipos de algoritmos, cada uno tiene sus ventajas respecto a la otra. Por parte de los algoritmos simétricos, son más rápidos con la misma longitud de clave y mensaje, y para conseguir el mismo nivel de seguridad las claves necesitan menor longitud. Y por parte de los asimétricos, obtenemos mayor seguridad al usar 2 claves, y también ahorramos el problema de intercambiar la única clave de forma segura. Es por eso mismo que normalmente los algoritmos asimétricos se usan para intercambiar las claves de sesión, y los simétricos para el intercambio de información.

3.1.1. AES

El estándar de cifrado avanzado (estándar de cifrado avanzado, AES, por sus siglas en inglés) es un algoritmo de clase simétrica. Desde el año 2001 es el estándar de ciberseguridad. Usa un esquema de cifrado por bloques con longitud fija de 128 bits, y opera con matrices de tamaño $4 * 4$ bytes. Existen 3 variantes del algoritmo, donde varía la longitud de la clave y el número de rondas. La variante con la clave más pequeña es de 128 bits y 10 rondas, la variante intermedia con clave de 192 bits y 12 rondas, y la más grande con 256 bits y 14 rondas. Actualmente, cualquiera de estas opciones se considera segura, pero en ciertos dispositivos donde los recursos son limitados, se siguen usando las variantes con menor longitud de clave. Aun así, no es un algoritmo que necesite de muchos recursos, ya que es rápido tanto en implementación software como hardware, además de que es sencillo de implementar.

A día de hoy, AES es considerado seguro desde el punto de vista criptográfico, es decir, no existe ataque que requiera menor coste computacional para obtener la clave que hacerlo mediante una búsqueda exhaustiva (fuerza bruta). Aunque no sea posible romper AES con

los recursos actuales (potencia de cómputo), sí que se han encontrado vulnerabilidades o sospechas de que es probable que en un futuro AES sea inseguro. Por ejemplo, se ha conseguido atacar AES utilizando un menor número de rondas, reduciendo el tiempo del ataque considerablemente. En cualquier caso, aunque AES sea criptográficamente seguro, la implementación física de este puede ser susceptible a otro tipo de ataques físicos, como los que se estudian en el presente trabajo.

Detalles del algoritmo AES

Como ya se ha mencionado anteriormente, AES es un algoritmo de cifrado por bloques. El cifrado por bloques está basado en algoritmos iterativos, y para cada iteración requiere una subclave, que se obtiene de la clave original. El cifrado por bloques en AES opera con bloques de 128 bits, que se copia en una matriz de tamaño $4 * 4$ bytes antes de pasar a la función ronda. Esta matriz se le denomina *State*, y es la donde se realizan las operaciones del algoritmo.

Existen 3 configuraciones posibles de clave, donde varía la longitud de clave, y en consecuencia el número de veces que se ejecuta la función ronda. Estas son las 3 posibilidades:

- Clave de 128 bits, 10 rondas.
- Clave de 192 bits, 12 rondas.
- Clave de 256 bits, 14 rondas.

Cuando se usa una clave de mayor longitud, aumentamos el nivel de seguridad, a costa de aumentar el tiempo de cifrado y descifrado. Una vez sabemos la longitud de la clave, se generan las subclaves necesarias para el algoritmo, con el mismo tamaño que el *State*. Estas subclaves se generan con una función llamada *Key Expansion*, y se genera un número de subclaves igual al número de rondas + 1, ya que se necesita una subclave al inicio del proceso y una subclave por cada ronda.

El algoritmo de cifrado y descifrado consiste en ejecutar una función llamada ronda un número de veces, en función de la longitud de la clave. Esta función consta de 4 operaciones:

- AddRoundKey: Esta operación realiza una operación XOR entre la matriz *State* y la subclave. El resultado es una nueva matriz *State* actualizada.

- SubBytes: Esta operación es una sustitución no lineal que opera de manera independiente para cada byte de la matriz *State*. Se utiliza una tabla de sustitución llamada *Sbox*, que es invertible. Esta tabla se suele representar en hexadecimal, y la sustitución se realiza en cada byte del *State*, usando su valor para saber por qué valor hacer la sustitución (ver Figura 3.1).

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

Figura 3.1: Tabla S-box con los valores de sustitución para cada byte (en hexadecimal) [Evans and Brown, 2001]

- ShiftRows: Esta operación realiza un desplazamiento de las 3 últimas filas de la matriz *State* hacia la izquierda. La segunda fila se desplaza 1 byte, la tercera fila 2 bytes y la última fila 3 bytes.

- MixColumns: Esta operación realiza una transformación lineal a cada columna de la matriz *State*. Es una operación matemáticamente compleja y computacionalmente costosa en comparación con las otras operaciones. Cada columna se multiplica por una matriz predeterminada, usando la columna original para obtener la nueva.

Para el proceso de descifrado, existen las versiones inversas a estas operaciones: InvShiftRows, InvSubBytes, InvMixColumns y AddRoundKey, que de por sí ya es inversa al ser una operación XOR. La diferencia respecto al proceso de cifrar es el orden de SubBytes y ShiftRows, que se permuta. Otro cambio es que la tabla S-Box es inversa a la original.

AES trabaja con bloques de 128 bits, pero hay casos donde tenemos mensajes para cifrar con mayor longitud. Para ello, existen varias configuraciones posibles a la hora de

implementar AES, donde varían diferentes aspectos. Las configuraciones posibles son las siguientes:

- ECB: La configuración Electronic Codebook Mode (ECB por sus siglas en inglés) es el tipo de configuración más simple. Como podemos observar en la Figura 3.2, el mensaje se divide en bloques, y cada bloque se cifra de forma separada. Esto permite que se pueda computar de forma paralela, y, por tanto, de una forma rápida, y tampoco existe una propagación de errores. Por contra, no es muy seguro, ya que el mismo bloque de entrada siempre obtiene el mismo cifrado.

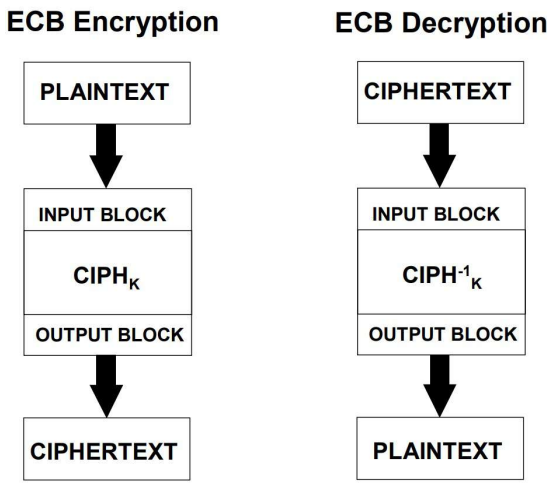

Figura 3.2: Modo de operación ECB [Dworkin, 2001]

- CBC: La configuración Cipher Block Channing (CBC, por sus siglas en inglés) usa un vector de inicialización (IV por sus siglas en inglés) para hacer cada cifrado único. En la Figura 3.3 se puede ver como a cada bloque de texto de entrada se le realiza la operación XOR con el bloque previo (o el vector de inicialización si es el primer bloque). Gracias a esto, cada bloque depende de todos los bloques de texto usados hasta el momento. Es muy importante que el vector IV sea aleatorio e impredecible, ya que en caso contrario puede ser inseguro ([OpenSSL Foundation, 2023]). Como ventaja, obtenemos un cifrado diferente para el mismo bloque de entrada, lo que aumenta la seguridad. Por contra, es más lento que ECB, si tiene propagación de errores, y no permite computación paralela al cifrar, pero sí al descifrar.

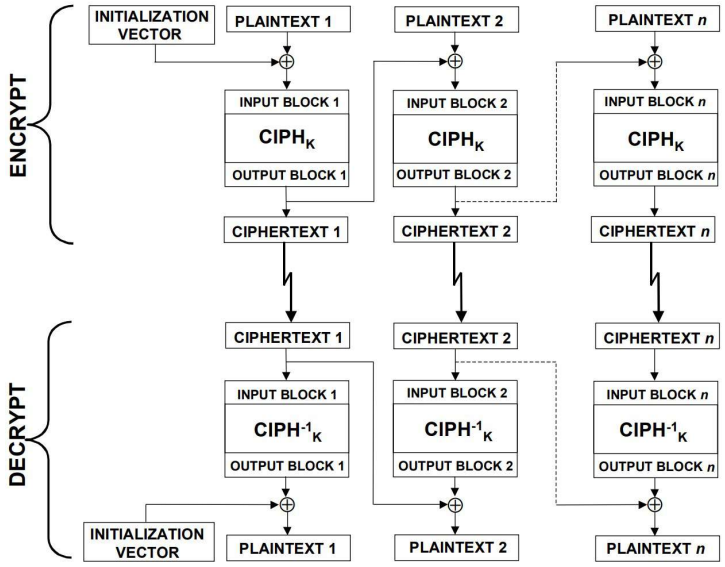

Figura 3.3: Modo de operación CBC [Dworkin, 2001]

- CFB: La configuración Cipher Feedback Mode (CFB, por sus siglas en inglés) no opera con el mismo tamaño de bloque, sino que opera con unidades más pequeñas. También requiere del vector IV. En la Figura 3.4 podemos ver que la idea es usar un registro de desplazamiento, donde primero se carga el IV, y se cifra con la clave. Después, se seleccionan los primeros s bits (los s bits más a la izquierda) y se hace la operación XOR con el segmento de tamaño s del texto de entrada. Finalmente, en el registro de desplazamiento quedan los bits restantes que no se han desplazado $(n-s)$ y los del resultado de la operación XOR. Este proceso se repite hasta obtener todos los fragmentos de texto cifrados. Esta configuración convierte el cifrado por bloques en una especie de cifrado por flujo, obteniendo así las propiedades de ambos tipos. Como desventaja, también tiene propagación de errores, ya que el cálculo de un fragmento depende de los anteriores. Además, por la misma razón no permite computación paralela al cifrar, pero sí al descifrar.

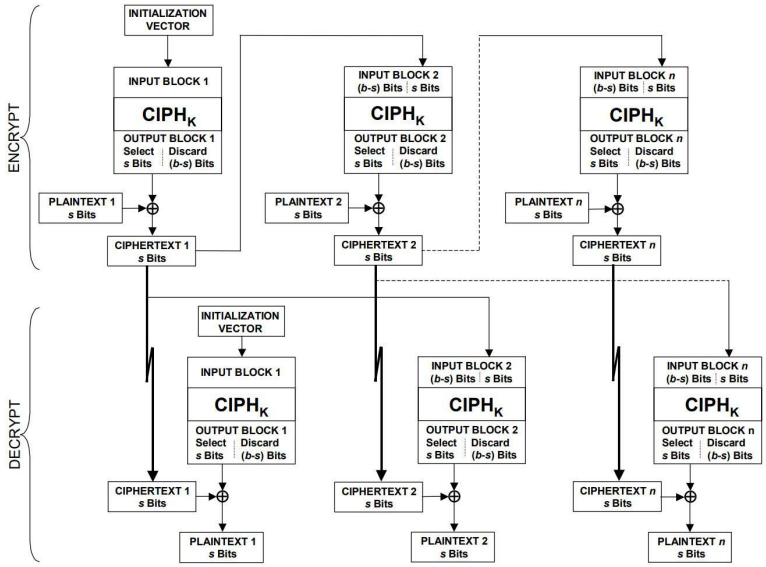

Figura 3.4: Modo de operación CFB [Dworkin, 2001]

- OFB: La configuración Output Feedback (u OFB) utiliza un sistema parecido al CFB, que lo podemos ver en la Figura 3.5. Usa el registro de desplazamiento (con el IV cargado al inicio) del tamaño de los segmentos de texto de entrada. Una vez encriptada, se carga la salida de la encriptación en el registro, y se realiza la operación XOR con el segmento del texto de entrada para obtener el segmento cifrado. El IV debe ser de un único uso, ya que de no ser así se puede obtener el segmento cifrado de un segmento de texto.

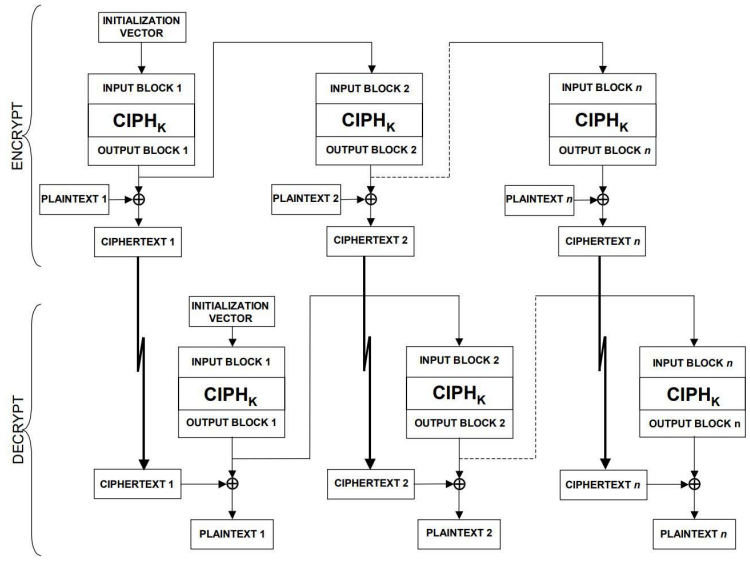

Figura 3.5: Modo de operación OFB [Dworkin, 2001]

- CTR: La configuración Counter Mode (o CTR) es considerado como la versión CFB basada en contadores. En la Figura 3.6 vemos que se usa un bloque de contador por cada bloque de texto de entrada, se encripta el bloque contador con la clave, y a dicha salida realizarle la operación XOR con el bloque de texto de entrada. Después de esto, se incrementa el contador en 1. En caso de que el último bloque sea parcial, solo se usaran los bits más significantes (los primeros) para realizar la operación XOR con el último bloque.

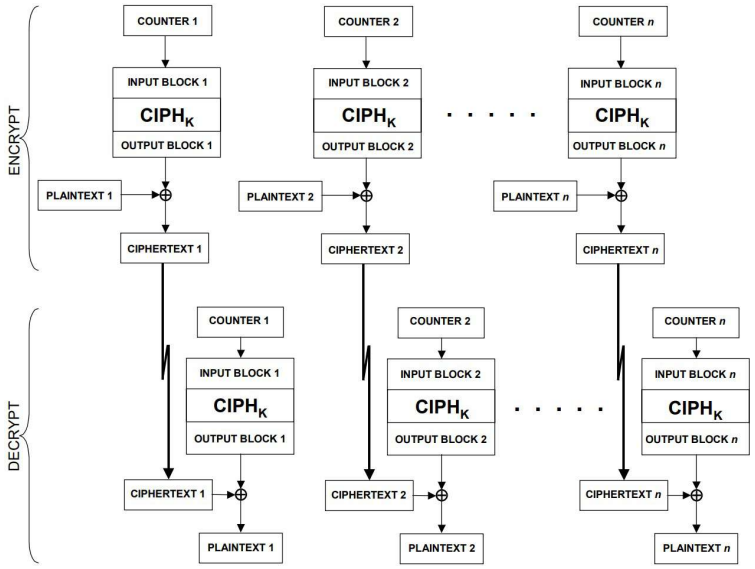

Figura 3.6: Modo de operación CTR [Dworkin, 2001]

3.2. Ataques en dispositivos criptográficos

Cuando hablamos de ataques a dispositivos criptográficos, la idea es obtener la clave criptográfica en dicho dispositivo. Estos dispositivos suelen ser sistemas embebidos de diversos tipos, donde encontramos "smart cards" como las tarjetas de crédito, las carteras de criptomonedas, o las licencias que se encuentran en los "token USB". Para ello, existen diferentes tipos de ataques, que se pueden clasificar. Para este trabajo, me centraré en los ataques físicos, ya que es un tipo de ataque que estos dispositivos no suelen proteger.

Los ataques físicos son aquellos ataques que buscan aprovechar las propiedades físicas de un dispositivo con el fin de obtener información (como la clave criptográfica) o alterar el funcionamiento del mismo (generar errores). Los objetivos de estos ataques pueden ser dispositivos criptográficos, sistemas de autenticación o sistemas de pago, entre otros. Existen varios tipos de ataques, que dependen de las características del ataque.

3.2.1. Taxonomía de ataques físicos

Un criterio para clasificar los ataques físicos es diferenciar entre ataques activos y pasivos.

- Ataque activo: En este tipo de ataque, el dispositivo, su entorno o sus entradas son manipulados para conseguir comportamientos o estados alterados en el dispositivo. La idea es obtener la clave durante esos estados alterados.
- Ataque pasivo: En este tipo de ataque, el atacante no altera el dispositivo de ninguna forma. La idea es obtener la clave observando propiedades físicas del dispositivo. Estos son el consumo de energía, la emisión de campos electromagnéticos y medición de tiempo, entre otros.

Otro criterio para clasificar los ataques es en función de la interfaz que se explota en el ataque. Hay interfaces que tienen acceso sencillo, mientras que otros requieren de equipo especializado para ello. Para ello, se distingue entre ataques invasivos, semi-invasivos y no invasivos, y todos pueden ser activos o pasivos.

- Ataques invasivos: Este tipo de ataques es el más potente que se puede realizar a dispositivos criptográficos. Requiere desmontar el dispositivo y usar el banco de pruebas para ello. Dependiendo de las señales que queremos observar o manipular, el ataque es pasivo o pasivo. Como contrapartida, este tipo de ataque requiere de equipo más caro y de más tiempo. Además, es posible dañar el dispositivo.
- Ataques semi-invasivos: Este tipo de ataque también necesita desmontar el dispositivo, pero no necesitan tener contacto con el chip del dispositivo. Estos ataques no requieren de un equipo tan caro, pero necesitan más tiempo y más conocimiento sobre el dispositivo a atacar. Además, también pueden dañar el dispositivo.
    - Ataque semi-invasivo pasivo: El objetivo es conseguir información sensible observando las celdas de memoria.
    - Ataque semi-invasivo activo: El objetivo es provocar fallos en el dispositivo con diferentes técnicas, como rayos X o láser. También se incluye la inyección de faltas semi-invasivo.
- Ataques no invasivos: En estos ataques, el dispositivo no es modificado ni alterado. La idea es analizar las interfaces directamente accesibles, sin dejar pruebas del ataque. No requieren de equipamiento muy caro. Los más conocidos y ampliamente

estudiados son los ataques no invasivos pasivos, también conocidos como Análisis de Canal Lateral. Además de estos, también existen los ataques no invasivos activos, también conocidos como Ataque de Inyección de Faltas (Non Invasive Fault Injection Attacks).

3.3. Análisis de Canal Lateral

El Análisis de Canal Lateral son un conjunto de ataques potentes que pueden saltarse las protecciones, software y hardware implementados en el dispositivo. Estos ataques se basan en obtener información sensible mediante observación de propiedades físicas de los dispositivos, y ver la fuga de información sensible que existe en dichas propiedades. Se clasifican en función de la propiedad física que explotan:

- Tiempo [Kocher, 1996]
- Consumo de energía [Kocher et al., 1999]
- Energía electromagnética [Quisquater and Samyde, 2001]
- Sonido [Backes et al., 2010]
- Óptico [Schlösser et al., 2012]
- Temperatura [Hutter and Schmidt, 2014]

Entre estas propiedades físicas, las más importantes son el de tiempo, de potencia y el de electromagnético.

Existen varios tipos de técnicas de Análisis de Canal Lateral. Estas técnicas se distinguen en 2 grandes clases: los no perfilados, y los perfilados.

3.4. Análisis de Canal Lateral No Perfilado

El Análisis de Canal Lateral No Perfilado utilizan solo la información obtenida por el canal lateral observado para conseguir la información del dispositivo. Se utiliza cuando no es posible hacer el perfilado del dispositivo, ya que hay veces donde no lo permite el dispositivo (dispositivos cerrados como las “smart cards”), bien porque suelen estar limitados en número de acciones o por no tener acceso a la clave secreta [Timon, 2019].

3.4.1. Análisis de Potencia Simple

Cuando un dispositivo electrónico tiene implementado un algoritmo criptográfico, realiza diferentes operaciones con diferentes costes computacionales. Esto se puede apreciar analizando datos (como por ejemplo los algoritmos DES y AES) y de esta forma obtener información relevante. Se analizan dos dependencias cuando se realiza un análisis de consumo:

- Dependencia operacional: existe dependencia entre el consumo de energía y la operación que se realiza por el dispositivo electrónico.
- Dependencia de datos: existe dependencia entre el consumo de energía y los datos que se están procesando por el dispositivo electrónico.

Cuando queremos explotar dichas dependencias, se utilizan las trazas de consumo, que son la medición de energía consumida mientras el dispositivo está ejecutando el algoritmo criptográfico. Esto se hace mediante el uso de un osciloscopio, que permite detectar pequeños cambios en el consumo de energía. Normalmente, se suelen necesitar varias trazas para el ataque, pero también podemos usar una única traza, aunque solo obtenemos dependencia operacional. A esto se le llama Análisis de Potencia Simple. Aunque pueda parecer algo muy sencillo, ya que solo se usa una sola traza, realmente es más complicado, ya que requiere de altos conocimientos sobre el dispositivo a atacar, y eso no siempre es posible. Por ello, no es ampliamente usado.

3.4.2. Análisis de Potencia Diferencial

Los análisis de potencia suelen necesitar más de una traza para que sean exitosos. Esto es porque generalmente se desconoce el funcionamiento interno del dispositivo. A este tipo de ataque se le denomina Análisis de Potencia Diferencial. A diferencia de un análisis de potencia simple, no necesita saber gran detalle del dispositivo a atacar, solo hace falta saber qué algoritmo de inscripción está implementado.

Para realizar el ataque, se realiza un análisis estadístico sobre múltiples trazas con el consumo de energía del dispositivo, para así vincular los valores intermedios del algoritmo. Para eso, se utiliza un modelo de fugas. Existen varios tipos.

- Hamming Weight (HW, por sus siglas en inglés): corresponde a contar el número de bits a 1. Por ejemplo, el valor Hamming Weight de 40 sería: $HW(40_{10}) = HW(00101000_2) = 2$.

- Hamming Distance (HD, por sus siglas en inglés): corresponde al número de bits cambiados de 0 a 1 y de 1 a 0 entre un estado v_0 y v_1. Por ejemplo, $HD(25_{10}, 30_{10}) = HD(00011001_2, 00011110_2) = 3$.

- Indentity Model (ID, por sus siglas en inglés): corresponde a usar el mismo valor que el valor intermedio dado. Por ejemplo, $ID(40_{10}) = 40$.

Una vez escogido el modelo que se quiere usar, se realiza el análisis estadístico entre las trazas de consumo y el modelo para cada posible valor. Siendo el valor calculado v= $f(p,k)$, donde f es el modelo seleccionado, p es el texto de entrada y k es la clave, se calcula para cada valor posible de k un valor intermedio y aplica el modelo, y después comparar el consumo real con el hipotético. La diferencia de las medias entre el consumo hipotético y las trazas se usa para encontrar la clave correcta. Esto es porque el valor intermedio, v, depende de del texto de entrada (que previamente conocemos) y la clave, y si adivinamos el valor intermedio, podemos recuperar la clave. En caso de que todos los valores del consumo hipotético sean parecidos, puede significar 2 cosas: o el dispositivo tiene implementados contra medidas para enmascarar la dependencia, o no hemos utilizado trazas suficientes.

Existe otra opción aparte de las diferencias de medias, que consiste en usar el coeficiente de correlación para determinar las relaciones lineales entre el consumo hipotético y el real. A esta variante se le llama Análisis de Potencia de Correlación, y en la práctica ofrece mayor eficiencia y robustez que la diferencia de medias.

3.5. Análisis de Canal Lateral Perfilado

El Análisis de Canal Lateral Perfilado tiene como idea generar un modelo del consumo de energía del dispositivo para obtener la información sensible. Estos ataques constan de 2 fases: una fase de perfilado, donde se genera el modelo usando un gran número de trazas, y una fase de ataque, donde se aplica el modelo con pocas trazas para obtener la clave. Hay diferentes tipos según la técnica usada para la generación del modelo: Clasificación Gaussiana (también denominado Template Attack o TA), Machine Learning

(Support Vector Machine (SVM), Random Forest (RF), Deep Learning (DL)) y Modelos Estocásticos. Las más populares son TA y ML.

3.5.1. Template Attack

El Template Attack se basa en crear un modelo multivariante de la distribución de probabilidad de la fuga. Se calcula la función de densidad de probabilidad (PDF) asumiendo que la fuga sigue una distribución Gaussiana. Es una estimación paramétrica, converge rápido y es ampliamente usado.

El objetivo principal es obtener la clave secreta de un dispositivo que realiza operaciones criptográficas. Para ello, hay que medir alguna de las propiedades físicas del dispositivo durante el cálculo de un valor intermedio relacionado con la clave y el texto de entrada.

En la fase de perfilado, se usa un conjunto de trazas (profiling traces) para generar el modelo multivariante de Gauss, que se define como un vector de medias y una matriz de covarianza por cada valor intermedio posible, que es lo que comúnmente se denomina plantilla.

En la fase de ataque, se usan trazas de ataque (attack traces) para obtener la clave. El atacante genera valores intermedios posibles, que dependen de los datos de entrada y la clave hipotética, y además, cada hipótesis de clave necesita un template (vector de medias y matriz de covarianza) para cada valor de entrada. Después, se calcula una puntuación discriminante para cada clave hipotética y se ordenan en orden de probabilidad descendente. De esta forma, si el ataque es exitoso, la clave correcta será la que se encuentra en primera posición.

Selección de Puntos de Interés

Uno de los problemas de los TA es su complejidad computacional y su necesidad de reducción dimensional. Para solucionar dicho problema, se puede hacer una selección de puntos con la información relevante de la fuga. Para ello, se usan diferentes funciones, como por ejemplo, el coeficiente de correlación de Pearson, el error cuadrático medio (o su versión normalizada) y la relación señal/ruido. Una vez aplicada alguna de esas funciones, se escogen los puntos que mayor valor obtienen. También existen otras técnicas para reducir el número de muestras en cada traza de consumo, conocidos como métodos de compresión, donde se encuentra el análisis de componentes principales. Además, re-

cientemente se ha estudiado la posibilidad de usar técnicas de Machine Learning para la reducción o selección de puntos.

A continuación, tenemos la descripción de las diferentes técnicas de selección de puntos de interés:

- **Coeficiente de Correlación de Pearson**: Métrica estadística que estima la dependencia lineal entre 2 variables, donde sus valores están entre -1 y 1. Para la selección de puntos, se calcula con los valores intermedios procesados y las trazas del consumo del dispositivo. El cálculo lo podemos ver en la Ecuación 3.1:

$$P_{x,y} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}} \tag{3.1}$$

- **Error Cuadrático Medio**: Se divide el conjunto de trazas en varios grupos con los mismos valores intermedios asociados. Después, se calcula la media de cada grupo, y se calcula la diferencia al cuadrado de cada par de medias. El cálculo lo podemos ver en la Ecuación 3.2:

$$SOSD_{x,y} = \sum_{i,j>i} (\bar{x}_{y_i} - \bar{x}_{y_j})^2 \tag{3.2}$$

- **Error cuadrático medio normalizado**: Versión normalizada del Error Cuadrático medio, que es el equivalente de a la T de student. La diferencia es que se tiene en cuenta la varianza de las trazas de consumo. El cálculo lo podemos ver en la Ecuación 3.3:

$$SOST_{x,y} = \sum_{i,j>i} ((\bar{x}_{y_i} - \bar{x}_{y_j}) / \sqrt{\frac{\sigma_{y_i}^2}{n_{y_i}} + \frac{\sigma_{y_j}^2}{n_{y_j}}})^2 \tag{3.3}$$

- **Análisis de Componentes Principales**: Esta técnica se basa en calcular Componentes Principales, derivadas de la combinación lineal de las variables originales. Para ello, se usa la técnica de Descomposición en valores singulares de la matriz de trazas de consumo (trazas x muestras). A diferencia de los anteriores, no tiene en cuenta el valor intermedio asociado a cada traza de consumo. Además, también se puede usar contra implementaciones con máscara, donde las técnicas anteriores no obtienen resultados concluyentes.

3.6. Contra medidas

Para aumentar la seguridad de los sistemas ciberseguros, se han creado diferentes métodos de protección. El más sencillo es actualizar la clave criptográfica cada poco tiempo, denominados clave de sesión. De esta forma, el atacante no tiene tiempo para obtener suficientes trazas con cada clave. Pero hay dispositivos donde esto no es posible de realizar, como por ejemplo la llave de un coche, donde la clave siempre debe de ser la misma para poder abrirlo.

Existen varias otras formas de proteger los dispositivos. Una de ellas es proteger a nivel físico el dispositivo, ya que el Análisis de Canal Lateral requiere de acceso físico al dispositivo. Entre ellas, podemos encontrar mecanismos para reducir la fuga de información, como usar filtros de consumo y generadores de ruido aleatorio. También existen las contra medidas contra ataques electromagnéticos, donde se reduce la emisión de las mismas encapsulando el núcleo del dispositivo. Sin embargo, estas contra medidas pueden ser retiradas del dispositivo por el atacante, ya que normalmente se suelen poder quitar con relativa facilidad.

La otra opción para proteger los dispositivos es reducir la dependencia entre los valores intermedios ejecutados y el consumo del dispositivo. Existen dos formas de hacer esto: evitar que exista una dependencia entre los valores intermedios ejecutados y el consumo, o aleatorizar el valor intermedio que se está procesando. Podemos agrupar estas contra medidas en dos grandes grupos, ocultación y enmascaramiento

3.6.1. Ocultación

Estas contra medidas están basadas en evitar o reducir la dependencia entre los valores intermedios y el consumo del dispositivo que implementa el algoritmo criptográfico. Esto se hace aleatorizando la ejecución del algoritmo o alterando las características del consumo para que todas las operaciones tengan un consumo similar. Con esto se intenta que sea más difícil encontrar dependencia entre el consumo y la información secreta.

Para aleatorizar el consumo, se puede cambiar en que momento exacto se realiza una operación de cada iteración. Si el valor intermedio se procesa cada vez en un instante de tiempo diferente, es necesario alinear todas las trazas de potencia para aplicar con éxito el ataque. Para ello, se pueden insertar operaciones ficticias o se puede cambiar el orden en el que se realizan las operaciones. Aleatorizar el consumo tiene un impacto significativo

en el rendimiento del dispositivo, y no son una gran medida de protección, ya que el atacante puede obtener un gran número de trazas y calcular la media de las mismas.

Para que todas las operaciones tengan un consumo similar, se puede hacer tanto en la parte software como en la parte hardware. Por el lado software, la idea es seleccionar instrucciones con un consumo similar. Por el lado hardware, se suelen añadir filtros de potencia y/o generadores de ruido aleatorio al diseño. También existe la opción de usar celdas lógicas llamada Dual Rail Precharge, que utilizan dos conexiones para cada señal, de forma que se asegura el equilibrio en las cargas capacitivas [Wild et al., 2015, Tang et al., 2017].

3.6.2. Enmascaramiento

El enmascaramiento es una variante de contra medida basada en aleatorizar los valores intermedios de un algoritmo criptográfico. Con esta contra medida, no tenemos que modificar el consumo de energía directamente. Esta contra medida se puede implementar mediante software y mediante hardware.

Por el lado del software, la idea es añadir ciertas operaciones al algoritmo criptográfico. El cálculo de estas operaciones es de orden t, es decir, aplicar t máscaras a cada valor intermedio, aumentando así la seguridad de la implementación. Aun así, el sistema más usado es el de primer orden, es decir, $t = 1$, debido a la complejidad de usar un orden mayor.

Para enmascarar un valor intermedio con una máscara de orden 1, se calcula así: $v_m = v * m$, donde $*$ es una operación. Las operaciones para enmascarar son 3 posibles: XOR, multiplicación modular y suma modular. Estas operaciones se clasifican en 2 tipos de esquemas para enmascarar:

- Máscara booleana: usar operaciones XOR
- Mascara aritmética: usar la multiplicación modular o la suma modular

Para saber qué esquema usar, es muy importante conocer el algoritmo a implementar. Y es que en caso de que el algoritmo tenga operaciones no lineares, el esquema de máscara aritmética no es tan sencillo, haciendo que se necesite mucho tiempo para diseñar una solución. En el caso del algoritmo AES, por ejemplo, Las operaciones ShiftRows, MixColumns y AddRoundKey son lineales, pero SubBytes no. Otra opción es combinar ambos esquemas para según que tipo de operaciones.

Por el lado hardware, nos encontramos con la idea de usar estilos de lógica enmascarada. La idea consiste en modificar la lógica de las celdas a nivel físico. Esta tecnología necesita una línea más para cada conexión de la celda física, que representa el bit de máscara asociado a cada una. Un ejemplo son las Masked Dual-Rail Precharge (MDPL) [Popp and Mangard, 2006], que también usan líneas extra para balancear la carga de los condensadores, y también los Random Switching Logic (RSL) [Chen and Zhou, 2006]. Con esto, podemos obtener una mayor seguridad sin modificar el algoritmo ya implementado.

En ambos casos, tanto en la implementación software como hardware, en caso de que la implementación de estas sea errónea, puede llevar a que la seguridad no se vea aumentada.

3.7. Algoritmos de estimación de distribución

Los algoritmos de estimación de distribución (EDA) son técnicas de optimización estocásticas que buscan soluciones potenciales creando modelos probabilísticos de candidatos prometedores. Se han aplicado en diferentes disciplinas, por ejemplo:

- Plegado de proteínas [Santana et al., 2008]
- Programación de flujo de trabajo [yao Wang et al., 2013]

A diferencia de otros algoritmos evolutivos, la principal ventaja de los EDA es su simplicidad. Por un lado, tienen muchos menos parámetros que otros algoritmos evolutivos (por ejemplo, los algoritmos genéticos, GA), ya que la nueva población se genera de una distribución de probabilidad obtenida del mejor individuo. Por otro lado, con otros heurísticos (como GA) no solo tenemos en cuenta parámetros usuales del algoritmo evolutivo, sino que también necesitamos un diseño adecuado. Debido a esto, descartamos otros tipos de algoritmos evolutivos, ya que aumentan la complejidad en vez de reducir el proceso.

Para este trabajo, se va a introducir el uso de estos algoritmos para el Análisis de Canal Lateral.

3.7.1. Algoritmos de estimación de distribución en Análisis de Canal Lateral

En el ámbito del Análisis de Canal Lateral, se pueden combinar los EDA con los Template Attack para obtener los puntos de intereses, para posteriormente realizar la fase de perfilado y la fase de ataque. Gracias a usar el EDA, evitamos diferentes análisis manuales de diferentes combinaciones de POI. Como una búsqueda exhaustiva es exponencial, nuestra idea será una búsqueda basada en medida de calidad junto con un algoritmo computacional moderno y eficiente.

El proceso de un EDA sigue el siguiente esquema:

1. Se genera una población inicial D_0 de tamaño N individuos, que son los diferentes candidatos de Puntos de Interés. Cada uno de ellos consta de un vector binario de longitud T, donde T es el número de muestras en cada traza de consumo:

$$x = \{x_1, x_2, ..., x_T\} = \{0, 1, 1, 0, .., 0\}$$

   Cada variable de ese vector está relacionado con cada muestra de la traza, y su probabilidad representa la probabilidad de que esa muestra sea escogida para la creación de la plantilla. Esta distribución de probabilidad puede inicializarse aleatoriamente o usando algún criterio, como por ejemplo, con la correlación de fuga.

2. Se evalúan los individuos por separado. Para ello, se genera la plantilla correspondiente al individuo, y se realizan X ataques para obtener la posición en la que se encuentra el candidato de la clave correcta. Esta posición se encuentra entre 0 y 255, ya que al realizar el ataque, solo se busca 1 de las 16 subclaves del total de la clave, y al ser de 8 bits, tenemos 256 posibles subclaves. Si se encuentra en la posición 0, significa que el candidato correcto es el primero de la lista, y, por tanto, se habrá conseguido realizar el ataque con éxito con los Puntos de Interés de dicho candidato.

3. Una vez evaluados los individuos, se escogen los R individuos más prometedores, es decir, los R individuos que han obtenido mejor puntuación, creando el subconjunto D_L^R. Con estos individuos, se recalcula la probabilidad de cada variable x_i del vector binario partiendo de las variables de los mejores individuos. La probabilidad se recalcula de la siguiente forma:

$$p_L(x_i) = \sum_{j=1}^{R} \frac{\delta_j(x_{j,i})}{R} \tag{3.4}$$

donde:

$$\delta_j(x_{j,i}) = \begin{cases} 1, & \text{si } x_{j,i} = 1. \\ 0, & \text{en otro caso.} \end{cases}$$

Es decir, la probabilidad de cada variable se recalcula en función de cuantos de los mejores individuos tienen a 1 esa variable. Cuantos más individuos tengan esa variable a 1, mayor será la probabilidad resultante.

4. Una vez recalculadas las probabilidades de las T variables, se genera una nueva población con dichas probabilidades. El proceso se termina cuando se alcanza una condición de parada. Esta condición puede ser cuando el mejor individuo tiene una puntuación razonable (por ejemplo, la subclave correcta está entre los primeros 5 candidatos), que el individuo evaluado ha conseguido la subclave correcta en la primera posición o se ha alcanzado el máximo número de iteraciones M.

3.7.2. Metodología Six Sigma

La metodología Six Sigma es una metodología que ayuda a las organizaciones a mejorar la calidad de un producto o servicio, aunque también se puede aplicar a un proceso de fabricación y producción. Se basa en un proceso llamado DMAIC (Definir, Medir, Analizar, Mejorar y Controlar, ver Figura 3.7), y utiliza técnicas estadísticas para analizar datos con el objetivo de reducir la variabilidad o eliminar las causas de los fallos.

A continuación, se detallan los pasos de la metodología Six Sigma:

- Definir: Identificar el problema y definir los objetivos y el plan de mejora.
- Medir: Recopilar datos relevantes usando las métricas necesarias.
- Analizar: Realizar el análisis sobre los datos recopilados para ver qué factores y de qué manera influyen en el resultado final.
- Mejorar: Desarrollar e implementar las soluciones necesarias.
- Controlar: Comprobar que las soluciones o mejoras solucionan o mitigan el problema a lo largo del tiempo.

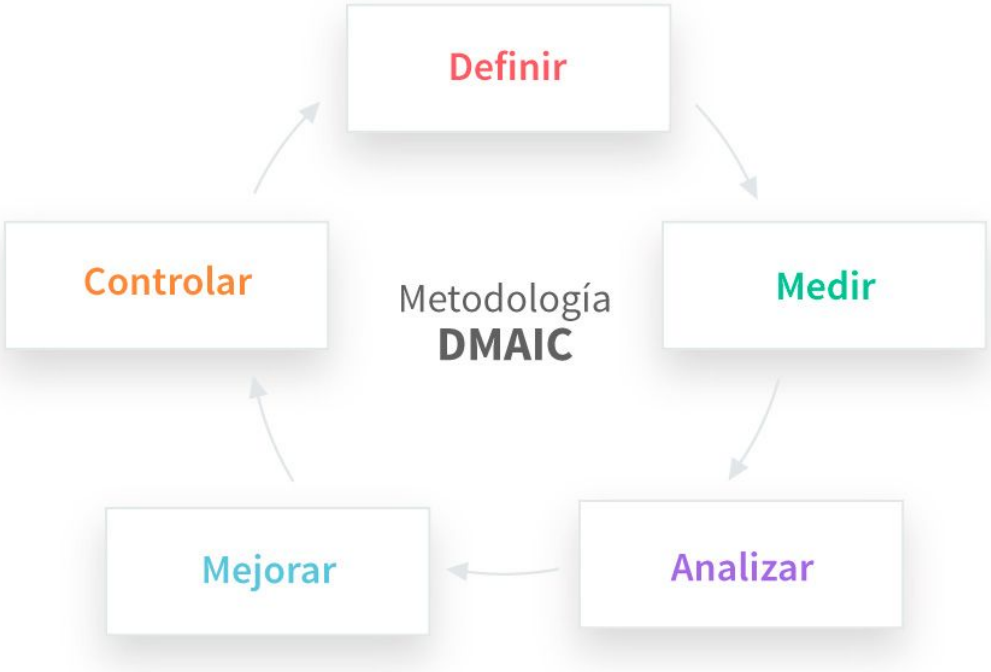

Figura 3.7: Proceso DMAIC [Sandrine, 2010]

En el Capítulo 5 se detallará más en profundidad como se implementara esta metodología.

4. CAPÍTULO

Diseño e Implementación

En este apartado, se va a hablar sobre el desarrollo de la herramienta, desde el planteamiento inicial, recursos utilizados y las utilidades implementadas.

4.1. Diseño de la herramienta

4.1.1. Objetivos

Antes de empezar a desarrollar la herramienta, se necesita saber cuáles son los objetivos que se quieren conseguir con la herramienta desarrollada. Los objetivos son los siguientes:

- Crear herramienta inicial: El primer objetivo es conseguir una herramienta que sea capaz de leer ficheros en formato *h5* del dataset del repositorio [urioja, 2022], e implementar el ataque de perfilado del repositorio [Bubberman et al., 2020].
- Modificar la herramienta para integrar un EDA, de forma que cada individuo de la población del EDA sea un vector de puntos de interés. La idea es evaluar cada individuo realizando un ataque de perfilado, e intentar obtener la clave de un dispositivo de esta forma.
- Paralelizar la ejecución de los ataques para reducir los tiempos de los ataques.
- Integrar diferentes tipos de inicialización, aparte del aleatorio, como por ejemplo SOST.

- Posibles mejoras añadidas. Esto incluye:
    - Realizar una batería de varios ataques durante la búsqueda, para mejorar la generabilidad de los modelos obtenidos
    - Integrar PCA para convertir las trazas
    - Transformar trazas de archivo *trs* a *h5*, que son las trazas que se obtienen de usar el programa para las mediciones.

4.1.2. Recursos utilizados

Para desarrollar la herramienta, se ha utilizado el lenguaje de programación Python. Se ha utilizado la versión 3.11.2, además de una lista de paquetes extra. Entre ellas, podemos destacar:

- Numpy: utilizado para realizar cálculos matemáticos con matrices
- Pandas: utilizado para generar archivos con los resultados
- Multiprocessing: utilizado para paralelizar la herramienta
- Matplotlib: utilizado para generar gráficas
- Datetime: utilizado para obtener los tiempos de ejecución
- Random: utilizado para aleatorizar la inicialización
- Trsfile: utilizado para leer archivos de extensión trs
- h5py: utilizado para leer y generar ficheros de extensión h5

Como entorno de desarrollo, se ha utilizado Visual Studio Code.

4.2. Implementación de la herramienta

Para hablar sobre la implementación de la herramienta, tenemos diferenciados los ficheros en función de su tarea dentro de la herramienta.

4.2.1. Ataque de perfilado

Este apartado engloba los ficheros necesarios para poder realizar un ataque de perfilado. Originalmente, estos ficheros eran los del repositorio [Bubberman et al., 2020], pero se han tenido que modificar y adaptar.

- Fichero attack\ta.py: En este fichero encontramos la clase TA, que incluye todas las funciones necesarias para realizar un ataque de perfilado. Originalmente, incluía las funciones en versión serie para el ataque, además de que obtenía los puntos de interés simplemente aplicando una fórmula (por ejemplo, correlación) y seleccionando los N puntos con mayor valor. Esto hace que normalmente el evaluador tenga que probar distintos análisis con distintas fórmulas y distinto número de puntos hasta que se obtenga un resultado correcto, si es caso. En la versión actual, incluye también las funciones para realizar ataques de forma paralela, y en vez de seleccionar los puntos de interés de forma manual, usa el vector de puntos de interés de los individuos generados por el EDA para hacer una búsqueda automática de los mejores puntos de interés. También incluye funciones donde se crea el gráfico del Guessing Entropy, que es un gráfico con la posición de la clave correcta con los diferentes números de trazas que se usan para el ataque de perfilado. En la Figura 4.1 tenemos un ejemplo. En él, podemos ver como evoluciona la posición de la clave correcta hasta llegar a la posición 0, lo que indica que el ataque ha tenido éxito.

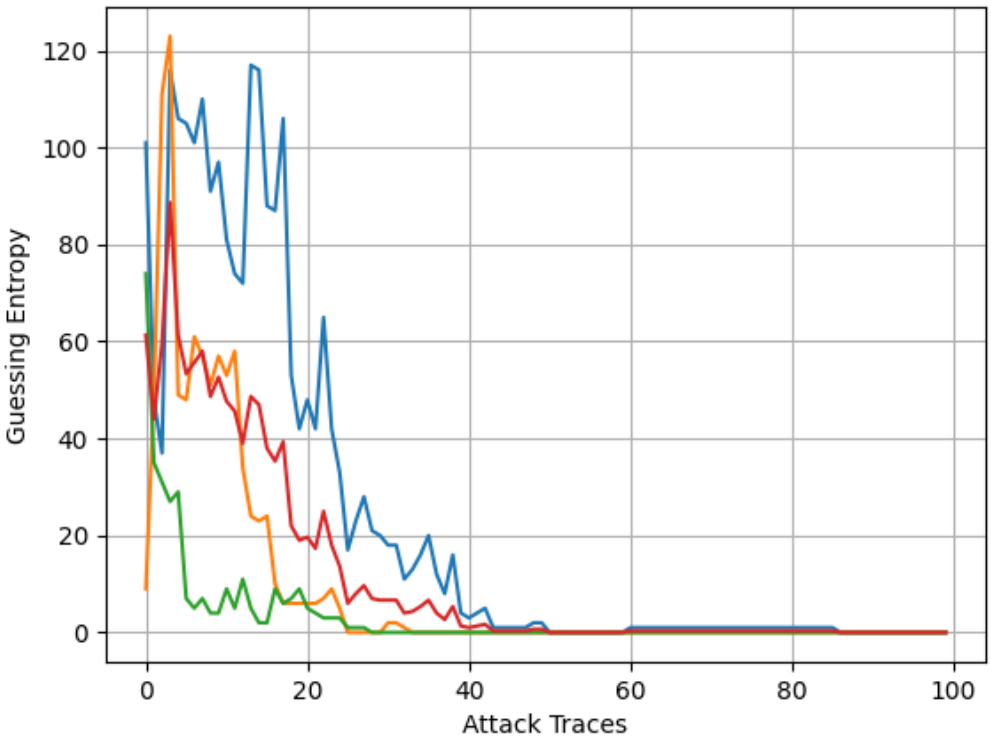

Figura 4.1: Ejemplo de Guessing Entropy

- Fichero analysis\sostd.py: En este fichero encontramos la clase SOSTD, que incluye las funciones para realizar el análisis de las trazas con el error cuadrático medio, que también se puede configurar para realizar el error cuadrático medio normalizado. De este fichero, usamos la función *list_of_best_indices*, qué dadas las trazas que queremos analizar junto con el texto de entrada y la clave asociada a cada traza, obtenemos un vector con la correlación en cada punto de la traza.

- Fichero util\load_hdf5_file.py: Originalmente, este fichero se llamaba fileloader, y leía ficheros con la extensión npy. Ahora, se encarga de leer las trazas, texto y clave del dataset desde un fichero h5.

- Fichero util\hamming_weight.py: Este fichero incluye la clase HammingWeight, que contiene las funciones para realizar los cálculos necesarios de la fuga de información. Como existen 2 posibles modelos de fuga, se han mantenido ambas opciones disponibles para usar.

- Fichero util\aes.py: Este fichero incluye la clase AES y la tabla S-BOX de sustitución. Es idéntico al fichero original, ya que se utiliza para el cálculo de la fuga.

4.2.2. EDA

En este apartado se encuentran los ficheros necesarios para el algoritmo de estimación de distribución. Estos ficheros son los del repositorio proporcionado por Ikerlan, y contienen lo necesario para hacer funcionar el EDA. Aun así, se han tenido que modificar algunos de los ficheros para adaptarlo a este proyecto.

Antes de explicar cada fichero, es conveniente explicar como es el funcionamiento del EDA para la herramienta. Primero de todo, necesitamos saber cuantos puntos tienen las trazas del dispositivo que vamos a analizar, y a cada punto se le asignara una probabilidad. Es decir, si tenemos 1000 puntos posibles, a cada uno le debemos asignar una probabilidad de ser escogida, que puede ser de forma aleatoria o siguiendo algún método, como SOST. Después, para cada individuo que tenemos en cada iteración, tenemos que muestrear cada uno de los puntos posibles, es decir, determinar si para dicho individuo es 1 (es decir, ese punto se ha escogido para ser un punto de interés) o 0 (en caso contrario), y así, poder saber cuáles son los puntos de interés de cada individuo. Tras esto, se evalúa cada individuo con un ataque de perfilado, y se ordenan los individuos en función de la puntuación obtenida. Cuando todos los individuos están evaluados, se escoge la mitad de los individuos, teniendo así los mejores individuos, y con los vectores de los puntos de interés de los mejores individuos, se recalcula la probabilidad de cada punto, en función de la frecuencia de cada punto entre los mejores individuos. Es decir, si un punto aparece en todos los mejores individuos, su nueva probabilidad será 1, si no aparece en ninguno, será 0, y si aparece en 3 de 15 puntos, su probabilidad será dé $3/15 = 0{,}2$. Y una vez recalculadas las probabilidades, se vuelven a generar individuos para poder ser evaluados.

Para evaluar los individuos, se ha creado la forma paralela de evaluarlos. Para hacer esto, en un inicio había dos posibilidades. La primera sería paralelizar el bucle donde se realiza el ataque del individuo, entre otros, ya que es el de mayor carga de trabajo supone. La segunda opción sería paralelizar la evaluación de los individuos, es decir, evaluar los individuos en paralelo en vez de uno en uno, de forma que cada núcleo del ordenador se encarga de evaluar un individuo a la vez. Se ha decidido probar la segunda opción, ya que requiere de menos cambios que el primer caso y, por tanto, más sencillo de implementar. Además, paralelizar de la primera forma no sería tan sencillo, ya que se requiere de sincronización entre núcleos para realizar el cálculo necesario

A continuación, se verán los ficheros más importantes del EDA.

- Fichero ICBernouilli.py: Este fichero contiene la clase ICBernouilli, que se encarga de asignar a cada punto de la traza su probabilidad de ser escogido como un punto de interés, y que se usa para muestrear cada individuo de la población en cada iteración del algoritmo.

- Fichero ICIndividual.py: Este fichero contiene la clase ICIndividual, y es el que se encarga de inicializar cada individuo, además de tener la función para imprimir la información del mismo. Esta última se tuvo que modificar para que la información del individuo sea su número de puntos de interés, ya que mostrar un vector binario de 1000 valores no es muy representativo.

- Fichero ICPopulation.py: Este fichero contiene la clase ICPopulation, y es el que gestiona la población de individuos. Incluye las funciones para ordenar los individuos, seleccionar los mejores individuos e imprimir la información de la población en cada iteración.

- Fichero ICSCAttack.py: Este fichero contiene la clase ICSCAttack, y es el fichero donde se encuentran todas las funciones necesarias para poder evaluar la población. Encontramos las funciones de la versión serie y de la paralela. También encontramos las funciones que evalúan el mejor individuo de cada iteración, que lo usamos para la fase de validación. Esto es para comprobar que los puntos de interés del mejor individuo no solo sirven para ciertas trazas, sino que puedan usarse con cualquier traza del mismo tipo del dataset. Originalmente, este fichero solo incluía las funciones en serie y sin las funciones de validación. De hecho, la función de evaluación solo tenía en cuenta el número de puntos de interés, y a mayor número de puntos, mayor puntuación asignaba a cada individuo.

- Fichero ICSelectBest.py: Este fichero contiene la clase ICSelectBest, y es la encargada de escoger los mejores individuos después de haberse ordenado. Para la selección, usa un porcentaje, que está definido de forma predeterminada con el valor 0.5, es decir, selecciona la mitad de la población evaluada. A este fichero no se le ha hecho ningún cambio.

4.2.3. EDASCA

En este apartado se va a detallar del fichero más importante de la herramienta. Originalmente, venía incluido en el repositorio facilitado por Ikerlan, pero ha sufrido muchos cambios. Originalmente, este fichero incluía las siguientes funciones:

- initUMDA: Se encarga de inicializar todos los parámetros relacionados con el EDA. Entre los parámetros, se encuentran el número de iteraciones máximo y el tamaño de la población.
- createPopulation: Se encarga de crear una población con el tamaño de población indicada. Crea los individuos con las propiedades necesarias, como el tamaño del vector binario para los puntos de interés.
- initializeModel: Se encarga de asignar a cada punto de la traza su probabilidad, que más adelante se usara para generar los vectores binarios de los puntos de interés de los individuos.
- sampleModel: Se encarga de muestrear los vectores binarios de los puntos de interés.
- evaluatePopulation: Se encarga de llamar a la función encargada de evaluar la población.
- sortingPopulation: Se encarga de llamar a la función que ordena los individuos en función de la puntuación obtenida en la evaluación.
- showPopulation: Se encarga de llamar a la función que muestra la información de la población.
- selectPopulation: Se encarga de llamar a la función que genera una población seleccionando los mejores individuos tras la evaluación.
- learnModel: Se encarga de recalcular las probabilidades de los puntos para la siguiente iteración, usando la población seleccionada con los mejores individuos y con sus vectores binarios.
- run: Esta es la función que se encarga de ejecutar el algoritmo EDA, usando todas las funciones anteriores.

Después de realizar los cambios, este fichero incluye nuevas funciones. Algunas de ellas son para realizar la evaluación de forma paralela, y otras para realizar diferentes inicializaciones de la primera población del algoritmo. Las funciones nuevas son las siguientes:

- initParameters: Esta función se ha creado para inicializar los nuevos parámetros necesarios, tanto para el EDA, como algunos de los parámetros para los ataques de perfilado, y algunos parámetros adicionales utilizados para algunas utilidades añadidas.

- init_ta: Esta función se encarga de cargar las trazas para los ataques de perfilado, además de guardarlos para poder realizar los ataques de la forma necesaria, ya que la clase TA necesita tener los datos previamente cargados.

- initialize_model: Aunque no exista una única versión de esta función, todas ellas tienen la misma función, que es encargarse de inicializar la población inicial según la forma deseada. Estos son los modos implementados:

  - Aleatorio: a cada punto se le asigna una probabilidad aleatoria entre 0 y 1.
  - Usando correlación mediante SOST: se calcula la correlación mediante SOST, y tras esto, se asigna la probabilidad a cada punto con el valor estandarizado de la correlación.
  - Selección manual: los puntos seleccionados a mano tendrán una probabilidad de 1, y los demás, 0.
  - Probabilidad decreciente: asigna una probabilidad cada vez menor según avanza en el número del punto al que asigna la probabilidad. Es decir, a los primeros puntos de la traza asigna mayor probabilidad que los del final. Esto está fundamentado en la idea de que, al atacar la S-BOX en la fase de cifrado, la fuga se encontrara en los primeros puntos de la traza, mientras que los últimos no tendrán fuga.
  - Primeros puntos: asigna una probabilidad de uno a los primeros N puntos, y 0 para el resto. Es una mezcla entre el modo de inicialización manual y decreciente.

- evaluatePopulationPool2: Esta función es la versión paralela para evaluar la población. La idea es generar un conjunto de procesos, donde cada proceso se encarga de ejecutar la función de evaluación de un individuo de la población.

- run_cleaned: Esta función es la que se encarga de ejecutar todo el proceso del EDA, con las nuevas funciones creadas y actualizado para usar los ataques de perfilado para evaluar los individuos. Originalmente, la ejecución de la herramienta original solo realizaba un ataque con cada individuo. En esta nueva versión, se incluyen dos cambios. El primero, en vez de realizar un ataque por individuo, se puede configurar un número diferente de ataques para evaluar los individuos, y a esta fase de evaluar todos los individuos se le ha denominado la fase de train. Y el segundo es una fase llamada test (o fase de validación), donde se valida si el mejor individuo es un modelo generalizado o no. Esto es muy importante, ya que si un individuo solo es válido para ciertas trazas, es posible que en la fase de train obtenga la clave, pero no en la de validación, por lo que no sería un modelo generalizado, ya que solo funcionaria con ciertas trazas. Además, también se encarga de detener el proceso iterativo del EDA si el mejor individuo consigue obtener la clave en la fase de validación. Devuelve el tiempo de inicialización, el tiempo del algoritmo EDA, el tiempo para el test del mejor individuo obtenido en todas las iteraciones del EDA y el tiempo al final de la ejecución. Estos tiempos sirven para poder calcular el tiempo necesitado por cada apartado, que será necesario en las pruebas.
- run_doe: Similar a la anterior, esta función devuelve también algunas de las puntuaciones obtenidas en la evaluación. Devuelve la puntuación media de la última población en entrenamiento, la puntuación del mejor individuo de la última población en entrenamiento y la puntuación del mejor individuo de la última población en validación.
- check_folder_name: A medida que pasan las iteraciones, la herramienta guarda el modelo (los puntos de interés) del mejor individuo en una carpeta. Esta función busca cuál de esos modelos guardados tiene mejor puntuación en validación.
- initialize_mode: Esta función se encarga de llamar a la función de inicialización correspondiente, según el método especificado, que puede ser por el usuario o el predeterminado (aleatorio).
- iteration: Esta función es la que se encarga de llamar a las funciones necesarias para evaluar la población de una iteración, obtener el mejor individuo y realizar el ataque de validación con dicho individuo.
- check_iteration: Esta función comprueba si el proceso iterativo del EDA debería detenerse. Esto ocurre cuando todos los individuos de la población tienen el mismo valor, por lo que seguir iterando no tendría sentido.

- calc_max_poi: Esta función se encarga de calcular cuál es el número máximo de puntos de interés puede tener un individuo. Esta función originalmente se realizaba en cada una de las evaluaciones de los individuos, por lo que al moverlo, ahorramos tiempo en cada una de las evaluaciones que se hace, ya que pasamos de ejecutar la función en cada individuo de cada iteración a solo 1 vez. Además, también sirve para comprobar si la inicialización de la población ha sido válida.

4.2.4. Otros ficheros

Para no tener todas las funcionalidades dentro del fichero EDASCA.py, se han creado nuevos ficheros con esas funcionalidades creadas. Entre ellos, hay algunos que merece la pena comentar un poco más en profundidad. Algunos son utilidades añadidas, y otros son ficheros para ejecutar la herramienta. A continuación, se listan esos ficheros:

- run_cleaned.py: Aunque no haya solo una versión de este fichero, todos llaman a la función run_cleaned de EDASCA.py. Además, desde este fichero podemos configurar los diferentes parámetros que necesitamos, como por ejemplo, tamaño de población, método de inicialización o número máximo de iteraciones. Por ello, podemos crear varias copias del mismo, donde cada uno tiene su configuración particular.

- trs_converter.py: Este fichero se encarga de convertir ficheros de extensión .trs a ficheros de extensión h5. Esta utilidad es útil cuando tenemos nuevas trazas y necesitamos probarlas con la herramienta. Aunque en el proyecto no se ha hecho una adquisición de nuevas trazas, sí que nos ha servido para comprobar que los ficheros con las trazas del dataset y los del fichero generado coinciden.

- trs_converter_pca.py: Similar al anterior, pero en este caso, convierte el fichero de extensión .trs a un fichero de extensión h5 tras realizar una transformación PCA.

Sin embargo, hay otros ficheros que simplemente realizan la llamada a una función dentro de EDASCA.py para realizar alguna prueba o comprobación concreta. A continuación, se listan esos ficheros:

- test_num_process.py: Permite ejecutar la herramienta para realizar comparaciones con diferentes números de núcleos del ordenador. Además, guarda los tiempos en un fichero CSV para poder analizarlos posteriormente.

- test_num_profiling_traces.py: Permite ejecutar la herramienta para comparar diferentes números de trazas de perfilado. También genera un fichero CSV con los tiempos.
- test_profiling_attack.py: Una modificación de la anterior, que también permite configurar el número de trazas para el ataque.
- test_poi.py: Permite ejecutar una función de la herramienta para comparar diferentes números de puntos de interés al tiempo de ejecución. También genera un fichero CSV con los tiempos.
- test_serie_parallel.py: Permite ejecutar la herramienta y comparar los tiempos en serie y en paralelo con la configuración deseada.
- DOE.py: Este conjunto de ficheros se encargan de realizar las pruebas de análisis que necesitamos. Ejecutan la función run_doe del archivo EDASCA.py, y guarda los resultados obtenidos en un fichero CSV para su posterior análisis.

Además de estos ficheros, también se han creado ficheros para generar los gráficos de los resultados de las diferentes pruebas realizadas. Estos ficheros han sido de mucha utilidad para poder representar los resultados de forma visual.

5. CAPÍTULO

Experimentación

La fase de experimentación constará de tres partes. La primera parte tratará sobre el entorno de experimentación, detallando la metodología seguida para las pruebas y el equipamiento necesario para las mismas. La segunda parte tratará sobre pruebas realizadas para medir el rendimiento. Por ejemplo, ver qué variables afectan más a los tiempos de ejecución, el tiempo sería y paralelo que necesita el programa para la ejecución y la duración de cada parte del programa según las variables que usemos. Y la tercera parte tratará de pruebas sobre el Análisis de Canal Lateral. Por ejemplo, ver qué variables afectan más para obtener la clave, que tipo de inicialización es mejor, si el Análisis de Componentes Principales ayuda o no, las diferencias entre los dispositivos analizados y las diferencias entre los tipos de protecciones de los dispositivos analizados.

5.1. Entorno de experimentación

En este apartado se van a detallar todo el material usado para esta fase de experimentación. Esto incluye los elementos Software y Hardware usados, además de la metodología utilizada para los experimentos.

5.1.1. Hardware

En el apartado Hardware nos encontramos los equipos utilizados para desarrollar la herramienta y los dispositivos sometidos a las pruebas.

Empezando por los equipos usados para desarrollar la herramienta, nos encontramos un portátil Dell Latitude 5590, con un procesador I7-8650U de 4 núcleos y 8 hilos y 16 GB de RAM, que se ha usado principalmente para desarrollar la herramienta, además de realizar alguna prueba pequeña para comprobar el correcto funcionamiento de la misma. El otro equipo utilizado es un ordenador de trabajo HP Z440, con un Intel Xeon e5-1650 v4 de 6 núcleos y 12 hilos y 32 GB de RAM, que se ha utilizado para ejecutar los diferentes experimentos diseñados.

En cuanto a los dispositivos a evaluar, nos encontramos con 2 dispositivos.

- Piñata: La Piñata es una placa de desarrollo creada por la empresa Riscure. Está basado en un núcleo ARM Cortex-M4F a una velocidad de reloj de 168 MHz. Esta placa ha sido modificada físicamente y programada para ser un objetivo de entrenamiento para el Análisis de Canal Lateral. Esto se ha conseguido eliminando los condensadores de desacoplamiento, que alteran la medición del consumo de la placa. En la Figura 5.1 podemos ver una imagen de la placa.
- Discovery: La STM32F411E-DISCO es una placa de desarrollo de los microcontroladores de la serie STM32F4. Está basada en un núcleo ARM Cortex-M4 de 32 bits a una velocidad de 100 MHz. Es una placa similar a la Piñata, ya que los microcontroladores son de la misma familia. A diferencia de la Piñata, esta placa no ha sido modificada, para poder evaluar un dispositivo lo más parecido a un escenario real, ya que los condensadores de desacoplamiento eliminan gran parte de la fuga. En la Figura 5.2 podemos ver una imagen de la placa.

Las trazas de los dispositivos se han obtenido de un dataset [urioja, 2022], donde se encuentran las trazas separadas en dos partes. La primera parte usa claves aleatorias, y son las trazas que se utilizaran para el apartado del ataque. Y la segunda parte usa clave fija, y son las trazas que se utilizaran para el modelo. Estas trazas se han obtenido mientras se ejecuta el algoritmo AES con clave de 128 bits, en modo ECB. Además, pueden ser de 3 tipos:

- Sin protección: el algoritmo no incluye ningún tipo de protección contra SCA. Es el escenario más sencillo y el que no necesita cambios sobre el algoritmo.
- Enmascarado tipo 1: el algoritmo ha sido modificado para usar enmascaramiento, pero la mascará se elimina por cada Byte de la SBox enmascarada. Esto hace que durante la operación de sustitución con la SBox exista una fuga.

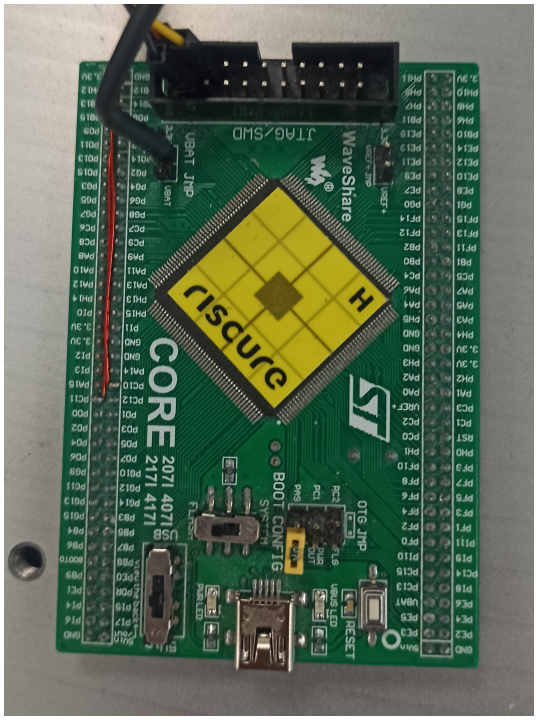

Figura 5.1: Placa Piñata

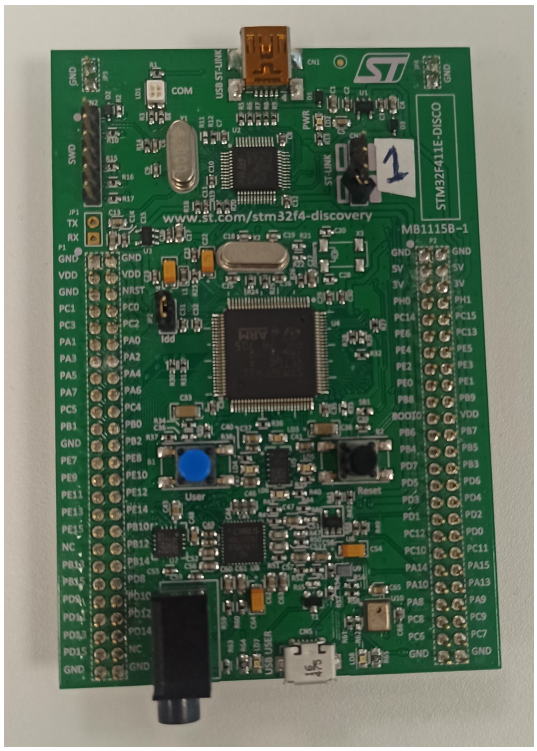

Figura 5.2: Placa Discovery

- Enmascarado tipo 2: el algoritmo se ha modificado para usar enmascaramiento, pero la mascará se elimina tras realizar la operación ShiftRows. Esto hace que durante la operación de sustitución con la SBox no haya fuga.

Para las pruebas que se van a hacer, se van a utilizar las trazas sin protección. Esto es debido a que son el escenario más sencillo para comprobar si los dispositivos son vulnerables a SCA.

5.1.2. Software

En el apartado Software nos encontramos con la herramienta desarrollada, que es el que hemos utilizado para realizar la evaluación de los dispositivos, además de las diferentes utilidades creadas para poder realizar dichos análisis. Al tener un dataset previo, no va a ser necesario usar ningún programa para obtener trazas, y todo lo relacionado con las pruebas se hará con la herramienta y las utilidades.

5.1.3. Metodología

En el apartado de la metodología, nos encontramos con 2 partes. La primera parte será la metodología a seguir para las pruebas de rendimiento, y la segunda para los de SCA.

Las pruebas de rendimiento van a ser independientes entre sí, por lo que la metodología a seguir es muy sencilla. La idea es realizar cada una de las pruebas un número determinado de veces, y utilizar la media de los resultados para analizar los resultados y graficar los datos.

Para las pruebas de SCA, se va a usar la metodología Six Sigma. Esto es porque, a diferencia de las pruebas de rendimiento, en estas pruebas sí es importante el resultado del análisis, y no tanto el tiempo. Esto hace que estas pruebas sean más importantes que las anteriores, y que sea más interesante aplicar esta metodología. Es por esto que necesitamos detallar algunos aspectos antes de empezar con la experimentación.

Para esta metodología, necesitamos un criterio para finalizar el proceso (criterio “Ok”), para poder limitar las pruebas que vayamos a hacer de alguna forma. En este caso, se va a definir como dicho criterio que en los resultados de evaluación se consiga obtener la clave del dispositivo en la 1ª posición. Además de esto, tenemos otro aspecto a tener en cuenta. Y es que, para las pruebas de SCA, vamos a utilizar una única subclave de las 16 que

completan la clave, ya que si somos capaces de obtener una subclave, podremos obtener el resto de las subclaves con el mismo proceso. Además, se necesita mucho menos tiempo para obtener una subclave que no la clave completa. Por ello, si la subclave correcta se encuentra en la 1ª posición, el dispositivo es vulnerable a SCA. Como estamos evaluando 2 dispositivos, las pruebas empezaran con el primero (Piñata), y cuando se alcance el criterio "Ok", se pasará al segundo (Discovery) hasta alcanzar el criterio "Ok" también.

5.2. Pruebas de rendimiento

Estas pruebas de rendimiento tienen 2 objetivos. El primero es ver qué variables o parámetros afectan directamente al tiempo de ejecución del programa y cuáles no. Y el segundo es ver de qué manera afecta la paralelización realizada a los tiempos de ejecución. Para ello, se han realizado los siguientes experimentos.

5.2.1. Probando diferentes puntos de intereses

Para este experimento, se han probado diferentes números de puntos de interés para ver de qué manera afectan a los tiempos de ejecución, tanto en serie como en paralelo. Se han probado desde 1 punto hasta 30 puntos, que es un rango amplio para poder comparar los tiempos. La ejecución se realizará tanto en la versión serie como en la paralela. El resto de parámetros configurados son los siguientes:

- Trazas de perfilado: 20000
- Trazas de ataque: 100
- Número de ataques: 1
- Número de ataques de validación: 1
- Tamaño de población: 8
- Número de iteraciones: 1
- Modo de inicialización: a mano

Además, para que los resultados de la prueba sean consistentes, se ha repetido cada prueba 5 veces, y se ha usado la media para obtener el resultado. En total se han realizado 145

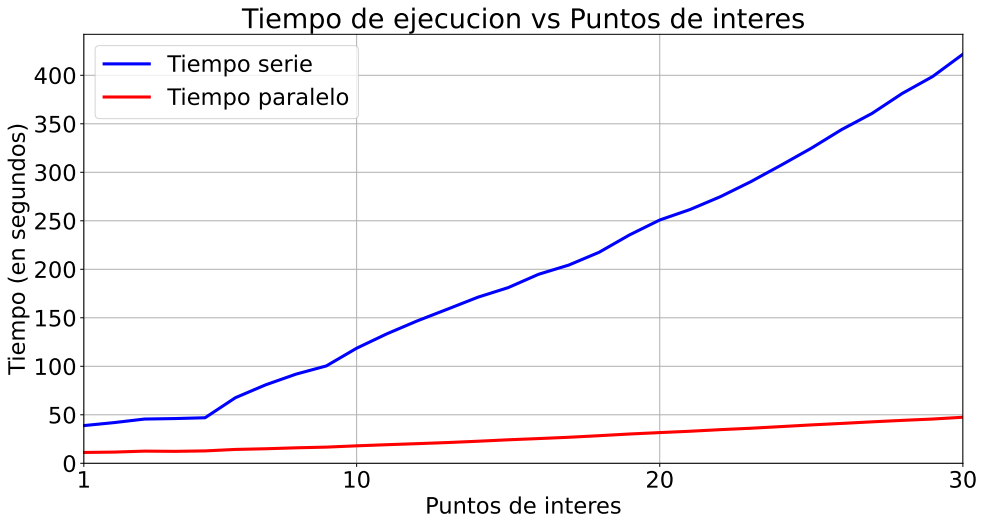

Figura 5.3: Tiempo de ejecución vs Puntos de Interés, serie y paralelo

pruebas para este experimento. En el gráfico de la Figura 5.3 podemos ver el tiempo medio de la ejecución en serie y en paralelo de la prueba, en función de los puntos de interés usados. A simple vista, podemos observar que en todas las pruebas el tiempo en serie es mayor que el tiempo paralelo, por lo que ya podemos ver que la paralelización funciona correctamente. Además, también se puede apreciar que la pendiente del tiempo serie es mayor que el del tiempo en paralelo. Esto indica que cuantos más puntos de interés seleccionamos, mayor es el factor de aceleración que obtenemos.

5.2.2. Probando diferentes trazas para modelo y ataque

Para este experimento, se han probado diferentes números de trazas para la fase de perfilado y para la fase de ataque. Para las trazas del modelo, se han usado los valores 10000, 20000 y 30000, y para las trazas de ataque, se han usado los valores 10, 50, 100 y 200. La ejecución se realizará en la versión paralela, para ahorrar tiempo. El resto de parámetros configurados son los siguientes:

- Número de ataques: 1
- Número de ataques de validación: 1
- Tamaño de población: 8
- Número de iteraciones: 1

- Modo de inicialización: aleatorio

Además, para que los resultados de la prueba sean consistentes, se ha repetido cada prueba 5 veces, y se ha usado la media para obtener el resultado. En total se han realizado 60 pruebas para este experimento. En el gráfico de la Figura 5.4 podemos ver el tiempo medio de la ejecución en paralelo de la prueba. Los colores nos indican el número de trazas de perfilado usadas, mientras que en el eje horizontal encontramos el número de trazas de ataque usadas. Como podemos apreciar, las diferencias de los tiempos de ejecución en las diferentes opciones de trazas de perfilado son muy pequeñas, mientras que se aprecia una gran diferencia cuando observamos las diferentes opciones de trazas de ataque usadas. Esto nos indica que las trazas para el ataque afectan más al tiempo de ejecución que las trazas de perfilado.

Por otro lado, en el gráfico de la Figura 5.5 podemos ver la media de los tiempos de inicialización de la prueba. Como podemos apreciar, el tiempo de inicialización sí que aumenta en gran medida cuando aumentamos las trazas de perfilado, pero no hay apenas diferencia entre las opciones de trazas de ataque. Esto nos indica que las trazas para el perfilado afectan más al tiempo de inicialización que las trazas de ataque. Aunque el aumento del tiempo no es proporcional al aumento del número de trazas para el ataque o de perfilado, es algo que se deberá tener en cuenta de cara a las pruebas de análisis, para ver si el aumento del número de trazas ayuda a obtener mejores resultados, y si dicha mejora compensa el aumento del tiempo de ejecución.

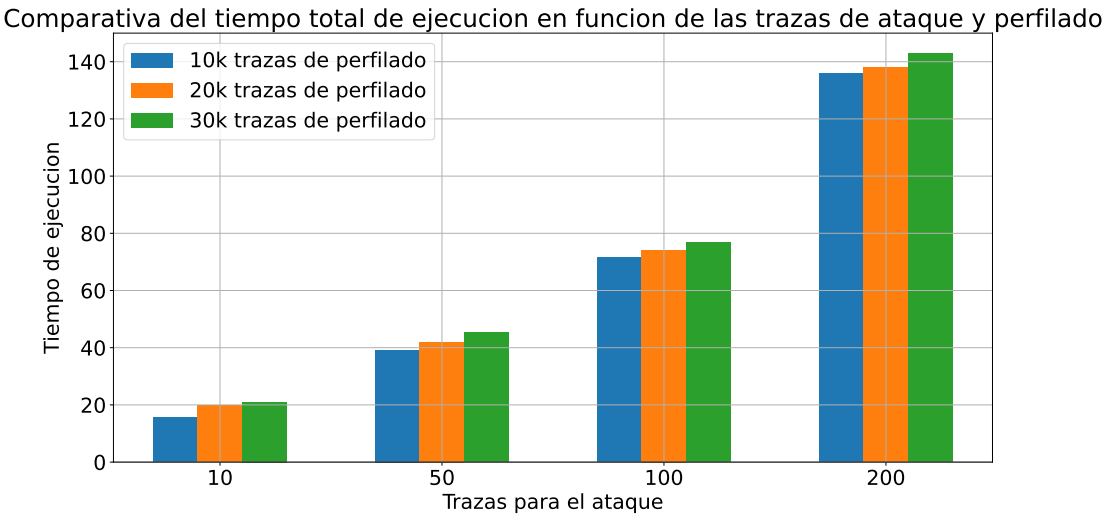

Figura 5.4: Tiempo de ejecución usando diferentes trazas para el modelo y ataque

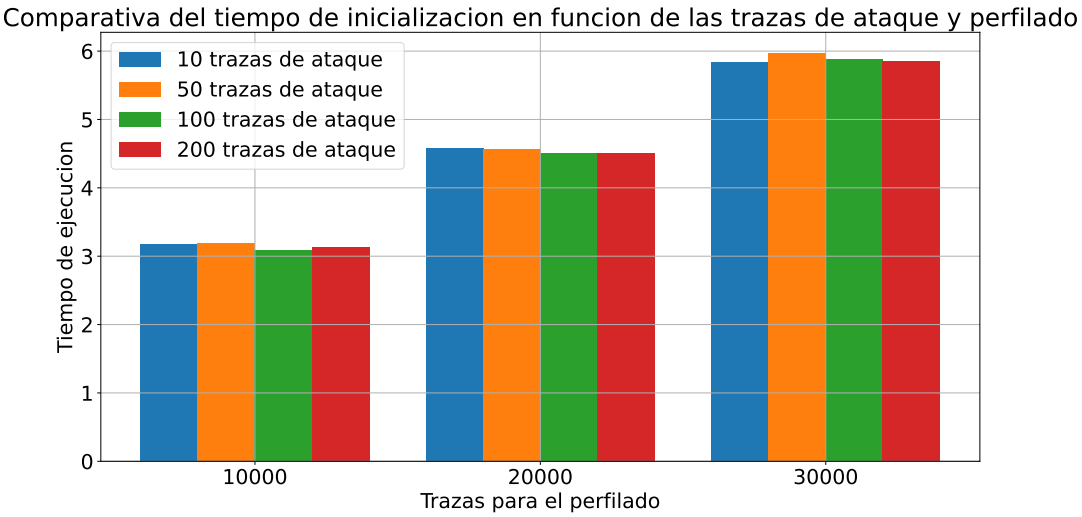

Figura 5.5: Tiempo de inicialización usando diferentes trazas para el modelo y ataque

5.2.3. Comparativa serie vs paralelo

Para esta parte de la experimentación, se van a realizar 2 experimentos diferentes. La primera constará de una prueba donde se probaran diferentes números de hilos para la versión paralela, y así obtener las métricas de rendimiento respecto a la ejecución en serie. Y la segunda constará de una comparación entre una ejecución completa en ambas versiones, para ver así la diferencia de tiempo obtenida en una prueba de un escenario posible de evaluación de un dispositivo.

Usando diferentes números de hilos

Para este experimento se ha ejecutado una pequeña prueba usando diferentes números de hilos, para comparar el rendimiento de la versión paralela respecto a la versión serie. El número de hilos usado está comprendido entre los valores 1 (serie) y 12. La ejecución se realizará tanto en la versión serie como en la paralela. El resto de parámetros configurados son los siguientes:

- Trazas de perfilado: 10000
- Trazas de ataque: 100
- Número de ataques: 1

- Número de ataques de validación: 1
- Tamaño de población: 24
- Número de iteraciones: 1
- Modo de inicialización: a mano

Además, para que los resultados de la prueba sean consistentes, se ha repetido cada prueba 10 veces, y se ha usado la media para obtener el resultado. En total se han realizado 120 pruebas para este experimento. En el gráfico 5.6 podemos ver el factor de aceleración que obtenemos con diferentes números de hilos. En él, se puede ver que usando 2, 3 y 4 hilos obtenemos un factor de aceleración superior al número de hilos (2.04, 3.67 y 4.35 respectivamente), pero con el resto de hilos, el factor de aceleración no sube de igual forma. El único caso donde vuelve a subir es con 12 hilos, que es el máximo disponible en el ordenador, obteniendo un factor de aceleración del 7.01. Además, en el gráfico 5.7 podemos ver como la eficiencia sigue un patrón inverso al anterior. Aun así, la eficiencia se mantiene en todo momento por encima del 50 %, por lo que podemos ver que no es un problema de escalabilidad del programa, ya que incluso con el número máximo de hilos sigue la eficiencia incluso aumenta un poco.

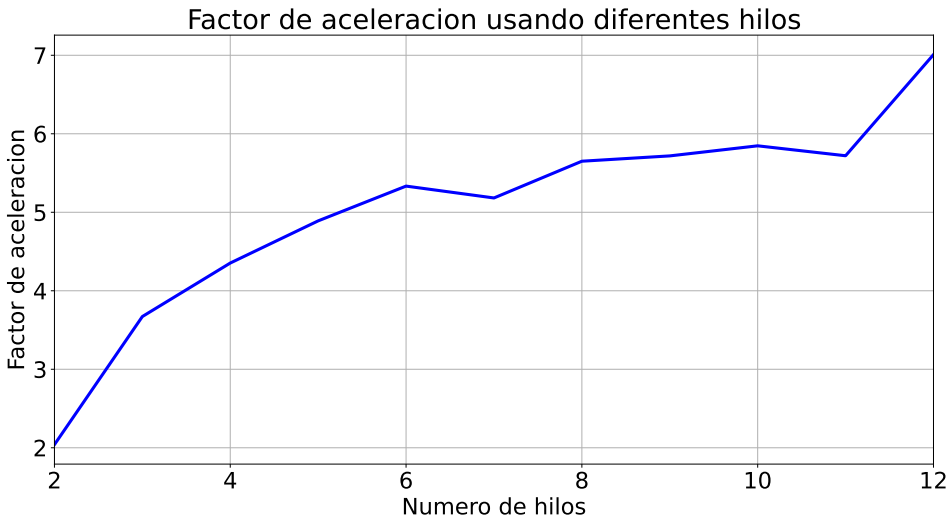

Figura 5.6: Factor de aceleración obtenido usando diferentes hilos

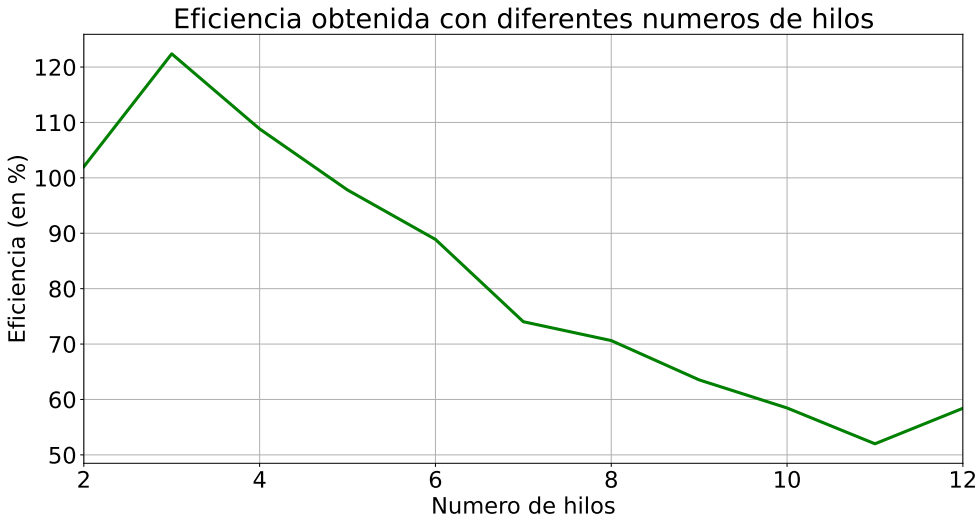

Figura 5.7: Eficiencia obtenida usando diferentes hilos

Ejecución completa

Para este experimento se ha ejecutado una prueba completa de un análisis, para comparar la diferencia de tiempo entre la versión serie y la paralela. El número de hilos usado para la versión paralela es de 12 hilos. Además, se ha utilizado el dataset de la Discovery, ya que es un escenario más complicado y que puede hacer que el análisis no termine en la primera iteración. El resto de parámetros configurados son los siguientes:

- Trazas de perfilado: 10000
- Trazas de ataque: 100
- Número de ataques: 2
- Número de ataques de validación: 2
- Tamaño de población: 30
- Número de iteraciones: 30
- Modo de inicialización: aleatorio

En este caso, la prueba se ha realizado 2 veces, ya que no necesitamos hacer tantas pruebas al ser una comparativa de un análisis completo, ya que es una prueba que necesita tiempo,

y la poca diferencia que pueda existir entre repeticiones no supone un gran cambio en los resultados finales. En la Figura 5.8 podemos ver los tiempos serie y paralelo al realizar las pruebas completas. Como era de esperar tras ver los resultados de la prueba anterior, hay una diferencia notoria entre el tiempo serie y paralelo de cada prueba, pero el factor de aceleración obtenido sigue sin ser proporcional al número de hilos usado (7.37291035 y 7.2770867 respectivamente).

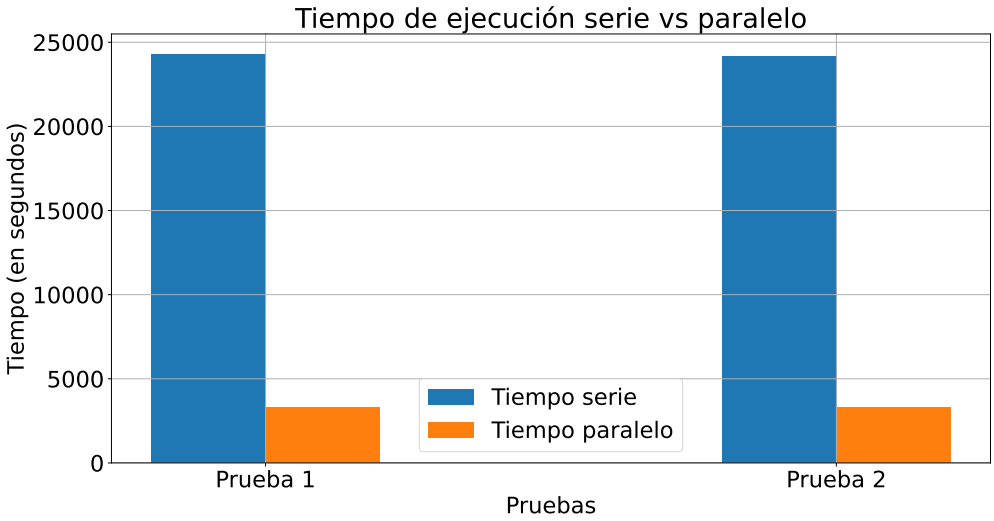

Figura 5.8: Tiempos de ejecución serie y paralelo

5.3. Pruebas de evaluación de dispositivos

Como ya se ha comentado previamente, para este apartado de la experimentación se va a usar la metodología Six Sigma. Por ello, vamos a seguir un proceso adaptado para nuestro caso específico. Primero, vamos a determinar qué variables queremos analizar, ya que no es necesario analizar todas. Primero, se han definido usa serie de variables como constantes, ya que son parámetros que no deberían de tener una influencia directa en los resultados. Algunos de esos parámetros se han probado en pruebas aisladas para ver si merecía la pena observarlos, como por ejemplo el modelo de fuga o el factor de corrección, pero se ha visto que con los parámetros definidos se obtienen mejores resultados. La lista la podemos ver en la Tabla 5.1. La lista de parámetros que quedan pendientes de analizar es la que encontramos en la Tabla 5.2.

En cada prueba se fijarán 3 de las variables para ser probados, y las demás tendrán un valor predeterminado para dicha prueba. Cada variable a probar tendrá 2 posibles valores, uno

Variable	Descripción	Rango	Valor
Modelo de fuga	Indica el modelo de fuga que se usara para la plantilla	Hamming Weight o Indentity Model	Hamming Weight
Agrupar trazas	Indica si se va a agrupar las trazas para reducir su dimensionalidad	Si o No	No
Número de ataques de validación	Número de ataques que se realizan con el mejor individuo de cada iteración	1 hasta 10	3
Número de trazas para el ataque de validación	Número de trazas que se usan para cada ataque de validación	1 hasta 1000	Determinado por el Número de trazas para el ataque
Factor de corrección (FC)	Coeficiente que se multiplica con el valor de los individuos	1 hasta 100	10
Fórmula de evaluación de individuos	Indica que formula se usa para evaluar los individuos	$GE * FC$ o $GE * FC$ normalizado o $GE * FC$ con NUM_POI	$GE * FC$

Tabla 5.1: Parámetros fijos para las pruebas de evaluación

será el valor bajo, y el otro el valor alto. Esto hace que cada prueba tenga 8 experimentos. En la Tabla 5.3 encontramos como se hará la configuración de cada una de las pruebas, teniendo en cuenta que variables están en su valor bajo y alto.

Para obtener los coeficientes de cada una de las variables a prueba en cada una de las pruebas, se va a utilizar la ecuación 5.1. En esta ecuación, Train se refiere a la posición de la subclave correcta del mejor individuo en ataque, y Test se refiere a la posición de la subclave correcta del mejor individuo en validación. La validación tiene mayor peso, ya que es donde vemos si realmente el modelo generado consigue obtener la subclave correcta.

$$Coeficiente = -(0{,}25 * (Train/255) + 0{,}75 * (Test/255)) \tag{5.1}$$

Como se ha dicho antes, el criterio "Ok" se cumplirá cuando se consiga obtener la subclave del dispositivo en la 1ª posición. Esto se cumplirá cuando la posición de la subclave tanto en la fase de ataque como en la de validación sea la 1ª posición. Esto hace que puedan ocurrir 2 cosas:

- Si se consigue el criterio "Ok" antes de haber analizado todas las variables, se realizarán pruebas para analizar las variables pendientes, y ver si esas variables tienen un efecto positivo o no en el resultado.

Variable	Descripción	Rango
Número de trazas para el modelo	Indica el número de trazas que se usan para generar la plantilla	5000 hasta 50000
Número de trazas para el ataque	Indica el número de trazas que se usan en cada ataque	10 hasta 1000
Número de ataques	Número de ataques a realizar para obtener la evaluación de cada individuo	1 hasta 10
Tamaño de la población	Número de individuos a usar en cada iteración	10 hasta 50
Número de iteraciones	Número máximo de iteraciones que el EDA tendrá para llegar al objetivo	5 hasta 50
Trazas transformadas con PCA	Indica si las trazas que se van a usar son las originales o son las obtenidas usando PCA	Si o No
Modo de inicialización	Indica de qué forma se va a iniciar la población de los individuos para la 1º iteración	Aleatorio, usando correlación, a mano, primeros x, probabilidad descendente

Tabla 5.2: Parámetros a fijar en las pruebas de evaluación

Experimento	A	B	C
1	Bajo	Bajo	Bajo
2	Bajo	Bajo	Alto
3	Bajo	Alto	Bajo
4	Bajo	Alto	Alto
5	Alto	Bajo	Bajo
6	Alto	Bajo	Alto
7	Alto	Alto	Bajo
8	Alto	Alto	Alto

Tabla 5.3: Configuración de experimentos

- Si no se consigue el criterio “Ok” antes de haber analizado todas las variables, se realizará una nueva iteración, variando los valores de las variables o añadiendo variables nuevas de la lista, para ver así el efecto en el resultado y si finalmente se consigue el criterio “Ok”.

5.3.1. Evaluando el dispositivo Piñata

Para evaluar el dispositivo Piñata, se ha utilizado la herramienta desarrollada para realizar una evaluación, utilizando las trazas de la implementación sin protección de la misma.

1º prueba

Para esta prueba, se ha propuesto evaluar los parámetros con sus correspondientes valores que encontramos en la Tabla 5.4. El campo de identificador es para identificarlos en los resultados. El resto de parámetros han sido fijados con los valores de la Tabla 5.5.

ID	Variable	Valor bajo	Valor alto
A	Número de trazas para el modelo	10000	20000
B	Número de trazas para el ataque	10	100
C	Número de ataques	1	3

Tabla 5.4: Parámetros a prueba

Variable	Valor
Tamaño de la población	30
Número de iteraciones	10
Trazas transformadas con PCA	No
Modo de inicialización	Aleatorio

Tabla 5.5: Parámetros fijados

Esta prueba se ha repetido 3 veces con los 8 experimentos diferentes que se han realizado. En la Figura 5.9 se muestran los parámetros usando el identificador con el efecto que generan en el resultado del análisis. Como podemos observar, las 3 variables generan un efecto positivo al utilizar el valor alto, siendo las trazas para el ataque las que más beneficio ofrecen. También podemos observar que los efectos son mejores cuando las variables se intercalan, es decir, cuando no son los dos valores en alto o bajo. Además,

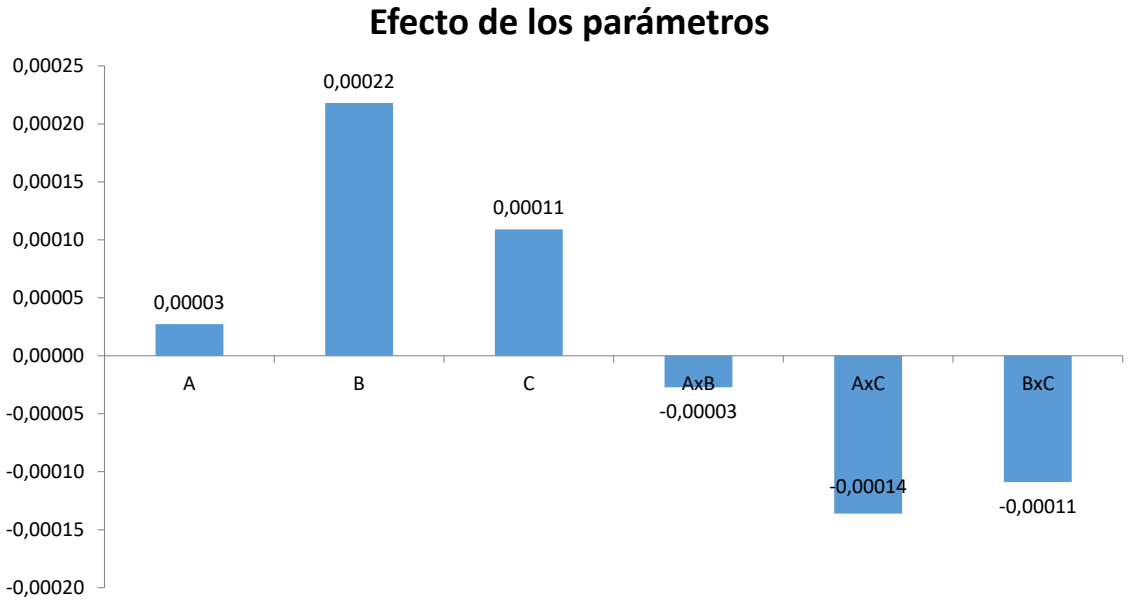

Figura 5.9: Resultados 1ª Prueba

durante esta prueba se ha llegado al criterio "Ok" establecido, ya que en la gran mayoría de los resultados se ha obtenido la subclave correcta, como podemos ver en la Tabla 5.6. De hecho, encontramos que existen varias configuraciones que en las 3 repeticiones obtienen la subclave correcta. En la Figura 5.10 podemos ver la posición de la subclave correcta en la fase de validación de la prueba 8, ronda 3. En él, encontramos 4 líneas graficadas, donde los colores azul, amarillo y verde son los ataques, y el color rojo, la media de los 3 ataques. Como podemos observar, a medida que aumentamos las trazas para el ataque, la posición va descendiendo a 0, llegando a 0 antes de las 100 trazas. Por lo tanto, podemos pasar a evaluar la Discovery.

Experimento	Ronda 1	Ronda 2	Ronda 3	Puntuación media
1	-0,00098	-0,00196	0	-0,00098
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	-0,00098	0	0	-0,00033
6	-0,00033	-0,00065	-0,00033	-0,00044
7	0	0	0	0
8	0	0	0	0

Tabla 5.6: Resultados de la 1ª prueba

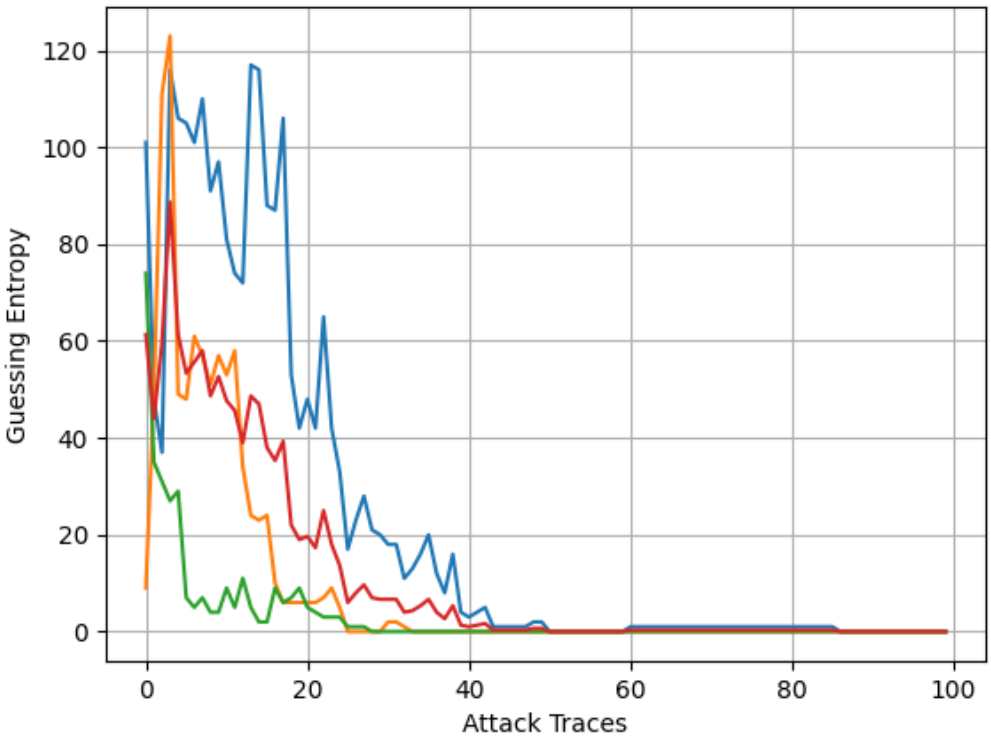

Figura 5.10: Posición de la subclave correcta prueba 8

ID	Variable	Valor bajo	Valor alto
A	Número de trazas para el modelo	10000	20000
B	Número de trazas para el ataque	10	100
C	Número de ataques	1	3

Tabla 5.7: Parámetros a prueba

Variable	Valor
Tamaño de la población	30
Número de iteraciones	10
Trazas transformadas con PCA	No
Modo de inicialización	Aleatorio

Tabla 5.8: Parámetros fijados

5.3.2. Evaluando el dispositivo Discovery

Para evaluar el dispositivo Discovery, se ha utilizado la herramienta desarrollada para realizar una evaluación, utilizando las trazas de la implementación sin protección de la misma.

1º prueba

Para esta prueba, se ha propuesto evaluar los parámetros con sus correspondientes valores que encontramos en la Tabla 5.7. En este caso, los valores coinciden con los de la 1ª prueba de la Piñata, ya que el objetivo es ver si con diferentes dispositivos obtenemos resultados similares. El resto de parámetros han sido fijados con los valores de la Tabla 5.8.

Esta prueba se ha repetido 3 veces con los 8 experimentos diferentes que se han realizado. En la Figura 5.11 se muestran los parámetros usando el identificador con el efecto que generan en el resultado del análisis. Como podemos observar, las 3 variables generan un efecto negativo al utilizar el valor alto, siendo las trazas para el ataque las que menos beneficio ofrecen. También vemos que, cuando 2 variables coinciden en tipo de valor, el resultado mejora. Estos resultados muestran un efecto completamente inverso a los obtenidos en el caso del dispositivo de la Piñata, y aunque no se puede asegurar, puede haber sido debido a la aleatoriedad al generar la población inicial. Además, como podemos ver en la Tabla 5.9, en ningún caso se ha llegado al criterio “Ok” establecido. Por otro lado,

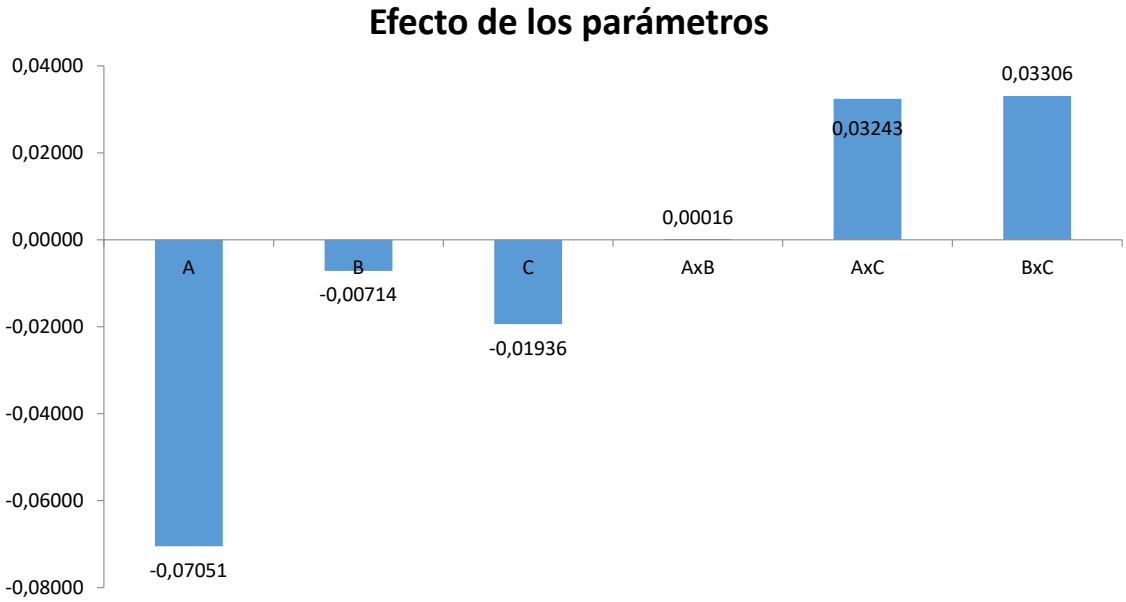

Figura 5.11: Resultados 1ª Prueba

los resultados de la Tabla 5.9, indican una alta variabilidad, por lo que los resultados no son concluyentes. Esto puede ser debido a que el número de trazas para el ataque no es suficiente, lo que perturba los resultados de la prueba. Por lo tanto, tenemos que seguir evaluando la Discovery.

Experimento	Ronda 1	Ronda 2	Ronda 3	Puntuación media
1	-0,2	-0,23922	-0,25098	-0,23007
2	-0,53562	-0,19379	-0,40817	-0,37919
3	-0,23333	-0,32843	-0,30882	-0,2902
4	-0,34379	-0,41438	-0,28660	-0,34826
5	-0,49216	-0,39608	-0,35882	-0,41569
6	-0,30588	-0,56373	-0,55915	-0,47625
7	-0,53627	-0,42353	-0,58922	-0,51634
8	-0,43889	-0,28758	-0,48399	-0,40349

Tabla 5.9: Resultados de la 1ª prueba

ID	Variable	Valor bajo	Valor alto
A	Número de trazas para el modelo	10000	20000
B	Número de trazas para el ataque	100	1000
C	Número de ataques	1	3

Tabla 5.10: Parámetros a prueba

Variable	Valor
Tamaño de la población	30
Número de iteraciones	5
Trazas transformadas con PCA	No
Modo de inicialización	Aleatorio

Tabla 5.11: Parámetros fijados

2º prueba

Para esta prueba, se ha propuesto evaluar los mismos parámetros que en la prueba anterior, pero modificando los valores. Esto lo podemos ver en la Tabla 5.10. Este cambio se realiza para confirmar que los resultados anteriores se deben a un número de trazas de ataque insuficiente. El resto de parámetros han sido fijados con los valores de la Tabla 5.11. Para esta prueba en concreto, se ha cambiado el número máximo de iteraciones, para intentar reducir el tiempo de ejecución de las pruebas, y también para quitar peso a una variable que se estudiara más adelante.

Esta prueba se ha repetido 3 veces con los 8 experimentos diferentes que se han realizado. En la Figura 5.12 se muestran los parámetros usando el identificador con el efecto que generan en el resultado del análisis. Como podemos observar, las 3 variables generan un efecto positivo al utilizar el valor alto, siendo el número de ataques el que más beneficio ofrece. Estos resultados muestran un efecto completamente inverso a los obtenidos en la prueba anterior, por lo que podemos confirmar que se necesitan más trazas para el ataque. También vemos que, cuando las variables A y B se intercalan en valor, el resultado empeora ligeramente, mientras que en el caso de A y C o B y C el resultado mejora cuando los valores coinciden en tipo, siendo este último el más notable. Vemos que el caso de que las variables A y B obtienen peor resultado, pero puede deberse a la aleatoriedad, y no es representativo. Sin embargo, el caso de que B y C tengan el mismo tipo de valor, y viendo que ambos obtienen un efecto positivo en el valor alto, está claro que a partir de ahora podemos fijar dichos valores. En el caso de la variable A también lo podemos hacer,

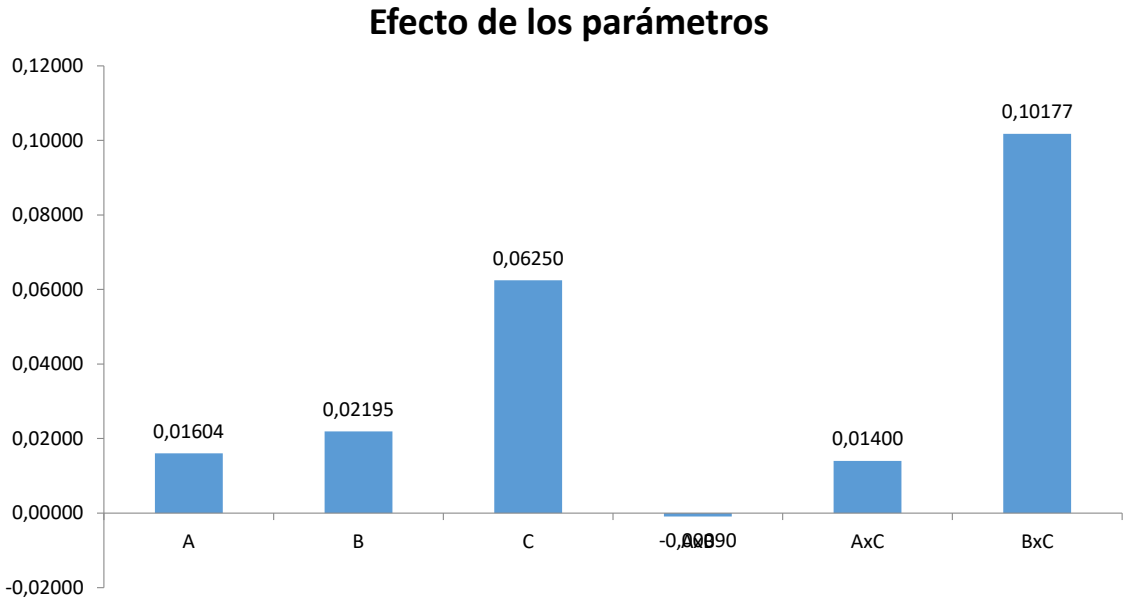

Figura 5.12: Resultados 2ª Prueba

aunque queda claro que su impacto no va a ser tan relevante. Aun así, en la Tabla 5.12 podemos ver que en ningún caso se ha llegado al criterio "Ok" establecido. Por lo tanto, tenemos que seguir evaluando la Discovery.

Experimento	Ronda 1	Ronda 2	Ronda 3	Puntuación media
1	-0,2667	-0,2137	-0,3196	-0,2667
2	-0,4154	-0,4477	-0,4056	-0,4229
3	-0,4353	-0,4853	-0,5020	-0,4742
4	-0,0487	-0,3176	-0,0056	-0,1240
5	-0,1471	-0,4941	-0,2902	-0,3105
6	-0,3588	-0,2935	-0,2817	-0,3113
7	-0,6343	-0,1422	-0,4902	-0,4222
8	-0,2444	-0,0863	-0,0154	-0,1154

Tabla 5.12: Resultados de la 2ª prueba

3º prueba

Para esta prueba, se ha propuesto evaluar nuevos parámetros que podemos ver en la Tabla 5.13. Este cambio se realiza para ver qué efecto tienen estos nuevos parámetros. El resto de parámetros han sido fijados con los valores de la Tabla 5.14. Como se ha mencionado en la prueba anterior, se han fijado los parámetros ya probados con sus respectivos valores.

ID	Variable	Valor bajo	Valor alto
A	Tamaño de la población	10	30
B	Número de iteraciones	5	10
C	Usar PCA	No	Si

Tabla 5.13: Parámetros a prueba

Esta prueba se ha repetido 3 veces con los 8 experimentos diferentes que se han realizado. En la Figura 5.13 se muestran los parámetros usando el identificador con el efecto que generan en el resultado del análisis. Como podemos observar, las 3 variables generan un efecto positivo al utilizar el valor alto, siendo la variable de usar PCA (variable C) el que más beneficio ofrece. Sin embargo, cuando vemos como interactúan las 3 variables, vemos que cuando la variable C está intercalada con las demás obtenemos mejor resultado, mientras que la variable A y B obtienen mejor resultado cuando tienen el mismo tipo de valor. Aun así, en la Tabla 5.15 podemos ver que en ningún caso se ha llegado al criterio "Ok" establecido. Por lo tanto, tenemos que seguir evaluando la Discovery.

Variable	Valor
Número de trazas para el modelo	20000
Número de trazas para el ataque	1000
Número de ataques	3
Modo de inicialización	Aleatorio

Tabla 5.14: Parámetros fijados

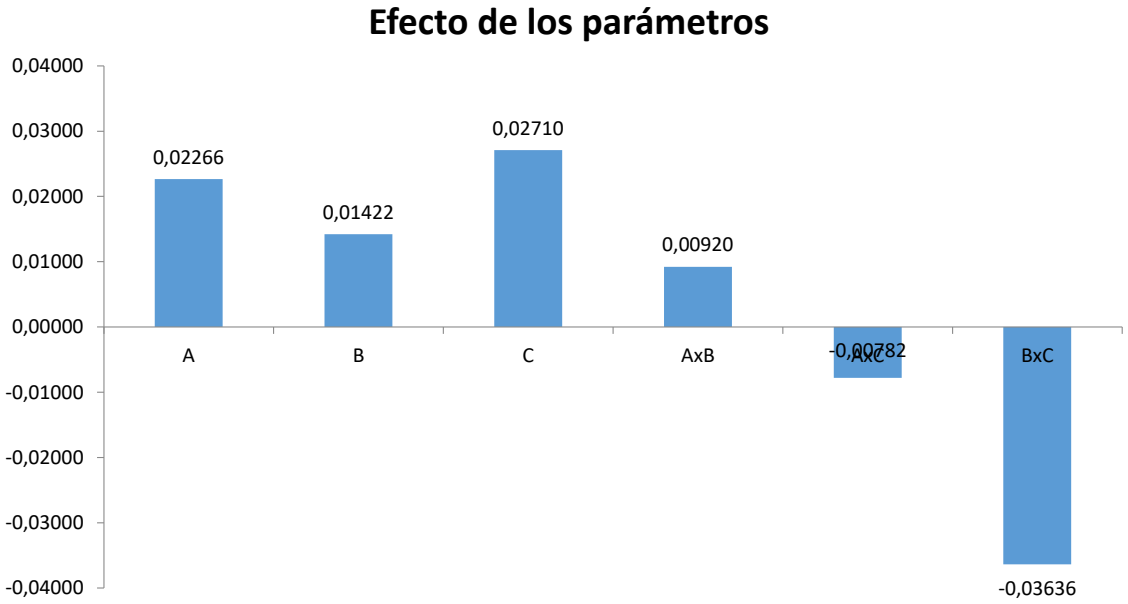

Figura 5.13: Resultados 3ª Prueba

Experimento	Ronda 1	Ronda 2	Ronda 3	Puntuación media
1	-0,39314	-0,41961	-0,51797	-0,44357
2	-0,17908	-0,55817	-0,35752	-0,36492
3	-0,32876	-0,51242	-0,43301	-0,42473
4	-0,47157	-0,27288	-0,34673	-0,36373
5	-0,61242	-0,53987	-0,24248	-0,46492
6	-0,40980	-0,21471	-0,24477	-0,28976
7	-0,13693	-0,39641	-0,31111	-0,28148
8	-0,52451	-0,24771	-0,36634	-0,37952

Tabla 5.15: Resultados de la 3ª prueba

4º prueba

Para esta prueba, se ha propuesto evaluar los parámetros que podemos ver en la Tabla 5.16. Se ha decidido cambiar la variable PCA y el de las iteraciones por las de trazas de ataque y tipo de inicialización. Este cambio se ha hecho por varias razones. La primera, porque no queda claro si PCA mejora o no los resultados con lo visto anteriormente. La segunda, porque aún queda la variable de inicialización por evaluar. Y la tercera, cambiar el número de iteraciones para volver a probar las trazas para el ataque es para comprobar

si con una inicialización diferente y con mayor tamaño de población se puede llegar al criterio “Ok”. El resto de parámetros han sido fijados con los valores de la Tabla 5.17. Se han fijado los parámetros ya probados con sus respectivos valores.

ID	Variable	Valor bajo	Valor alto
A	Tamaño de la población	30	50
B	Número de trazas para el ataque	100	1000
C	Inicialización	Aleatoria	Correlación

Tabla 5.16: Parámetros a prueba

Variable	Valor
Número de trazas para el modelo	20000
Número de ataques	3
Número de iteraciones	10
Trazas transformadas con PCA	No

Tabla 5.17: Parámetros fijados

Esta prueba se ha repetido 3 veces con los 8 experimentos diferentes que se han realizado. En la Figura 5.14 se muestran los parámetros usando el identificador con el efecto que generan en el resultado del análisis. Como podemos observar, las 3 variables generan un efecto positivo al utilizar el valor alto, siendo la variable de Número de trazas para el ataque (variable B) el que más beneficio ofrece. También vemos que la variable del tamaño de población (variable A) no ofrece apenas algún beneficio. Sin embargo, cuando vemos como interactúan las 3 variables, vemos que cuando la variable C está intercalada con las demás obtenemos mejor resultado, mientras que la variable A y B obtienen mejor resultado cuando tienen el mismo tipo de valor. Además, en la Tabla 5.15 podemos ver que hay varios casos donde se ha conseguido el criterio “Ok” establecido. De hecho, vemos que de las 3 pruebas donde se ha llegado al criterio “Ok” 2 son usando correlación, y además, existen otras 3 pruebas donde la subclave correcta se queda muy cerca (entre las 3 primeros posiciones) de llegar al criterio “Ok” cuando se usa correlación. Por lo tanto, que la interacción de la variable C con las demás variables sea negativa puede deberse a que algunas pruebas han obtenido peor resultado y la media sea menor, aún y cuando realmente está claro que ha ayudado a lograr el criterio “Ok”.

En la Figura 5.15 podemos ver la posición de la subclave correcta en la fase de validación de la prueba 8, ronda 3. En él, encontramos 4 líneas graficadas, donde los colores azul, amarillo y verde son los ataques, y el color rojo, la media de los 3 ataques. Como podemos

observar, en 1 de los 3 ataques le cuesta más llegar a 0, pero aun así consigue llegar a 0 antes de 1000 trazas. Por lo tanto, ya podemos dejar de seguir evaluando la Discovery.

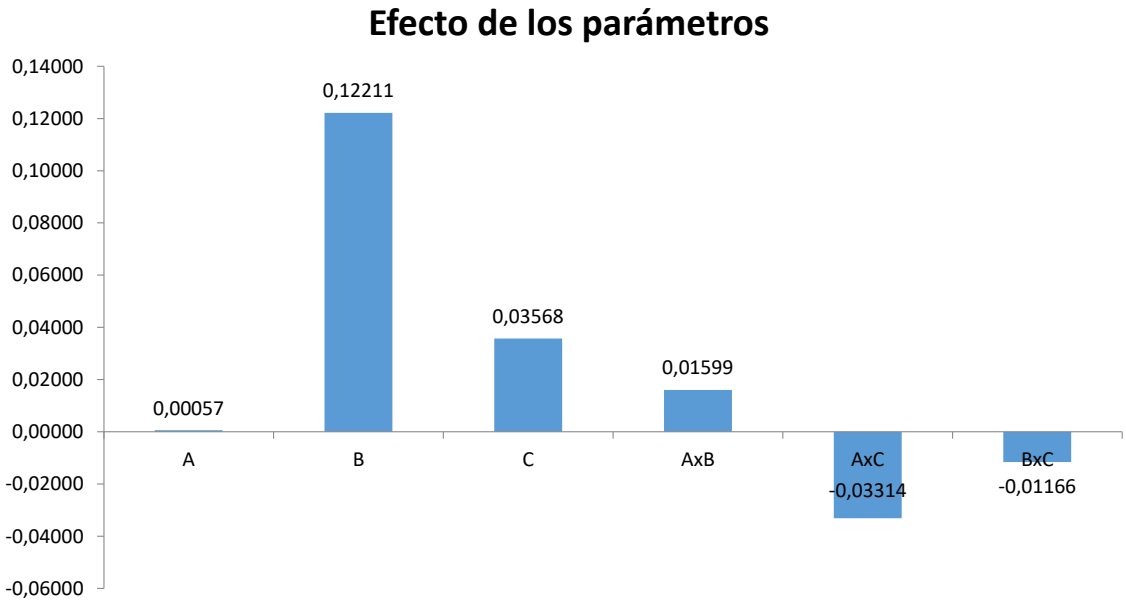

Figura 5.14: Resultados 4ª Prueba

Experimento	Ronda 1	Ronda 2	Ronda 3	Puntuación media
1	-0,43693	-0,11275	-0,51176	-0,35381
2	-0,19706	-0,14216	-0,23889	-0,19270
3	-0,17549	-0,13758	-0,04118	-0,11808
4	-0,00686	0	-0,00490	-0,00392
5	-0,29346	-0,33562	-0,32549	-0,31819
6	-0,26307	-0,26503	-0,34183	-0,28998
7	-0,01732	0	-0,03922	-0,01885
8	0	-0,00784	-0,10294	-0,03693

Tabla 5.18: Resultados de la 4ª prueba

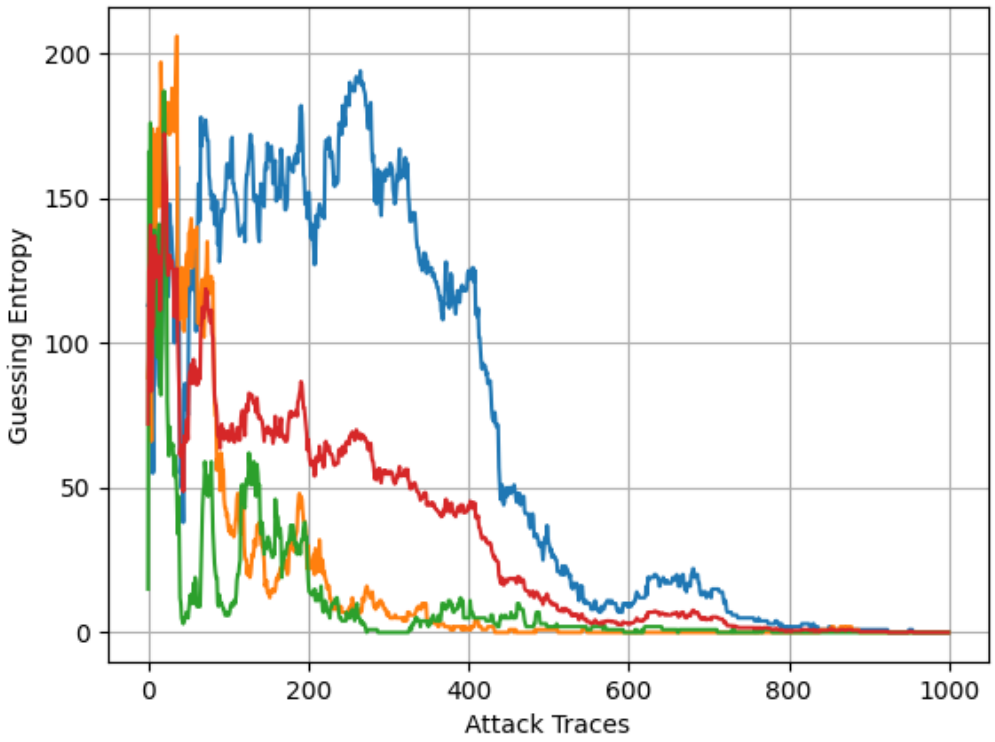

Figura 5.15: Posición de la subclave correcta prueba 8

5.3.3. Evaluando dispositivos con contramedidas

Una vez que hemos conseguido ver que ambos dispositivos son vulnerables cuando no usan ningún mecanismo de protección ante SCA, queremos probar si es posible atacar a ambos dispositivos cuando usan la contramedida de enmascaramiento. En este caso, vamos a usar las trazas del enmascaramiento tipo 1 (ver Sección 5.1.1) para comprobar si la herramienta también puede obtener la subclave correcta ante contramedidas.

Piñata

Para comprobar si la Piñata es susceptible a SCA, se va a utilizar la misma configuración que ha conseguido obtener la subclave en la fase de evaluación. En la Tabla 5.19 podemos ver los valores usados en diferentes parámetros. Se ha decidido usar los valores altos de la prueba con trazas sin protección porque debería ser algo más complicado obtener la

Variable	Valor
Número de trazas para el modelo	20000
Número de trazas para el ataque	100
Número de ataques	3
Tamaño de la población	30
Número de iteraciones	10
Trazas transformadas con PCA	No
Modo de inicialización	Aleatorio

Tabla 5.19: Parámetros utilizados

subclave en esta prueba, y porque dicha configuración tiene un éxito de obtener la clave en las 3 rondas que se han hecho.

En la Figura 5.16 podemos ver el gráfico con la posición de la subclave correcta al realizar esta prueba. Como podemos ver, todos los ataques posicionan la subclave correcta en la posición 0 desde las 60 trazas de ataque, por lo que vemos que se ha conseguido obtener la subclave del dispositivo de la Piñata con la configuración utilizada. Como información añadida, esta prueba ha necesitado 168 segundos para ejecutarse, y solo ha necesitado 1 iteración.

Discovery

Ahora, vamos a comprobar si la Discovery también es susceptible a SCA. Para ello, vamos a utilizar la misma configuración que ha conseguido obtener la subclave en la fase de evaluación. Como en este caso había varias opciones que habían conseguido la clave, se ha optado por usar los valores altos de trazas de ataque y modo de inicialización, ya que son las configuraciones que la mayoría de éxitos en la prueba con el dataset sin protección. En la Tabla 5.20 podemos ver los valores usados en diferentes parámetros.

En la figura 5.17 podemos ver el gráfico con la posición de la subclave correcta al realizar esta prueba. Como podemos ver, todos los ataques posicionan la subclave correcta en la posición 0 un poco antes de usar 1000 trazas de ataque, es decir, que se ha conseguido obtener la subclave con la configuración utilizada. También podemos ver que ha sido más difícil obtener la subclave, ya no solo porque con 100 trazas ningún ataque la posiciona en la posición 0, sino porque con menos de 200 trazas solo 1 ataque la posiciona por debajo de la posición 25. Como información añadida, esta prueba ha necesitado 20 minutos y 17 segundos para ejecutarse, y solo ha necesitado 1 iteración.

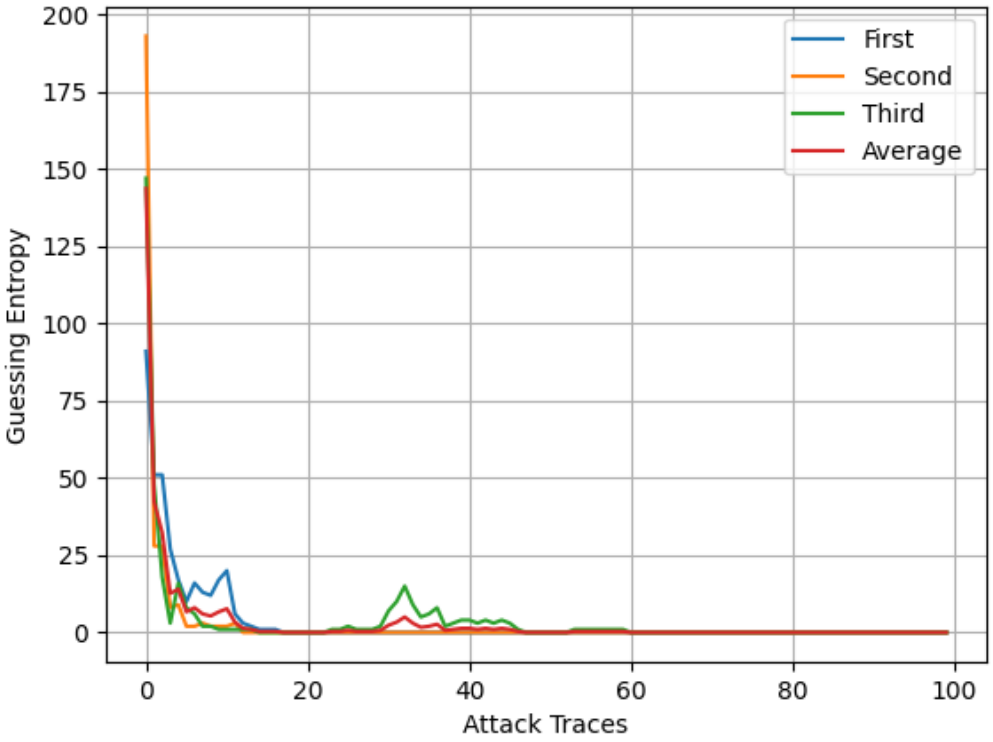

Figura 5.16: Posición de la subclave correcta Piñata con contramedida

Variable	Valor
Número de trazas para el modelo	20000
Número de trazas para el ataque	1000
Número de ataques	3
Tamaño de la población	30
Número de iteraciones	10
Trazas transformadas con PCA	No
Modo de inicialización	Correlación

Tabla 5.20: Parámetros utilizados

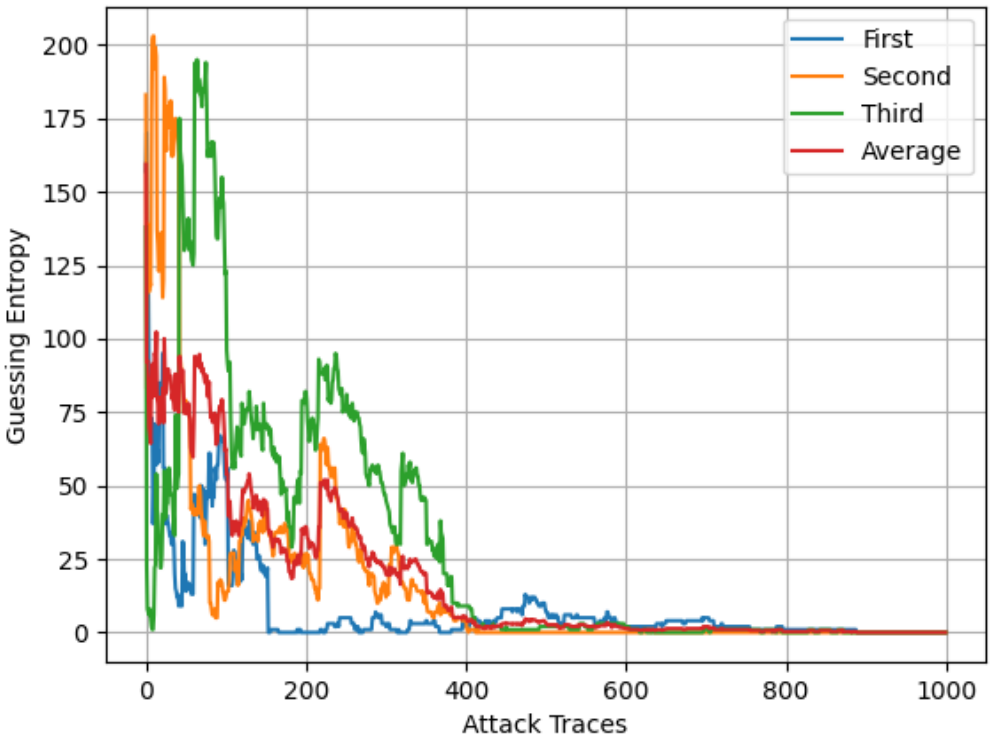

Figura 5.17: Posición de la subclave correcta Discovery con contramedida

5.3.4. Conclusiones de las pruebas

A continuación, se presentan las conclusiones resultantes tras haber llevado a cabo el presente proyecto. Por un lado, se exponen las conclusiones obtenidas tras las pruebas de rendimiento y, por otro lado, las conclusiones resultantes de las pruebas de análisis realizadas, incluyendo las de pruebas ante contramedidas.

Comenzando con las conclusiones relativas a las pruebas de rendimiento, podemos destacar tres aspectos clave. El primero sería que el programa en paralelo escala mejor que el programa en serie, como hemos podido observar en la prueba de los diferentes puntos de interés, donde la diferencia de tiempos entre serie y paralelo es de casi 10 veces más con el máximo número de puntos de interés probado. La segunda es que en ninguna de las pruebas de rendimiento realizado se ha conseguido un factor de aceleración igual al número de hilos cuando usamos más de 4 hilos. En condiciones normales, esto podría deberse a que, o bien el programa no escala bien al aumentar el número de hilos, o bien que la paralelización no es correcta. Sin embargo, en los resultados vemos como con 12 hilos

obtenemos un factor de aceleración superior que con 5, 6, 7, 8, 9, 10 y 11 hilos. El tercer aspecto sería que, aunque el tiempo de ejecución no se reduce en proporción al número de hilos usado con la ejecución completa, sí que podemos observar una diferencia notable que haga que merezca la pena usar la versión paralela frente a la versión serie.

Siguiendo, en este caso, con las conclusiones obtenidas tras las pruebas de análisis realizadas, podemos sacar dos conclusiones clave. La primera es que ambos dispositivos, tanto la Piñata como la Discovery, son susceptibles a ataques de canal lateral cuando no usan ningún mecanismo de protección frente a estos. De hecho, como el dispositivo de la Piñata es un dispositivo preparado para ello, es bastante más sencillo y requiere de menos tiempo para obtener la clave de dicho dispositivo frente a la Discovery, donde se han realizado varias pruebas más para poder ver que es vulnerable a este tipo de ataques. La segunda es que ambos dispositivos también son sensibles a ataques de canal lateral cuando usan un mecanismo de protección de enmascaramiento incorrectamente implementado, dando una falsa sensación de seguridad. De hecho, se ha podido observar que la misma configuración sirve para obtener la clave frente al dispositivo con protección y sin protección. Esto evidencia la importancia de comprobar que los mecanismos de protección están correctamente implementados y que son eficaces.

6. CAPÍTULO

Conclusiones y trabajo futuro

El Análisis de Canal Lateral ha sido un área investigada durante años, y se ha podido ver su lo potentes que pueden llegar a ser. Sin embargo, estos ataques requieren de mucho esfuerzo humano, haciendo que no siempre se lleguen a realizar correctamente.

Con el objetivo de mitigar esta dependencia, y, por lo tanto, optimizar el proceso, Ikerlan quiere desarrollar una herramienta capaz de realizar este tipo de ataques mediante Algoritmos de Estimación de la Distribución, para eliminar la mayor parte de la interacción humana durante el proceso de evaluación de los dispositivos. Aunque originalmente ya existía una herramienta que integraba el algoritmo, este no era apto para realizar las evaluaciones, puesto que era una herramienta que no realizaba ningún análisis de canal lateral.

Siendo el objetivo principal de este proyecto es el desarrollar la herramienta, se ha podido ver que la integración de los Análisis de Canal Lateral en la herramienta original ha sido un éxito, ya que se ha visto que el funcionamiento del mismo es válido para evaluar dispositivos. Además, se han propuesto diferentes mejoras a la idea original, para ayudar así al proceso de evaluación de dispositivos. Estas mejoras incluyen todo tipo de utilidades y funciones añadidas, que facilitan el análisis, como diferentes métodos de inicialización de la población, incluir una fase de validación para comprobar que los mejores individuos son modelos generalizados o detener el proceso iterativo cuando el algoritmo ha convergido, es decir, cuando todos los individuos son equivalentes. También se ha propuesto una mejora para el tiempo de ejecución, y es que la versión paralela desarrollada necesita 7 veces menos tiempo en comparación con la implementación original, como se ha podido

ver en las pruebas realizadas.

Por otro lado, otro de los objetivos era analizar la seguridad de ciertos dispositivos frente a Análisis de Canal Lateral. Como se ha podido ver en las pruebas, los dispositivos evaluados se muestran vulnerables, y se ha conseguido recuperar la clave, validando así el correcto funcionamiento de la herramienta.

Sin embargo, aunque el objetivo de Ikerlan se ha cumplido exitosamente, la herramienta desarrollada tiene varios aspectos donde es posible ampliar el objetivo inicial. Como se ha podido comprobar, usar los algoritmos de estimación de la distribución ayudan a reducir en gran medida la dependencia humana. Aun así, en las pruebas realizadas se ha podido ver como sigue habiendo una dependencia a la hora de seleccionar los parámetros y sus valores.

Además de esto, también hemos podido ver que, aunque la paralelización ha ayudado mucho en los tiempos de ejecución, aún no es perfecta. Es por esto que se podría probar otras librerías para paralelizar la herramienta, y en vez de hacer que se ejecuten las evaluaciones de los individuos en paralelo, paralelizar la función de evaluación de los individuos.

Otro de los aspectos donde se puede ampliar o mejorar es en el algoritmo de estimación de la distribución. Y es que el algoritmo previamente implementado no ha sido modificado, y al ser muy básico, es posible que se pueda intentar mejorar. Entre estas mejoras podemos encontrar la selección de mejores candidatos, ya que, en vez de hacerlo con un número fijo (como la mitad de la población) se podría hacer en función de las puntuaciones de los individuos, o evaluar los mejores individuos de nuevo en la fase de validación, y entre ellos, los que mejor puntuación obtienen, o también añadir el porcentaje de individuos a seleccionar como un parámetro a ajustar, para ver cuál obtiene mejores resultados.

En cuanto a los dispositivos evaluados, también podemos extender el objetivo fijado. Y es que en este proyecto solo se han utilizado las trazas obtenidas de los consumos de energía de los dispositivos. Es por ello que se podría intentar trabajar en 2 aspectos, el primero, desarrollar también la parte de adquisición de trazas de los dispositivos, y así no depender de los dispositivos medidos en dataset públicos, y el segundo, explorar otro tipo de trazas de los dispositivos, como puede ser el de los campos electromagnéticos.

Bibliografía

[Backes et al., 2010] Backes, M., Dürmuth, M., Gerling, S., Pinkal, M., Sporleder, C., et al. (2010). Acoustic side-channel attacks on printers. In *USENIX Security symposium*, volume 9, pages 307–322.

[Bubberman et al., 2020] Bubberman, W., Karayalcin, S., Meester, M., Braakman, O., and Picek, S. (2020). Side-channel Analysis Toolbox. https://github.com/AISyLab/side-channel-analysis-toolbox.

[Chen and Zhou, 2006] Chen, Z. and Zhou, Y. (2006). Dual-rail random switching logic: A countermeasure to reduce side channel leakage. In Goubin, L. and Matsui, M., editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 242–254, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Dworkin, 2001] Dworkin, M. J. (2001). Recommendation for block cipher modes of operation :.

[Evans and Brown, 2001] Evans, D. and Brown, K. (2001). Fips 197 federal information processing standards publication advanced encryption standard (aes).

[Hutter and Schmidt, 2014] Hutter, M. and Schmidt, J.-M. (2014). The temperature side channel and heating fault attacks. In Francillon, A. and Rohatgi, P., editors, *Smart Card Research and Advanced Applications*, pages 219–235, Cham. Springer International Publishing.

[Kocher et al., 1999] Kocher, P., Jaffe, J., and Jun, B. (1999). Differential power analysis. In Wiener, M., editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Kocher, 1996] Kocher, P. C. (1996). Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Koblitz, N., editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg. Springer Berlin Heidelberg.

[OpenSSL Foundation, 2023] OpenSSL Foundation, I. (2023). /news/vulnerabilities-1.1.1.html.

[Popp and Mangard, 2006] Popp, T. and Mangard, S. (2006). Implementation aspects of the dpa-resistant logic style mdpl. In *2006 IEEE International Symposium on Circuits and Systems*, pages 4 pp.–2916.

[Quisquater and Samyde, 2001] Quisquater, J.-J. and Samyde, D. (2001). Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In Attali, I. and Jensen, T., editors, *Smart Card Programming and Security*, pages 200–210, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Sandrine, 2010] Sandrine (2010). Metodología dmaic six sigma - caletec.

[Santana et al., 2008] Santana, R., LarraÑaga, P., and Lozano, J. A. (2008). Protein folding in simplified models with estimation of distribution algorithms. *IEEE Transactions on Evolutionary Computation*, 12(4):418–438.

[Schlösser et al., 2012] Schlösser, A., Nedospasov, D., Krämer, J., Orlic, S., and Seifert, J.-P. (2012). Simple photonic emission analysis of aes. In Prouff, E. and Schaumont, P., editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 41–57, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Tang et al., 2017] Tang, W., Jia, S., and Wang, Y. (2017). Dual-voltage single-rail dynamic dpa-resistant logic based on charge sharing mechanism. *Chinese Journal of Electronics*, 26(5):899–904.

[Timon, 2019] Timon, B. (2019). Non-profiled deep learning-based side-channel attacks with sensitivity analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):107–131.

[urioja, 2022] urioja (2022). Github - urioja/aesptv2: Improved version of the aespt dataset.

[Wild et al., 2015] Wild, A., Moradi, A., and Güneysu, T. (2015). Evaluating the duplication of dual-rail precharge logics on fpgas. In Mangard, S. and Poschmann, A. Y.,

editors, *Constructive Side-Channel Analysis and Secure Design*, pages 81–94, Cham. Springer International Publishing.

[yao Wang et al., 2013] yao Wang, S., Wang, L., Liu, M., and Xu, Y. (2013). An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *International Journal of Production Economics*, 145(1):387–396.