Universidad de Valencia



**Texto

Descripción generada automáticamente**

**Grado en Ingeniería Electrónica Industrial**

Trabajo de Fin de Grado

**Seguridad en Microcontroladores:**

**Análisis de Side-Channel Attacks y Contramedidas**

**Autor: ABDALAH EL KENNOUSSI**

**Tutor: José Rafael Lajara**

**Valencia, Octubre – 2023**

# DECLARACION DE AUTORÍA

**Yo, ABDALAH EL KENNOUSSI declaro la autoría del Trabajo Fin de Grado titulado “Seguridad en Microcontroladores: Análisis de *Side-Channel Attacks* y Contramedidas” y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual. El material no original que figura en este trabajo ha sido atribuido a sus legítimos autores.**

**Valencia, 1 de enero de 2023**

**Fdo:**

# RESUMEN

En los microcontroladores existen diversos tipos de ataques. Algunos se pueden clasificar como directos y hacen operaciones como un volcado de memoria por JTAG. Otros buscan mecanismos indirectos, son los *side-channel attacks*.

En este proyecto haremos el estudio de un tipo de ataques indirectos y propondremos unas contramedidas para evitarlos. Nos centraremos en el “*Power Analysis*” que generalmente son los ataques más conocidos por su eficacia a la hora de deducir contraseñas y códigos encriptados.

El primer paso es estudiar el funcionamiento del sistema “CHIPWHISPERER LITE” y entender su funcionamiento con los dos tipos de ataques. A partir de aquí se planteará y ejecutarán los dos tipos de ataques para ver cómo reacciona nuestro *Target*, que en este caso es el ARM CORTEX-M.

Una vez realizados los ataques y obtenidos los resultados se procederá a realizar un estudio de prevención en contra de los dos tipos de ataques y se pondrán a prueba para ver si surten efecto.

# AGRADECIMIENTOS

Quiero expresar mi sincero agradecimiento a todos los profesores por el apoyo y el gran trabajo y conocimientos aportados durante la carrera, pero en especial a José Rafael Lajara que ha sido mi tutor y apoyo durante la realización de este trabajo de fin de grado. Su experiencia y conocimientos en este sector me han servido de guía durante todo el proceso de investigación y redacción de mi tesis. Su compromiso con la excelencia académica y su disposición a invertir tiempo y esfuerzo en este proyecto me han ayudado a mejorar muchas cualidades en este sector y cada vez interesarme más.

Además, quiero agradecerle por su paciencia y disponibilidad para responder a mis preguntas, aclarar mis dudas y ofrecer sugerencias constructivas que han mejorado significativamente este proyecto.

Espero tener la oportunidad de mantener el contacto y continuar aprendiendo de usted en el futuro. Gracias una vez más por su apoyo inquebrantable.

# Contents

[DECLARACION DE AUTORÍA 3](#_Toc157788051)

[RESUMEN 5](#_Toc157788052)

[AGRADECIMIENTOS 7](#_Toc157788053)

[Contents 9](#_Toc157788054)

[1 Introducción 11](#_Toc157788055)

[1.1 Introducción 11](#_Toc157788056)

[1.2 Motivación 11](#_Toc157788057)

[1.3 Objetivos 12](#_Toc157788058)

[2 Materiales y métodos 15](#_Toc157788059)

[2.1 ChipWhiperer Lite 32 Bits con Target Device Incorporado 15](#_Toc157788060)

[2.2 Máquina Virtual de ChipWhisperer 18](#_Toc157788061)

[2.3 Chipwhisperer Lite 32 BITS 20](#_Toc157788062)

[2.4 Tipos de targets 20](#_Toc157788063)

[3 ESTUDIO TEÓRICO 23](#_Toc157788064)

[3.1 Tipos de side channel attacks 23](#_Toc157788065)

[3.1.1 Análisis de potencia (*Power Analysis*) 23](#_Toc157788066)

[3.1.2 Análisis de tiempo (*Clock Glitching* o *Timing Analysis*) 27](#_Toc157788067)

[3.1.3 Otros 27](#_Toc157788068)

[4 Trabajo práctico 29](#_Toc157788069)

[4.1 Ataques simples de Power analysis 29](#_Toc157788070)

[4.2 Ataques simples de Clock glitching 29](#_Toc157788071)

[4.3 Implementación y prueba de contramedidas para ataques de power analysis 29](#_Toc157788072)

[4.4 Implementación y prueba de contramedidas para ataques de clock glitching 29](#_Toc157788073)

[5 Conclusiones 31](#_Toc157788074)

[6 Bibliografía 33](#_Toc157788075)

[7 Anexos 35](#_Toc157788076)

[7.1 Código Fuente X 35](#_Toc157788077)

[7.2 Código Fuente Y 35](#_Toc157788078)

# Introducción

## Introducción

Los *Side Channel Attacks* son un tipo de ataque informático que explora las fugas involuntarias de información que se producen durante el funcionamiento de un sistema, un ordenador o cualquier dispositivo electrónico [1][2][3][4]. A diferencia de los ataques tradicionales, que se centran en vulnerabilidades en el software o hardware, este tipo de ataques se basan en la observación de señales indirectas o canales colaterales, como el consumo de energía, el tiempo de ejecución o las emisiones electromagnéticas para descifrar información encriptada, de ahí su nombre.

Estos ataques pueden tener como objetivo la obtención de datos confidenciales, como claves de cifrado, contraseñas o información personal, sin acceder directamente a la fuente de esos datos. Los *Side Channel Attacks* pueden ser especialmente peligrosos en entornos donde la seguridad de la información es crítica, como sistemas de criptografía, sistemas de pago electrónico o dispositivos médicos.

Los investigadores y expertos en ciberseguridad trabajan constantemente en la detección y mitigación de los *Side Channel Attacks*, desarrollando técnicas de contramedidas, como el enmascaramiento de datos, la introducción de ruido en las señales o el diseño de hardware resistente a este tipo de ataques. La comprensión de las amenazas asociadas a estos ataques es esencial para garantizar la integridad y la confidencialidad de los sistemas y datos en un mundo cada vez más conectado y dependiente de la tecnología.

Uno de los ataques más comunes y que además se va a tratar en este proyecto es: “*Power Analysis Attack*”, aunque teóricamente también vamos a aprovechar para explicar que son los “*Clock Glitching Attack*”.

Los ataques por análisis de potencia [5] se basan en que el atacante estudia el consumo de energía de un dispositivo hardware criptográfico para poder desencriptarlo. En cambio, para los ataques de reloj [6] podemos manipular el reloj que se presenta al dispositivo para provocar un comportamiento no deseado.

En general, los ataques por canales laterales se aprovechan de la implementación hardware de un sistema informático en algoritmos como el *Advanced Encrypton Standard* (AES) o el *Triple Data Ecryption Algorithm* (DEA) para explotarlos por medio de diferentes ataques laterales. Además, es común que estos ataques vayan dirigidos a microcontroladores.

## Motivación

Investigar el tema de los *Side Channel Attacks* es fundamental en el mundo actual de la ciberseguridad. Estas amenazas, que se basan en el análisis de información indirecta, representan un desafío constante y una oportunidad de mejora en la protección de datos y sistemas.

En la era digital en la que vivimos, la información es el recurso más valioso. La seguridad de esta información es esencial para proteger nuestras vidas, nuestra privacidad y nuestras instituciones. En este contexto, la investigación de los *Side Channel Attacks* se vuelve una tarea crítica.

Al comprender en profundidad cómo funcionan los *Side Channel Attacks* podemos desarrollar contramedidas efectivas para prevenirlos y proteger nuestros sistemas y datos. La investigación en esta área nos permite anticiparnos a los posibles agresores y fortalecer nuestras defensas.

Además, la investigación en *Side Channel Attacks* fomenta la innovación en la seguridad cibernética. A medida que los atacantes desarrollan nuevas técnicas, los investigadores también deben estar un paso adelante, diseñando sistemas más robustos y seguros. Esta constante competencia es lo que impulsa el avance de la tecnología y nos permite mantenernos a salvo en un mundo cada vez más interconectado.

En resumen, la investigación en *Side Channel Attacks* es crucial porque nos ayuda a proteger nuestra información, promueve la innovación en seguridad cibernética y puede tener un impacto positivo en muchas áreas de la tecnología. Al trabajar juntos para comprender y mitigar estas amenazas, podemos garantizar un futuro digital más seguro y confiable. Se espera que este trabajo sirva como base para otros futuros trabajos de investigación que avancen en nuevos conocimientos de este tipo de ataques y contramedidas en nuevos *targets*.

## Objetivos

En este TFG, nuestro objetivo es estudiar dos tipos de ataques laterales: el *Power Analysis Attack* y el *Clock Glitching* pero solo vamos a poner a prueba nuestro dispositivo con el *Power Analysis Attack.*

Para este estudio emplearemos el hardware CHIPWHISPERER LITE como herramienta de generación y análisis de ataques y nuestro *Target* víctima será un microcontrolador ARM CORTEX – M.

El objetivo principal es realizar los dos ataques utilizando nuestro micro y ver cómo reacciona nuestro *Target*. Una vez obtenidos los resultados, se estará en condiciones de proponer contramedidas y ponerlas a prueba para evitar este tipo de brechas.

Hay varios sub-objetivos, como son:

* Mejorar la seguridad informática: estudiar los *Side Channel Attacks* me permitirá comprender las vulnerabilidades de los sistemas y dispositivos electrónicos frente a técnicas de ataque que van más allá de los métodos convencionales. Al entender cómo se pueden explotar canales secundarios para obtener información confidencial, puedo desarrollar contramedidas más efectivas y fortalecer la seguridad de los sistemas.
* Evaluación de la resistencia de sistemas y dispositivos: Estudiar los *Side Channel Attacks* en este TFG me proporcionará una herramienta valiosa para evaluar la resistencia de sistemas y dispositivos a posibles amenazas.
* Protección de la privacidad: Estudiar los *Side Channel Attacks* es esencial para proteger la privacidad de los usuarios y en este caso poder mejorar mi propia seguridad en dispositivos propios.
* Estudio y evaluación con diferentes programas: Esto me permite el estudio de dichos ataques empleando varios programas diferentes con los que puedo realizar los análisis de todos los comportamientos a la hora de ejecutar dichos ataques.
* Desarrollar softwares de contramedida: La posibilidad de desarrollar nuevos softwares para poder evitar todo este tipo de ataques y poder implementarlos en el micro.

En general, el principal objetivo es entender todo el funcionamiento de este tipo de ataques laterales y poder implementar las contramedidas necesarias para poder evitar o atenuar ataques.

# Materiales y métodos

## ChipWhiperer Lite 32 Bits con Target Device Incorporado

El ChipWhisperer-Lite [7] representa la búsqueda más agresiva de NewAE Technology Inc. para proporcionar, de la forma más fácil y eficaz, el estudio de los ataques laterales, tanto los ataques de potencia como los ataques de *Glitching* de reloj a cualquier persona con conocimientos en el sector, de estudiantes a ingenieros de software. Es completamente de código abierto (hardware, software, firmware, código FPGA) y está proporcionando una revolución en la seguridad de hardware.

El ChipWhisperer-Lite integra hardware para realizar mediciones de análisis de potencia, programación de dispositivos, *Glitching*, comunicaciones serie y un ejemplo de objetivo que puede cargarse con algoritmos criptográficos, todo en una sola placa. La versión de placa única viene en dos variantes: Atmel XMEGA o STM32F3 como objetivo víctima.

Se puede separar la parte de captura y el *target* para conectarlos a otros objetivos. La parte de captura está disponible como un componente independiente y viene lista con conectores SMA y para el objetivo, pero requiere un objetivo externo.

**Especificaciones:**

**Tabla

Descripción generada automáticamente**

Ilustración 1: Especificaciones del Chipwhisperer [6]

Estas son las especificaciones que adjunta nuestra placa y podemos observar que, en este caso, los dos tipos de ataques disponibles son el de reloj y el de la potencia. Por otra parte, se puede observar cual es nuestro *Target Device* o víctima, varía según el tipo de placa, pero en una de ellas nos ofrece el STM32Fx, un microcontrolador ARM Cortex-M.

Por otra parte, se pueden observar los tiempos de medida que ofrece la placa con un *Offset* de 200 ps como mínimo. Tiene rangos de frecuencia de 5 a 200 MHz y una salida de reloj con salida de *Glitch* incorporada.

**Arquitectura síncrona:**

**Imagen de la pantalla de una computadora

Descripción generada automáticamente con confianza media**

Ilustración 2: Ejemplo de arquitectura síncrona vista en osciloscopio

ChipWhisperer puede utilizarse para inyectarse en la señal de reloj de un dispositivo objetivo y aplicar multiplicaciones y desplazamientos de fase para muestrear en el punto deseado(s) durante el ciclo de reloj. Esto asegura que los puntos de muestreo estén directamente relacionados con el reloj digital que genera las señales de interés.

**Tipos de placa:**

Un circuito electrónico

Descripción generada automáticamente con confianza media

Ilustración 3: Imagen de ChipWhisperer. A la izquierda la placa de generación y análisis de ataques, a la derecha la víctima

La compañía ofrece varias posibilidades a la hora de realizar la compra. En la Ilustración 3 se puede observar la parte de captura y el objetivo en una misma placa, pero cabe la posibilidad de realizar la compra por partes separadas si el objetivo que queremos atacar es diferente o si queremos incorporar osciloscopios externos u otros tipos de medidores.

En las siguientes imágenes podemos observar los diferentes tipos de pedidos que se pueden realizar:

Tipo 1:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Ilustración 4: ChipWhisperer single board

Este modelo es con el cual vamos a trabajar. Incorpora en la misma placa el objetivo y la placa de medida, todo interconectado entre sí. Es más útil si se tiene claro el tipo de ataques que se quieren realizar y el objetivo deseado, aunque te limita el acceso a otras funciones como el poder conectar objetivos externos diferentes al que ya tenemos incorporado en la placa. También nos limita el no poder conectar osciloscopios para visualizar el comportamiento sin tener que soldar o hacerlo manualmente desde la placa.

Tipo 2:

Texto

Descripción generada automáticamente

Ilustración 5: ChipWhisperer placa de ataque con conectores soldados

Este modelo solo incorpora la parte de captura por lo que se necesitaría comprar aparte un objetivo para atacar. Por eso incorpora cables y conectores para poder realizar las medidas y las conexiones necesarias.

Tipo 3:

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

Ilustración 6: ChipWhisperer, placa de ataque y víctima separadas y con conectores

Este modelo incorpora las dos partes anteriores, pero por separado. Es decir, tanto la placa de captura como el objetivo, pero en diferentes placas además de aportar cables de conexionado para poder interconectarlo todo.

En resumen, dependiendo del tipo de pruebas que se quieran realizar y el nivel de conocimiento que se tenga, se elige un tipo diferente de producto.

**Software:**

El ChipWhisperer ofrece un software incorporado que te permite realizar todo tipo de medidas y programación necesaria. ChipWhisperer es una cadena de herramientas de código abierto para la investigación en seguridad embebida. Todos los targets y los hardware de captura en este catálogo son compatibles con una aplicación de captura basada en Python. La naturaleza de código abierto significa que se pueden realizar modificaciones según necesidades específicas, ya sea desarrollando tus propios algoritmos o realizando validaciones en objetivos propietarios.

Actualmente ya no existe la versión visual de Windows, en la cual se tenía acceso a varios programas gráficos de captura y medida. Ahora todo es realizado basándonos en la programación en Python, lo cual nos permite tener aún más libertad y expansión a la hora de querer adaptarlo a nuestros requerimientos. Aunque, por otra parte, requiere más conocimientos y dedicación.

## Máquina Virtual de ChipWhisperer

Hay varias formas de realizar la instalación de todo lo necesario para poder ejecutar todas las funciones del Micro, pero existe la posibilidad de descargar la máquina virtual en la cual está incorporado todas las funciones y programas de captura y ataque para el ChipWhisperer.

En este caso, accederemos a la máquina virtual para poder acceder a los archivos, pero trabajaremos con Jupyter con acceso mediante un *host* desde nuestro navegador. Tiene directamente implementado un intérprete Python en el cual podemos ejecutar nuestros códigos para poder estudiar el comportamiento del micro.

En este TFG trabajaremos con VirtualBox y la extensión o la imagen de la computadora preconfigurada del ChipWhisperer con Jupyter, intérprete Python y todas las librerías necesarias.

Aquí explicaremos mediante una serie de imágenes cuál es la forma de acceder a la máquina virtual y cómo poder tener acceso a Jupyter.

Primero hay que introducir el usuario y la contraseña, que por defecto ambos son “vagrant”, ver Ilustración 7. Una vez accedemos nos indica que introduzcamos una contraseña nueva para el servidor de Jupyter.

Una vez creada la contraseña ya tendríamos acceso a Jupyter. Ahora hay que ver en la barra de la parte inferior de nuestra máquina virtual la IP de red a la que estamos conectados y copiarla (Ilustración 8). Introducir esta IP en nuestro navegador seguido por el puerto de Jupyter (8888). Al final, la URL quedaría algo parecido a XXX.XXX.XXX.XXX:8888 como en la Ilustración 9.

Accedemos a Jupyter e introducimos la contraseña que hemos definido en la máquina virtual. En este punto ya estaríamos dentro de Jupyter y tendríamos acceso a todos los ejemplos y archivos de chipwhisperer, tal y como se ve en la Ilustración 10.

* Primer paso:

Captura de pantalla de computadora

Descripción generada automáticamente

Ilustración 7: Máquina virtual de Chipwhisperer

* Segundo paso:



Ilustración 8: Icono de red de la máquina virtual

* Tercer paso:



Ilustración 9: IP y el host en nuestra barra de direcciones del navegador web

* Cuarto paso:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Ilustración 10: Panel principal de Jupyter desde el navegador web

Una vez realizados todos estos pasos, ya tendríamos acceso a los archivos de Jupyter y ya podríamos realizar las pruebas de iniciación para conectar nuestro microprocesador y comprobar que hay comunicación tanto a la placa de ataques como a la víctima.

En este caso, Jupyter es una plataforma interactiva en la que podemos acceder a los archivos de nuestro Chipwhisperer y poder programar y ejecutar los archivos tanto en código C como en Python. También existen varios test y tutoriales interactivos a los que se puede acceder para poder iniciarse en este mundo.

## Chipwhisperer Lite 32 BITS

El "ChipWhisperer Lite" es una versión específica del proyecto ChipWhisperer, que es un conjunto de herramientas de código abierto desarrolladas para realizar análisis de seguridad en dispositivos embebidos y sistemas integrados. La herramienta se centra en la detección y evaluación de ataques de canales laterales, como los ataques de análisis de potencia, que buscan explotar las fugas de información de consumo de energía para obtener datos confidenciales. Este es solo un ejemplo de los muchos otros tipos de ataques que se pueden realizar con esta herramienta y los cuales explicaremos con más detalles posteriormente.

ChipWhisperer Lite está diseñado para proporcionar una plataforma de bajo costo y fácil acceso para aquellos interesados en aprender y experimentar con ataques de canales laterales en sistemas embebidos.

Ofrece capacidades para realizar ataques de análisis de potencia y de reloj, así como otras técnicas de evaluación de seguridad. Incluye hardware y software que permiten a los usuarios interactuar con dispositivos integrados para evaluar su resistencia a ataques de canales laterales. Es un proyecto de código abierto, lo que significa que su hardware y software son accesibles al público [7].

## Tipos de targets

Existen varios *targets* con los cuales podemos adaptar nuestro ChipWhisperer para realizar diferentes tipos de ataques [8]. Normalmente según el tipo de ataques existen unos *targets* u otros, por ejemplo, en nuestro caso usaremos el ARM CORTEX-M.

Existen varios tipos de *targets*, citaremos los más usados o los más comunes:

* *Targets* Integrados: este tipo de targets normalmente vienen incorporados ya en la placa de captura, es decir, viene todo incorporado en la misma placa. Nosotros emplearemos el CW303 – ARM.

En la siguiente imagen os adjuntaremos los 3 *targets* integrados que existen.

Pantalla de un celular con letras

Descripción generada automáticamente

Ilustración 11: Diferentes Targets Integrados

* *Targets* Externos: este tipo de targets viene incorporado en una placa totalmente diferente a la de captura, por lo que tenemos dos placas diferentes las cuales pueden interactuar entre sí conectándolas mediante cableado y otro tipo de conexiones como (20-pin o cables SMA).

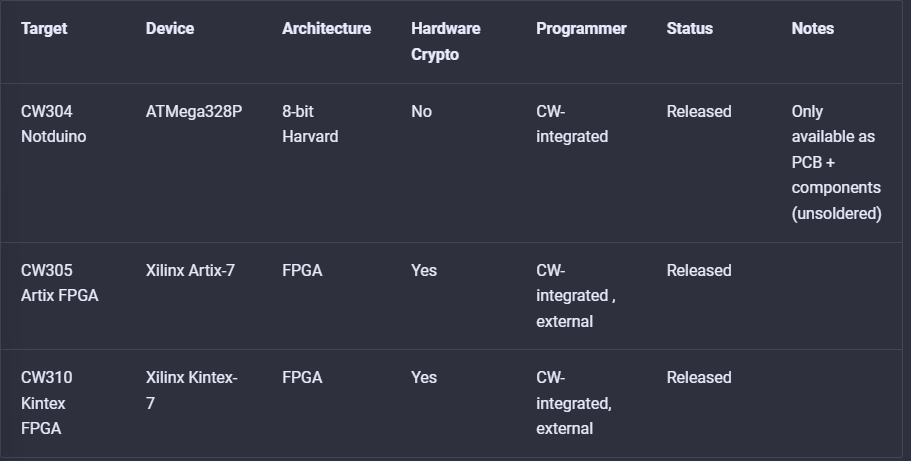


Ilustración 12: Diferentes Targets Externos

* *Targets* CW308 UFO: este tipo de *targets* suele ser empleado para atacar otros tipos de *targets*. Ofrecen muchas comodidades para alimentar otras placas *target* incluyendo reguladores de voltaje, circuitos de accionamiento cristal, selección de reloj, etc.

Estos son algunos de ellos, pero en [9] se pueden encontrar todos los diferentes targets.

32 bits

Pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza media

Ilustración 13: CW308 Targets 32 bits

8/16 bits



Ilustración 14: CW308 Targets 8/16bits

En conclusión, en función del tipo de uso que le queremos dar o qué tipo de ataques queremos realizar, utilizaremos una placa u otra o varias a la vez. Cabe destacar que normalmente, al ser integradas, se entiende que es para realizar el tipo de ataque que se ha elegido, pero existe la posibilidad de poder separarla de la placa de captura cortando las conexiones y luego soldar otro tipo de targets si queremos ampliar nuestra búsqueda.

# 

# ESTUDIO TEÓRICO

Existen varios tipos de ataques laterales, cada uno de ello va dirigido a aprovechar diferentes aspectos y características de los sistemas. Los tipos de ataques más comunes son los dos en que se centra este TFG, que se estudiarán más en detalle, para el resto sólo esbozaremos una introducción.

## Análisis de potencia (*Power Analysis*)

Este tipo de ataque se basa en medir y analizar el consumo de energía de un dispositivo durante su operación. Puede revelar información sobre las operaciones internas, como cálculos criptográficos, a través de patrones en el consumo de energía. Básicamente, son ataques que utilizan la variación en el consumo de energía del hardware durante los cálculos. Existen dos tipos de análisis o de ataques cuando hablamos de la interpenetración del consumo de energía del hardware:

* El Análisis de Potencia Simple (SPA): consiste en la interpretación visual de trazas de potencia, o gráficos de la actividad eléctrica a lo largo del tiempo.
* El Análisis de Potencia Diferencial (DPA): es una forma más avanzada de análisis de potencia, que permite a un atacante calcular los valores intermedios dentro de las operaciones criptográficas a través del análisis estadístico de datos recopilados de múltiples operaciones criptográficas.

Diagrama

Descripción generada automáticamente

Ilustración 15: Diagrama de análisis de Potencia Diferencial

El ataque DPA consta de tres fases principales: en primer lugar, la fase de recopilación de datos, durante la cual se recopila un conjunto de trazas. Luego, tenemos una fase de entrenamiento, en la cual el atacante calibra la plantilla (ataque clásico) o entrena el modelo de aprendizaje automático.

Finalmente, está la fase de ataque, donde la plantilla/modelo se utiliza para recuperar un conjunto de claves secretas y evaluar la resistencia de la implementación a un *Side Channel Attack*. Durante esta fase, los atacantes utilizan múltiples trazas para cada clave atacada y combinan sus probabilidades para hacer la mejor suposición posible.

Un aspecto técnico importante y, a menudo pasado por alto, de este tipo de ataque de canal lateral es que no recuperará la clave de una sola vez; en cambio, el ataque se centra en recuperar un solo byte de clave. Por esta razón, para AES128, se necesitan 16 ataques (y 16 modelos) para recuperar la clave completa.

Captura de pantalla de un celular con letras

Descripción generada automáticamente

Ilustración 16: Fases para un Ataque de Potencia

Para explicar con detalle cómo funciona el *Power Analysis Attack* explicaremos las 4 fases con más detalle.

### Fase 1. Recopilar trazas de potencia

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ilustración 17: recopilación de Datos de Potencia

1. El programa responsable de recopilar todas las gráficas o trazas de potencia, en este caso viene dado por el Chipwhisperer, nos ofrece directamente la posibilidad de tener un osciloscopio integrado dentro de las herramientas que ofrece.
2. El código provoca la encriptación de una clave y un texto plano elegidos al azar en el hardware objetivo.
3. Al finalizar la encriptación, el programa detiene la captura y recopila la traza de potencia del osciloscopio. La traza y su etiqueta (la clave y el texto plano utilizados) se añaden al conjunto de datos del ataque.

### Fase 2. Configuración del Hardware y Evaluación de resultados

En la práctica, para facilitar las cosas, el código encargado de recopilar las trazas no controla directamente el osciloscopio ni interactúa con el chip objetivo. En su lugar, utiliza hardware (ChipWhisperer) que estandariza esas interacciones y las hace fáciles de programar. Para tener un poco de idea genérica de cómo se vería una traza de potencia podemos observar la Ilustración 18.

Imagen que contiene Gráfico

Descripción generada automáticamente

Ilustración 18: Traza de potencia

Al final de cada captura se obtiene un conjunto de trazas de potencia que se asemeja al que se muestra en la captura de pantalla anterior. Esta imagen muestra la traza de potencia para una implementación de AES no protegida. Es evidente que esta implementación tiene muy pocas contramedidas contra ataques de canal lateral (SCA), ya que se pueden ver claramente los 10 picos en el consumo que se corresponden con los 10 picos de AES en forma de ondas. Las implementaciones mejor protegidas aún muestran este patrón de 10 picos, pero no necesariamente de manera tan clara.

Una vez recopilada toda esta información, pasamos a la fase de entrenamiento y evaluación de información capturada.

El objetivo de recopilar trazas es construir dos conjuntos de datos de ejemplos: uno de entrenamiento y otro de evaluación. El conjunto de entrenamiento es utilizado por algoritmos de ataque, ya sean basados en aprendizaje profundo o no, para aprender cómo cambia el consumo de energía en relación con la clave utilizada durante la encriptación. El conjunto de evaluación, como se discutirá más adelante, se utiliza una vez que el algoritmo está entrenado/calibrado para evaluar la eficiencia del ataque.

Para el conjunto de entrenamiento se puede elegir libremente qué claves y textos planos se utilizarán durante la computación, ya que se asume que este paso se ejecuta en dispositivos que están bajo el control del atacante. Aquí es útil tener un algoritmo de muestreo de la gráfica eficiente para asegurar que el espacio se muestree correctamente. La cantidad de trazas a recopilar varía según la protección de la implementación

El conjunto de evaluación es diferente. Su objetivo es representar un objetivo real. Como resultado, las claves utilizadas en este conjunto deben generarse completamente al azar, ya que el atacante no tiene control sobre cómo los usuarios eligen sus claves. Además, este conjunto debe generarse en un chip diferente para asegurar que el ataque se generalice entre chips, ya que puede haber ligeras variaciones de comportamiento debido a la tolerancia de fabricación del chip.

Existen 2 tipo diferentes de captura, síncrona y asíncrona.

Diagrama

Descripción generada automáticamente

Ilustración 19: Captura síncrona y asíncrona

Realizar una captura sincrónica significa que el osciloscopio utiliza la salida del reloj de la CPU de destino para alinear su muestreo con la ejecución de la CPU de destino, como se puede ver arriba. Esto hace posible capturar el consumo de energía en la frecuencia de reloj objetivo sin perder ninguna información. Este tipo de captura está reservado para la evaluación durante el desarrollo del chip, porque en entornos del mundo real, el atacante no tiene acceso al reloj interno de la CPU, ya que ningún hardware seguro y sensato lo expondrá jamás.

En cambio, realizar una captura asíncrona implica que el reloj de la CPU no se está utilizando para sincronizar el reloj del osciloscopio. Como se ilustra arriba, ser asíncrono le obliga a sobre muestrear para evitar perder cambios en el consumo de energía. Más precisamente, el teorema de muestreo de Shanon-Nyquist establece que es necesario muestrear al menos dos veces más rápido para calificar adecuadamente la señal. En la práctica, es necesario capturar al menos cuatro veces la velocidad del reloj objetivo para estar seguro y asegurarse de tener rastros numéricos explotables. La captura asincrónica es el único modo realista al evaluar el rendimiento del ataque, porque un atacante nunca tiene acceso al reloj del elemento seguro. Si bien el uso de la recopilación asíncrona puede parecer obvio, lamentablemente muchos esfuerzos de investigación utilizan la captura sincrónica, porque el sobre muestreo crea rastros que son cuatro veces más grandes, lo que los hace más difíciles de almacenar y atacar. Por ejemplo, algunas de las implementaciones de AES más protegidas abarcan más de 100 000 puntos por seguimiento, lo que en modo asíncrono le llevó a capturar 400 000 puntos. Procesar ejemplos de ese tamaño utilizando el aprendizaje profundo es todo un desafío, ya que consume mucho la memoria de la GPU.

### Fase 3. Entrenamiento

La fase de entrenamiento se realiza en una computadora bajo el control del atacante y tiene como objetivo construir un modelo de fuga utilizando los rastros recopilados anteriormente.

Las técnicas SCA tradicionales, como los ataques de plantilla y los ataques SCAAML (asistidos por canal lateral con aprendizaje profundo), utilizan una “fase de entrenamiento” para aprender esos patrones, pero la forma en que lo hacen es muy diferente: los ataques de plantilla, que son Los ataques de última generación, previos al aprendizaje profundo, utilizan datos de entrenamiento para realizar un análisis estadístico multivariado , creando un modelo de fuga aproximado conocido como modelo *Hamming Weight Power*. Los ataques SCAAML utilizan datos de entrenamiento sin procesar para entrenar modelos de aprendizaje profundo que pueden predecir un valor objetivo directamente a partir de ellos. En cierto modo, esto es muy similar a lo que sucedió en el campo de la visión por computadora: los ataques de plantilla, al igual que los antiguos algoritmos de visión, se basan en características creadas por humanos, mientras que los modelos de aprendizaje profundo trabajan directamente con los datos sin procesar.

La siguiente pregunta que necesitamos responder es: ¿qué valores debería predecir el ataque? La respuesta obvia sería el valor de los bytes de la clave. Sin embargo, en la práctica, para la mayoría de las implementaciones, esto no funcionará a menos que, por error, se haya incluido la carga de la clave en la memoria en la captura. Se llama a esto un error porque, en la mayoría de los escenarios del mundo real la clave está integrada en el dispositivo y solo se carga una vez, no en cada encriptación. Por lo tanto, esperar observarla con cada encriptación es un modelo de ataque poco razonable y conduce a resultados falsos positivos.

En cambio, el ataque intenta predecir los valores de lo que se llaman puntos de ataque, también conocidos como variables (de fuga) sensibles en la comunidad de criptoanálisis. En pocas palabras, un punto de ataque es un punto en el algoritmo donde la computación causó algún cambio en la memoria (cambiar un valor de registro, establecer un valor, etc.) que tiene alguna relación con el valor que intentamos recuperar (por ejemplo, XORing It). Cambiar los valores de la memoria desencadena un cambio en el consumo de energía, lo que significa que este cambio será observable en la traza de potencia.

Diagrama

Descripción generada automáticamente

Ilustración 20: Ejemplo Gráfico del Comportamiento de AES Attack Points [10]

Como se puede ver en el diagrama anterior, que representa las operaciones de AES a lo largo del tiempo, AES tiene muchos puntos de ataque a lo largo de sus 10 rondas (los puntos de ataque están representados por los puntos rojos/amarillos). Sin embargo, en la práctica, la mayoría de ellos no son invertibles; no podemos inferir el valor del byte clave a partir del valor supuesto. Como resultado, nos enfocamos solo en aquellos marcados con puntos rojos, que son directamente invertibles. Los tres principales, todos ubicados en la primera ronda y representados en el lado derecho del diagrama, son:

* Clave (key): Intentar inferir la clave ya que el valor del byte se almacena en la memoria.
* Sub\_bytes\_in: El valor del byte objetivo después de que la clave se almacena con el texto plano.
* Sub\_bytes\_out: El valor del byte después de ser sustituido por otro valor utilizando la caja AES.

La facilidad con la que se ataca un punto sobre otro depende de la implementación. Para TinyAES, tanto sub\_bytes\_in como sub\_bytes\_out son fáciles de atacar y key, como se esperaba, no funciona.

Las arquitecturas basadas en RNN (red neuronal recurrente) y CNN residual (red neuronal convolucional) pueden dar buenos resultados. Sin embargo, a menudo es difícil encontrar los parámetros de arquitectura correctos, razón por la cual la mayoría de los estudios de este tipo de ataques se suele utilizar en gran medida el hiper ajuste para encontrar lo que funciona para un objetivo determinado y se suele emplear la forma de preactivación de ResNet (también conocida como ResNet v2), una arquitectura popular para visión artificial para crear este tipo de arquitecturas.

Una vez que encontremos la arquitectura de modelo correcta y los hiper parámetros correctos, el procedimiento de entrenamiento del modelo es muy estándar. Las únicas dos cosas que vale la pena destacar son:

* Las entradas deben escalarse o los modelos no convergerán en absoluto. Escalamos las entradas entre -1/1 pero 0/1 también debería funcionar.
* La capa de salida debe usar una activación softmax y se debe usar entropía cruzada categórica como función de pérdida. Además de ser la mejor manera de hacer converger la red, el uso de una activación softmax es obligatorio para llevar a cabo el ataque porque garantiza que la red genere una distribución de probabilidad sobre todos los valores de bytes que son resultados posibles, como se analiza en la siguiente sección.

### Fase 4. Ataque

Como último paso, el modelo entrenado se utiliza para recuperar claves que no se vieron durante el entrenamiento para evaluar la efectividad del ataque. El conjunto de datos de ataque es un poco diferente del de entrenamiento, ya que las claves utilizadas en él deben extraerse de forma puramente aleatoria para imitar el hecho de que, en entornos del mundo real, las claves de los dispositivos se generan aleatoriamente cuando se inicializan.

En este punto, para recuperar las claves después del análisis, recopilación de datos y evaluación de dichos resultados, recurriremos a ChipWhisperer y a nuestro modelo ya previamente diseñado para la extracción de la clave que necesitamos.

Diagrama

Descripción generada automáticamente

Ilustración 21 modelo de suma para predecir que byte es correcto

Como se mencionó anteriormente, una de las principales ventajas de utilizar el aprendizaje profundo para SCA es que facilita la realización de un ataque probabilístico que escala según el número de trazas utilizadas. Simplemente necesitas acumular las predicciones de los modelos, como se muestra en el diagrama anterior, para decidir cuál es el valor de byte más probable. Esta suma directa significa que, cuantas más trazas combines durante el ataque, más evidencia se acumulará, y mejor será tu tasa de éxito, siempre y cuando tu modelo haya convergido (es decir, predijo de manera más precisa que al azar).

Esta formulación del ataque también tiene la ventaja de producir naturalmente una lista completamente ordenada de suposiciones potenciales, lo que facilita la adición de un componente de fuerza bruta para probar las combinaciones más probables en los 16 bytes clave. Por último, pero no menos importante, nos permite saber cuán lejos está la suposición correcta de la parte superior, lo que nos permite medir mejoras en el ataque y comparar modelos de ataque de manera más detallada.

## Contramedidas Contra los *Power Analysis Attacks*

Para protegerse eficazmente contra ataques, el enfoque más confiable es asegurar un tiempo constante para todas las operaciones que dependen de datos secretos.

**Descripción de la Contramedida**

Mantenemos dos Arrays: uno contiene la contraseña correcta y el otro contiene la contraseña ingresada por el usuario. También conservamos el bucle que compara ambos arreglos, pero lo modificamos ligeramente para evitar abortos tempranos en caso de que una letra sea incorrecta.

* **Tamaño fijo de los Arrays**: Modificamos ambos arreglos para que tengan un tamaño fijo de 32 bytes y los rellenamos con el carácter '\0'. Esto asegura que siempre se realicen 32 iteraciones, independientemente de la longitud real de la contraseña, lo que permite un tiempo de ejecución constante, aunque sacrificando algo de rendimiento.
* **Evitar abortos tempranos**: Muchos piensan que el canal lateral se debe únicamente a abortos tempranos, pero esto es incorrecto. Queremos un tiempo constante para todas las operaciones dependientes de datos secretos. El uso del enfoque estándar (sin abortos tempranos) expone el programa a ataques de canal lateral debido a la comparación if (correct\_passwd[i] != passwd[i]), que depende del secreto. Esto crea una desproporción en la ejecución, ya que el programa ejecuta una instrucción en caso de igualdad y no lo hace en caso contrario. Añadir un else con la misma operación no resuelve el problema, ya que afecta al rendimiento y no protege contra la predicción de bifurcación.

### Solución propuesta 1. Tiempo constante

La solución consiste en:

* Operación XOR: Para cada byte de las dos cadenas, calcular (correct\_passwd[i] XOR passwd[i]).
* Acumulación del resultado: Realizar una OR bit a bit del valor actual de una variable acumuladora con el resultado del XOR y almacenar el resultado en esta variable (inicializada a 0).

Al final del bucle, la variable acumuladora será igual a 0 únicamente si las dos cadenas son iguales. Esto asegura un tiempo de ejecución constante y protege contra ataques de canal lateral.

**Implementación Detallada**

* Inicializar dos arreglos de tamaño fijo (32) y rellenarlos con '\0'.
* Comparar cada byte de los arreglos utilizando XOR.
* Acumular el resultado de las operaciones XOR en una variable inicializada a 0.
* Al final del bucle, verificar si la variable acumuladora es 0 para determinar si las contraseñas son iguales.

Este enfoque garantiza que el tiempo de ejecución no dependa del contenido de las contraseñas, mejorando la seguridad contra ataques de canal lateral.

La modificación realizada se ve reflejada en el código C original, el cual hemos adaptado para realizar las contramedidas correspondientes junto con el código Python que ejecutamos en el Jupiter.

### Solución propuesta 2. Ruido aleatorio

Para protegerse eficazmente contra ataques de canal lateral, una estrategia efectiva es añadir ruido a las operaciones de comparación, dificultando que un atacante pueda inferir información sensible a partir de observaciones externas.

La clave de esta contramedida es introducir cálculos adicionales y operaciones sin sentido dentro del bucle de comparación. Estas operaciones no afectan el resultado final, pero crean un patrón de ejecución más ruidoso y complejo, haciendo más difícil para un atacante extraer información útil a partir del análisis de tiempo u otros métodos de observación.

Este enfoque garantiza que las diferencias en el tiempo de ejecución y otros patrones sean menos predecibles, dificultando los ataques de canal lateral sin necesidad de hacer las comparaciones en tiempo constante explícitamente. La inyección de ruido hace que las observaciones del atacante sean menos fiables para inferir el contenido de la contraseña.

Esta es la segunda contramedida que hemos decidido aplicar en este ataque y evitar la obtención del resultado esperado por el atacante.

### Solución propuesta 3. Evitar abortos tempranos

La última contramedida que hemos decidido aplicar también en este caso es, en el propio código existente, poner a 0 el return haciendo que a la hora de ejecutar el código no devuelva ningún resultado o un carácter totalmente aleatorio.

## Otros

### Análisis de Emisión Electromagnética (EMA):

Los ataques de canal lateral por emisión electromagnética son un tipo de ataque en el que un atacante observa las emisiones electromagnéticas generadas por un dispositivo electrónico durante su operación. Estas emisiones pueden contener información que permita inferir datos sensibles procesados por el dispositivo, como claves criptográficas o contraseñas.

Como funciona:

* Emisiones Electromagnéticas: Todos los dispositivos electrónicos, incluyendo computadoras, teléfonos y dispositivos criptográficos, emiten señales electromagnéticas durante su funcionamiento. Estas señales pueden ser inadvertidamente moduladas por las operaciones internas del dispositivo, reflejando información sobre los datos que están siendo procesados.
* Captura de Señales: Un atacante puede utilizar equipos de captura, como antenas y receptores sensibles, para detectar y registrar estas emisiones electromagnéticas. Dependiendo del tipo de dispositivo y la operación que se esté ejecutando, las emisiones pueden variar en frecuencia, amplitud y otras características.
* Análisis de Señales: Una vez capturadas, las señales pueden ser analizadas utilizando técnicas de procesamiento de señales y análisis estadístico para extraer patrones correlacionados con las operaciones internas del dispositivo. Por ejemplo, se pueden identificar cambios en las emisiones que correspondan a operaciones específicas como la carga de datos en memoria o la ejecución de instrucciones de comparación.

Ejemplos de Ataques:

* Ataques a Claves Criptográficas: Un ejemplo común es el ataque a dispositivos criptográficos que implementan algoritmos como RSA o AES. Los cambios en las emisiones electromagneticas durante la ejecución de estos algoritmos pueden revelar información sobre las claves criptográficas, permitiendo que un atacante las recupere.
* Ataques a Teclados: Otro ejemplo es la captura de emisiones electromagnéticas de un teclado. Cada tecla presionada puede generar un patrón único de emisiones, permitiendo que un atacante reconozca las teclas pulsadas y, por ende, reconstruya información sensible como contraseñas o mensajes.

Contramedidas:

* Blindaje Electromagnético: Envolver los dispositivos electrónicos con materiales que bloqueen o reduzcan las emisiones electromagnéticas puede dificultar la captura de señales. Esto se puede lograr utilizando recintos metálicos o materiales conductores.
* Filtrado de Señales: Implementar filtrado y acondicionamiento de señales dentro del dispositivo para minimizar las emisiones electromagnéticas en las frecuencias sensibles.
* Ruido Intencional: Añadir ruido electromagnético a las emisiones del dispositivo para dificultar que un atacante pueda distinguir las señales útiles. Este ruido puede ser generado por circuitos diseñados específicamente para tal fin.
* Diseño de Hardware Seguro: Diseñar circuitos y componentes electrónicos que minimicen las variaciones en las emisiones electromagnéticas durante la operación, haciendo más difícil para un atacante correlacionar las señales con las operaciones internas.
* Operaciones Criptográficas en Tiempo Constante: Asegurar que las operaciones criptográficas tomen el mismo tiempo de ejecución independientemente de los datos procesados, reduciendo las variaciones en las emisiones.

### Análisis de sonido (*Acoustic Analysis*):

Los ataques mediante análisis acústico son un tipo de ataque de canal lateral en el que un atacante escucha y analiza los sonidos emitidos por un dispositivo electrónico para inferir información sensible, como claves criptográficas o contraseñas.

Funcionamiento:

* Emisiones Acústicas: Dispositivos electrónicos emiten sonidos durante su operación, generados por componentes como procesadores y teclados.
* Captura de Sonidos: Los atacantes utilizan micrófonos sensibles para captar estos sonidos.
* Análisis de Sonidos: Los sonidos capturados se analizan para identificar patrones que revelen operaciones internas del dispositivo.

Ejemplos de Ataques:

* Aislamiento Acústico: Usar materiales que absorban el sonido para reducir emisiones acústicas.
* Ruido Blanco: Generar ruido blanco para enmascarar los sonidos específicos del dispositivo.
* Diseño de Hardware Silencioso: Minimizar emisiones acústicas mediante el diseño de hardware.
* Operaciones Criptográficas en Tiempo Constante: Asegurar que las operaciones criptográficas se realicen en tiempo constante para evitar variaciones acústicas.

Entre estos existen varios más los cuales no citaremos pero que se pueden investigar externamente.

En conclusión, los ataques de canal lateral representan una amenaza significativa para la seguridad de los sistemas electrónicos, ya que explotan las emisiones no intencionales como señales electromagnéticas, acústicas y de energía, para inferir información sensible sin necesidad de comprometer directamente el sistema. Estos ataques pueden revelar datos críticos, como claves criptográficas, contraseñas y otra información confidencial, a través del análisis de patrones en las emisiones del dispositivo durante su operación.

La combinación de estas contramedidas puede aumentar significativamente la resistencia de los sistemas frente a ataques de canal lateral. Es fundamental que los diseñadores y desarrolladores de sistemas electrónicos consideren estas amenazas y apliquen las medidas de seguridad adecuadas durante el diseño y la implementación para proteger la integridad y confidencialidad de los datos sensibles.

# Trabajo práctico

## Ataques simples de Power analysis

Este ataque nos va a permitir traspasar la seguridad de los dispositivos mediante determinar cuándo está ejecutando una operación. En esta prueba experimental usaremos un *Simple Password Check* y demostrar cómo es un ataque mediante análisis de potencia.

En este experimento utilizaremos la plataforma de Jupyter para ejecutar los códigos tanto en C como en Python para poder visualizar y modificar nuestro ataque en tiempo real. Primero explicaremos el código en C que hemos programado con nuestro ataque y luego el código en Python que ejecutamos para visualizarlo.

### Código C víctima comentado

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Logotipo, nombre de la empresa

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Texto

Descripción generada automáticamente

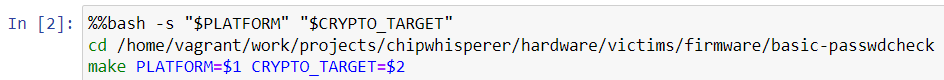
### Código Python que realiza el ataque explicado

#### Descarga del firmware víctima

Forma, Rectángulo

Descripción generada automáticamente con confianza media

Hay que especificar qué dispositivo estamos utilizando y nuestro objetivo. En este caso ARM.



Necesitaremos configurar nuestra PLATAFORMA y luego construir (compilar) el firmware víctima. En el código anterior se indica la ruta del código C víctima y se invoca al programa make para construirlo.

Texto

Descripción generada automáticamente

Realizar una conexión con nuestro dispositivo ChipWhisperer. Además de realizar el ataque, también será el encargado de descargar el código de la víctima.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Especificar la ubicación del código (fichero binario resultado de la compilación del código C) que le vamos a pasar a nuestra placa para que programe y ejecute. Una vez identificada la ruta de este archivo enviamos el código a nuestra víctima a través de ChipWhisperer mediante la conexión realizada anteriormente.

#### Realización del ataque I. Captura de traza

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Como se mencionó al comienzo, el firmware que cargamos en el objetivo implementa una verificación básica de contraseña. Después de recibir una contraseña terminada en '\n', el objetivo la verifica y entra en un bucle infinito, por lo que antes de comunicarnos con él, necesitaremos reiniciarlo. Haremos esto con frecuencia, por lo que definiremos una función, reset\_target(), que reinicie el objetivo.

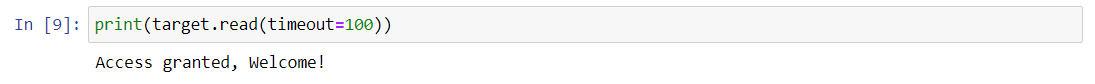
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

El objetivo nos envía un texto al iniciar. Después de ejecutar el siguiente bloque, mostramos este texto por pantalla.



Ahora podemos enviarle una contraseña al objetivo. Por ejemplo, en este caso le enviamos la contraseña correcta para ver el funcionamiento.



Obtenemos la respuesta. Le enviamos la contraseña correcta y nos ha devuelto el mensaje de éxito.



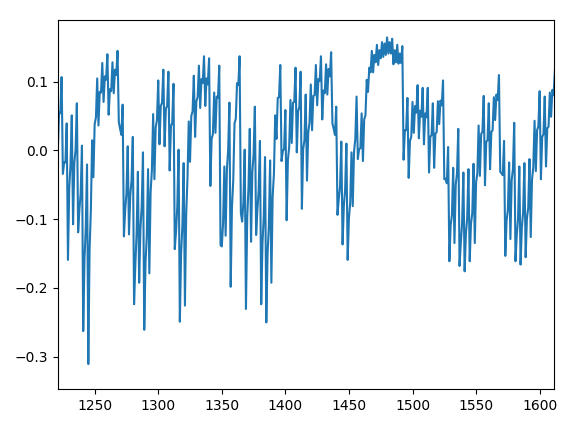
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ahora que podemos comunicarnos con nuestro sistema, nuestro próximo objetivo es obtener una traza de potencia mientras el objetivo está en funcionamiento y le enviamos la contraseña correcta. Para hacer esto, armaremos el osciloscopio con scope.capture() justo antes de enviar nuestro intento de contraseña, y luego registraremos la traza en la variable trace.

Gráfico, Gráfico de barras

Descripción generada automáticamente



Ahora que tenemos la traza, la vamos a mostrar gráficamente.

#### Realización del ataque II. Obtención de la contraseña

Interfaz de usuario gráfica, Texto, Aplicación

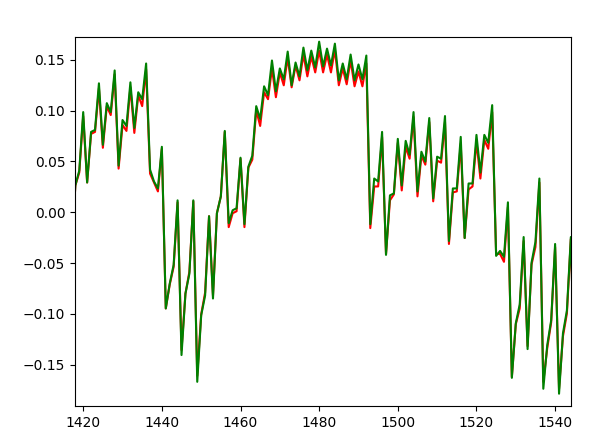
Descripción generada automáticamente

Ahora que podemos capturar trazas, podemos comenzar a programar nuestro ataque. Primero haremos una función para adivinar una contraseña y devolver una traza de potencia, ya que repetiremos esos pasos muchas veces.

Gráfico

Descripción generada automáticamente

A continuación, probamos dos contraseñas diferentes y veremos si las trazas de potencia difieren en longitud. Luego, trazaremos ambas trazas en la misma figura con diferente color.



Ampliando la imagen se puede observar que difieren ligeramente en longitud.

Gráfico

Descripción generada automáticamente

A continuación, generamos un *array* de caracteres con los que probar contraseñas aleatorias. En la imagen superior, vamos probando una a una todas las letras minúsculas y números, y llamamos a la función anterior para que se envíe como contraseña probada al sistema víctima. Si variamos la contraseña se podría observar cómo cambia la traza según el número y longitud de caracteres.

Gráfico

Descripción generada automáticamente

Ampliando la imagen al ser varias trazas apenas se puede distinguir, pero ampliándolo lo suficiente se puede observar como cada traza tiene una ligera variación respecto a las demás como se puede observar en la imagen a continuación.

Gráfico

Descripción generada automáticamente

Imagen ampliada de la traza anterior donde se puede observar como difieren en longitud todas las trazas.



Texto

Descripción generada automáticamente

Nos enfocaremos en usar una coincidencia de Suma de Diferencias Absolutas (SAD, por sus siglas en inglés), que es un método para medir la diferencia entre dos señales para encontrar si una traza objetivo está o no desplazada en el tiempo respecto a una traza de referencia.

El cálculo de SAD es muy simple:

Texto

Descripción generada automáticamente

Donde Tref (j) es un punto único de la traza de referencia, ttarget (j) es un punto único de la traza objetivo, y (J) es el punto a lo largo de la traza en el que estamos tomando la diferencia. Esto se realiza a lo largo de la longitud de la traza de referencia, (J). En resumen, estamos restando las dos trazas, tomando el valor absoluto y luego sumando estas diferencias absolutas. Si este valor es bajo, las trazas son muy similares. Si el valor es alto, son muy diferentes.

Nuestra estrategia se basa en lo siguiente:

* Capturar una traza de referencia y encontrar un segmento único.
* Adivinar otro carácter y deslizar la referencia a lo largo de la traza, calculando SAD en cada desfase hasta encontrar uno por debajo de un cierto umbral.
* Repetir esto hasta encontrar un carácter con un desfase diferente al de la referencia; este es el carácter correcto.
* Repetir con el resto de los caracteres hasta haber descifrado la contraseña.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Con la traza inicial (contraseña correcta) y la obtenida ahora (contraseña aleatoria), se alinean y muestran gráficamente.

Ahora que tenemos una función que funciona para encontrar un desfase usando una coincidencia de SAD, capturamos una traza de referencia, así como una traza con una conjetura correcta. Esta sección de código captura una traza de referencia con la conjetura "a\n" y una traza correcta con la conjetura "h\n". Luego, ambas trazas se trazan en un gráfico: la traza de referencia en rojo y la traza correcta (desfasada) en verde.

Interfaz de usuario gráfica

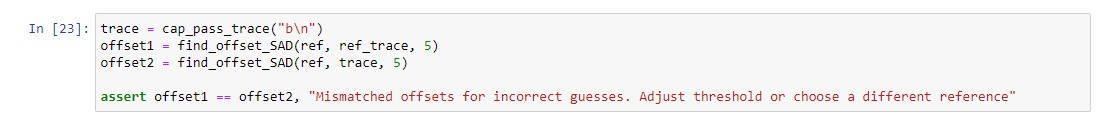
Descripción generada automáticamente

Arriba se muestra un detalle de la traza capturada. A la hora de tomar como referencia las trazas se debe tener en cuenta:

**Debe ser bastante única.** Por ejemplo, la sección que seleccionemos no debe coincidir con ninguna sección posterior en la misma traza.

**Debe ser una porción que se desplace en el tiempo.** Por ejemplo, el principio de la traza de energía probablemente no se desplace en el tiempo aquí, pero las secciones posteriores definitivamente deberían hacerlo.

Finalmente seleccionamos nuestra referencia.



Se busca alinear ambas trazas. Para ello se busca un offset que aplicar a una de ellas mediante la función find\_offset\_SAD(). Una vez alineadas, se puede comparar el momento en el que se empieza a evaluar la contraseña y cuánto dura dicha evaluación. Esto también nos da una buena oportunidad para encontrar un buen umbral. Si el umbral es demasiado alto, la traza coincidirá antes. Si es demasiado bajo, no coincidirá en absoluto.

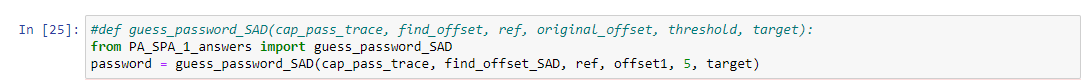
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Si el carácter enviado como contraseña aleatoria coincide con el carácter esperado por la víctima para la contraseña, la evaluación de la contraseña habrá durado más tiempo. Entrará en el último if offset > offset1 y se muestra el carácter obtenido.

Si este proceso se repite varias veces, se irán obteniendo todos y cada uno de los caracteres que formaban la contraseña original.

En resumen, si el desplazamiento es mayor que el que obtenemos de la traza de referencia, sabemos que tenemos un carácter correcto.



Extender esto al resto de la contraseña tampoco es demasiado difícil. Simplemente iteramos a través de todos los caracteres.

Este código define una función guess\_password que usa el método de coincidencia SAD para adivinar los caracteres de la contraseña uno por uno. Si encuentra un carácter correcto, lo añade a la contraseña parcial y continúa hasta que la contraseña completa se ha adivinado. Si en algún punto el umbral es incorrecto, la función imprime un mensaje y devuelve la contraseña parcial o completa encontrada hasta ese momento.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Finalmente, hemos podido descifrar la contraseña como se muestra en la imagen de arriba.

## Implementación y prueba de contramedidas para ataques de power analysis

### Solución propuesta 1. Tiempo constante

Para nuestra contramedida, mantendremos dos Arrays: uno contendrá la contraseña correcta y otro que contiene la contraseña enviada por el usuario. También mantendremos el bucle que itera a través de ambas cadenas, comparándolas, pero ligeramente modificado para evitar abortos tempranos en caso de una letra incorrecta.

Para hacerlo, modificamos ambos Arrays para tener un tamaño fijo (32) y los llenamos con "\0". Haremos 32 iteraciones, a pesar de que el tamaño real de la contraseña sea mucho más corto. Esto nos permite tener un tiempo de ejecución constante, aunque sacrificaremos el rendimiento.

Esta solución consiste en conseguir un tiempo constante para todas las operaciones dependientes. Entonces, usando el mismo enfoque que el desarrollador (sin el aborto temprano), exponemos nuestro programa a un ataque de canal lateral debido a la comparación if(correct\_passwd[i]!=passwd[i]), que es dependiente de la clave, resultando en una desproporción en la operación, ya que el programa ejecutará una instrucción en caso de igualdad y no en caso contrario. Agregar un else con la misma operación no es una solución, ya que afecta el rendimiento y no protege nuestro código de una posible predicción de rama.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

En la imagen superior se puede ver la modificación del código original para poder aplicar nuestra contramedida [11].

Nuestra solución consiste en:

Para cada byte de las dos cadenas:

* Calcular (left\_i XOR right\_i)
* OR bit a bit el valor actual de passwd con el resultado del XOR y almacenar la salida en passwd (que se inicializó en 0).

Al final del bucle, passwd será igual a 0 si y solo si las dos cadenas son iguales.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

En la imagen superior se puede ver el primer intento después de programar nuestro ChipWhisperer con la modificación anterior. Se puede observar que efectivamente a la hora de llevar a cabo el ataque nos da un valor totalmente diferente.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Segundo intento tratando de descifrar la contraseña.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Tercer Intento. Efectivamente no se consigue ningún valor similar a la contraseña original.

### Solución propuesta 2. Ruido aleatorio

La función inject\_noise() está diseñada para introducir un retraso aleatorio en la ejecución del programa, con el propósito de dificultar los ataques de canal lateral que se basan en el análisis del tiempo de ejecución.

Texto

Descripción generada automáticamente

Aquí se genera un número aleatorio entre 0 y 99. La función rand() devuelve un número entero aleatorio, y el operador % 100 toma el resto de la división del número aleatorio por 100, limitando así el rango a 0-99.

Este bucle for introduce un retardo basado en el número aleatorio generado. La palabra clave volatile se usa para evitar que el compilador optimice el bucle, asegurando que se ejecute exactamente como se escribe. El bucle simplemente itera desde 0 hasta noise\_delay, introduciendo así un retardo variable.

Al introducir retrasos aleatorios, la función inject\_noise()hace que el tiempo de ejecución sea menos predecible y más difícil de analizar para un atacante, aumentando así la seguridad del sistema contra este tipo de ataques.

Texto

Descripción generada automáticamente

La función inject\_noise()se llama al principio de cada iteración del bucle para introducir un retraso aleatorio. Esto tiene el propósito de enmascarar el tiempo de ejecución del bucle, dificultando que un atacante pueda analizar patrones de tiempo para deducir la contraseña.

Se compara el carácter de la contraseña ingresada(passwd[i]) con el carácter correspondiente de la contraseña correcta(correct\_passwd[i]). Si los caracteres no coinciden, se procede a ejecutar el bloque de código dentro del if.

Si los caracteres no coinciden, se llama nuevamente a inject\_noise() para introducir otro retraso aleatorio. Esto asegura que el tiempo de ejecución no sea significativamente menor incluso cuando se detecta un carácter incorrecto, dificultando aún más los ataques de análisis de tiempo.

Se establece la variable passbad en 1, indicando que la contraseña ingresada es incorrecta.

En resumen, este código compara cada carácter de la contraseña ingresada con la contraseña correcta, mientras introduce retrasos aleatorios mediante la función inject\_noise() en cada iteración del bucle y también al encontrar una discrepancia. Esto se hace para prevenir ataques de canal lateral basados en el análisis del tiempo de ejecución, haciendo que el tiempo total sea menos predecible.

Texto

Descripción generada automáticamente

Como se puede observar en la imagen de superior, hemos conseguido evitar que se pueda descifrar la contraseña.

### Solución propuesta 3. Evitar abortos tempranos

Los abortos tempranos en los ataques de canal lateral aprovechan la variabilidad en el tiempo de ejecución para extraer información sobre los datos procesados. Para protegerse contra estos ataques, es crucial implementar comparaciones de tiempo constante y otras técnicas que enmascaren o eliminen las diferencias de tiempo de ejecución, haciendo que el sistema sea menos susceptible a la extracción de información a través del análisis de tiempo.

Por lo que, en esta solución, lo que hemos implementado es usar el código original, pero modificando el bucle, haciendo que passbad = 0 por lo que a la hora de ejecutar el ataque no obtenemos nada de vuelta. El código no devuelve ningún valor sobre la contraseña.

Texto

Descripción generada automáticamente con confianza media

Con esto lo que obtenemos es que nuestro bucle no devuelva nada por lo que no obtenemos ningún valor que pueda poner en riesgo nuestro código.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Efectivamente, se puede observar en la imagen superior que a la hora de ejecutar el código no obtenemos ningún valor de vuelta.

# Conclusiones

Los side channel attacks son técnicas que explotan información indirecta emitida por sistemas computacionales, como el tiempo de ejecución, el consumo de energía, las emisiones electromagnéticas y acústicas, para inferir datos sensibles. Estos ataques no dependen de vulnerabilidades en los algoritmos criptográficos en sí, sino en su implementación física. Dos tipos prominentes son el análisis de potencia simple (SPA) y el análisis de potencia diferencial (DPA), que se centran en el consumo de energía del dispositivo.

Para mitigar estos ataques, se utilizan técnicas como la inyección de ruido, que introduce variabilidad aleatoria en los tiempos de ejecución, y los algoritmos de tiempo constante, que aseguran que las operaciones siempre tomen la misma cantidad de tiempo, independientemente de los datos procesados. Aunque ninguna contramedida es infalible por sí sola, una combinación de varias estrategias puede proporcionar una protección efectiva contra los ataques de canal lateral, garantizando así una mayor seguridad en los sistemas criptográficos.

# Bibliografía

1. Wikipedia contributors. Side-channel attack. Wikipedia, The Free Encyclopedia. January 10, 2024, 11:14 UTC. Available at: <https://en.wikipedia.org/w/index.php?title=Side-channel_attack&oldid=1194717603>. Accessed January 27, 2024.
2. Colaboradores de Wikipedia, "Ataque de canal lateral," Wikipedia, La enciclopedia libre, <https://es.wikipedia.org/w/index.php?title=Ataque_de_canal_lateral&oldid=155734766> (descargado 1 de diciembre de 2023).
3. Jake Hert. Understanding Side Channel Attacks Basics. All about circuits, <https://www.allaboutcircuits.com/technical-articles/understanding-side-channel-attack-basics/>. Accedido el 27 de enero de 2024.
4. Elie Bursztein, Hacker’s guide to deep-learning side channel attacks: the theory. Elie publications, <https://elie.net/blog/security/hacker-guide-to-deep-learning-side-channel-attacks-the-theory/>. Accedido el 27 de enero de 2024.
5. Wikipedia contributors. Power analysis. Wikipedia, The Free Encyclopedia. August 8, 2022, 01:31 UTC. Available at: <https://en.wikipedia.org/w/index.php?title=Power_analysis&oldid=1103018064>. Accessed January 27, 2024.
6. Wikipedia contributors. (2024, January 26). Timing attack. In *Wikipedia, The Free Encyclopedia*. Retrieved 17:43, February 25, 2024, from <https://en.wikipedia.org/w/index.php?title=Timing_attack&oldid=1199238911>
7. <https://media.newae.com/datasheets/NAE-CW1173_datasheet.pdf>
8. <https://github.com/newaetech/chipwhisperer>
9. <https://rtfm.newae.com/Targets/>
10. Advanced Encryption Standard. (2024, 22 de enero). *Wikipedia, La enciclopedia libre*. Fecha de consulta: 13:22, enero 22, 2024 desde <https://es.wikipedia.org/w/index.php?title=Advanced_Encryption_Standard&oldid=157483752>.
11. https://yan1x0s.medium.com/side-channel-attacks-part-1-timing-analysis-password-recovery-607716bfc56a.

# Anexos

## Código Fuente X

¿¿Esto a que se refiere?? Si he insertado ya el código en la memoria, no es necesario ponerlo aquí.

## Código Fuente Y