Design of a Set of Software Tools for Side-Channel Attacks

A. Fuentes, L. Hernández, A. Martín and B. Alarcos

***Abstract*— This contribution presents the design and the first experimental results of a set of software tools to carry out side-channel attacks against cryptographic devices, especially smartcards. To this aim, the main attacks of this class are commented, with special emphasis in power analysis attacks. The final objective is to make this set of tools available to the scientific community, so that it can be improved and enlarged according to particular needs.**

***Keywords*— side channels, cryptography, software tools, security.**

I. Introducción

TRADICIONALMENTE se acepta que la Criptología se divide en dos partes claramente definidas, la Criptografía y el Criptoanálisis [8], [15]. La primera de ellas tiene como objetivo principal el diseño y la elaboración de sistemas que permitan el cifrado de información, de modo que sólo los usuarios autorizados puedan recuperar la información original, mediante el correspondiente proceso de descifrado. Ambos procesos, cifrado y descifrado, utilizan algoritmos cuyas entradas son la información que se desea cifrar o descifrar y determinadas claves. Por el contrario, el Criptoanálisis tiene como fin romper el secreto de tales comunicaciones, ya sea determinando las claves que se utilizaron en los procesos de cifrado y descifrado, o rompiendo el algoritmo en el que se basan tales procesos.

Como ya se ha dicho, una de las características fundamentales de los sistemas criptográficos en general, ya sean criptosistemas simétricos o asimétricos, esquemas de firma digital, protocolos de acuerdo o intercambio de clave, etc., es el estudio de su seguridad. La primera de las medidas que debe garantizarse para considerar que un sistema criptográfico es seguro es que el espacio de todas sus posibles claves ha de ser lo suficientemente elevado como para hacer inviable un ataque por fuerza bruta, es decir, por la prueba exhaustiva de todas las claves. La segunda medida que se debe considerar es la imposibilidad de romper el algoritmo en el que se basa el sistema criptográfico bajo estudio.

A. Fuentes, Departamento de Tratamiento de la Información y Criptografía, Instituto de Tecnologías Físicas y de la Información, Consejo Superior de Investigaciones Científicas, Madrid, España, alberto.fuentes@iec.csic.es

L. Hernández, Departamento de Tratamiento de la Información y Criptografía, Instituto de Tecnologías Físicas y de la Información, Consejo Superior de Investigaciones Científicas, Madrid, España, luis@iec.csic.es

A. Martín, Departamento de Tratamiento de la Información y Criptografía, Instituto de Tecnologías Físicas y de la Información, Consejo Superior de Investigaciones Científicas, Madrid, España, agustin@iec.csic.es

B. Alarcos, Departamento de Automática, Escuela Politécnica Superior, Universidad de Alcalá, Madrid, España, bernardo.alarcos@uah.es

De este modo, se debe asegurar que el algoritmo no es vulnerable a ataques del tipo hombre en el medio (*man in the middle*), a medio camino, diferencial, etc. [6]. En otras ocasiones, la seguridad de los sistemas es hipotética (aunque aceptada universalmente), como en el caso de algoritmos que basan su seguridad en determinados problemas matemáticos, fundamentalmente de la teoría de números, como la factorización de enteros [17], el logaritmo discreto o elíptico [7], [9], la suma de un subconjunto o mochila [5], etc.

La clasificación más general de los ataques (teóricos) consiste en considerarlos como pasivos o activos [15]. Un ataque se dice pasivo si el atacante sólo monitoriza el canal de comunicaciones e intenta vulnerar la confidencialidad de los datos, esto es, obtener el texto claro a partir del texto cifrado; mientras que un ataque se llama activo si el adversario intenta borrar, añadir o, en general, alterar la transmisión de la información; es decir, su objetivo es amenazar la integridad de los datos, su autenticidad y su confidencialidad. Los ataques pasivos se suelen dividir en los siguientes tipos, dependiendo de la clase de información a la que el atacante tenga acceso: ataque al texto cifrado, al texto claro conocido, al texto claro elegido, al texto claro elegido adaptativo, al texto cifrado elegido y al texto cifrado elegido adaptativo. En cuanto a los ataques más extendidos a los protocolos destacan los siguientes: a la clave conocida, por repetición, por suplantación, por diccionario, por búsqueda hacia adelante y por intercalado.

Por otra parte, es sabido que sólo los algoritmos criptográficos cuya seguridad teórica ha sido demostrada o garantizada son los que pasan a la fase de ser implementados en dispositivos criptográficos como etiquetas RFID, tokens USB y, principalmente, tarjetas inteligentes. Las principales razones que motivan las implementaciones de protocolos criptográficos en estos dispositivos son su ubicuidad, facilidad de uso, comodidad y el hecho de que incorporen medidas de seguridad. Todo ello lleva al uso de procesadores criptográficos optimizados para tareas específicas que son embebidos en dichas tarjetas.

Todos estos dispositivos capaces de realizar operaciones criptográficas se han convertido en una herramienta indispensable en muchas actividades de nuestra vida cotidiana, como por ejemplo, la identificación de individuos (autenticación de usuarios en sistemas de información, tarjetas de identificación personal, pasaportes, etc.), la firma electrónica para garantizar el acceso a determinada información o el pago por bienes y servicios (tarjetas de firma, de identificación personal, etc.) y el almacenamiento de información cuya manipulación está restringida a entidades específicas (tarjetas prepago, tarjetas SIM, etc.).

Sin embargo, la seguridad teórica de los sistemas criptográficos no garantiza su seguridad práctica, dado que en su implementación intervienen otros muchos factores que es preciso considerar y que no suelen ser tenidos en cuenta desde el punto de vista del diseñador de un protocolo criptográfico.

La información almacenada en los dispositivos criptográficos (típicamente una tarjeta inteligente), sólo puede ser utilizada a través de los algoritmos definidos por los desarrolladores del dispositivo, por lo que un usuario de dicha tarjeta sólo puede hacer uso de datos y algoritmos tras ciertas comprobaciones y con las restricciones que se hayan implementado. Por todo ello, una constante fundamental en todos los productos criptográficos y, por tanto, en sus implementaciones, es el análisis de su seguridad, tanto desde el punto de vista teórico como práctico.

Un atacante a un protocolo de acuerdo de clave, de firma digital o de cifrado/descifrado, por ejemplo, siempre buscará su parte más débil. Y ésta es, en muchos casos, su implementación en una tarjeta inteligente.

Los tipos de ataque que tienen como objetivo obtener información a partir de la implementación insegura de un protocolo criptográfico en una tarjeta inteligente, se han dado en llamar ataques por canal lateral (*side channel attacks*) o por inducción de fallos (*fault attacks*).

Debe tenerse en cuenta que cuando un criptoprocesador ejecuta determinado código, que implementa un protocolo o algoritmo específico, conlleva, entre otras características, un tiempo de computación, un consumo de potencia eléctrica, la generación de ondas electromagnéticas, etc.

La hipótesis en la que se basan estos ataques es que las magnitudes e intensidades de dichas características dependen directamente de las instrucciones, operaciones matemáticas y datos utilizados por el procesador. De esta forma, la clave criptográfica empleada (ya sea secreta o privada) puede ser inferida mediante el análisis de la información obtenida midiendo las fugas que se producen por estos denominados canales laterales.

En este trabajo se estudian los ataques por canal lateral a sistemas criptográficos implementados en tarjetas y se presenta el diseño de un conjunto de herramientas que permitan llevar a cabo ataques por canal lateral. Se pretende poner dicha herramienta al servicio de la comunidad científica interesada.

El resto del contenido de esta comunicación se estructura de la siguiente manera. En la sección II se hace un repaso general de los principales tipos de ataques a los diferentes dispositivos físicos. El diseño de las herramientas de ataque por canal lateral se lleva a cabo en la sección III; mientras que en la sección IV se presentan algunos resultados obtenidos durante la fase de diseño de tales herramientas. Finalmente, las principales conclusiones de este trabajo se muestran en la sección V.

II. Ataques a Dispositivos Físicos

Como ya se ha mencionado, hasta hace unos años la garantía de los sistemas criptográficos se basaba en el estudio teórico de su seguridad; sin embargo, desde la publicación de los artículos de Kocher [12] y Boneh et al. [3], esta garantía ya no es suficiente si se desea que los sistemas criptográficos implementados en dispositivos físicos sigan considerándose seguros. Dado que estos ataques están relacionados con la implementación física, se suelen denominar genéricamente ataques físicos [8].

Los problemas de seguridad en las implementaciones surgen bien por la existencia de determinados canales por los que es posible obtener información sensible de la zona considerada segura del dispositivo, bien por la posibilidad de inducir fallos en el comportamiento del circuito electrónico y, analizando el comportamiento del sistema, deducir información sobre las claves.

Dado que los ataques físicos son menos generales que los criptoanálisis clásicos, puesto que están específicamente ligados al modo de implementación, arquitectura del chip, etc., se suelen clasificar, bien como invasivos, semiinvasivos o no invasivos [2], [18] según que se manipule el dispositivo o sólo se haga uso de información disponible; bien como activos o pasivos, en analogía con los criptoanálisis clásicos, según que los ataques traten de manipular el funcionamiento del dispositivo o sólo observen el comportamiento del mismo durante el procesado del algoritmo criptográfico.

En los ataques físicos se supone, de forma análoga al criptoanálisis tradicional, que se verifica el principio de Kerckhoffs [11], esto es, el atacante tiene acceso al dispositivo y conoce el algoritmo criptográfico que está implementado en el chip, así como los detalles de la implementación; de modo que lo único que no se conoce es la clave. Por otra parte, se supone que el dispositivo se podrá hacer funcionar tantas veces como se desee, eligiendo los valores de entrada, y se podrá actuar sobre él o medir ciertos parámetros en su entorno.

A continuación se describen los principales tipos de ataques físicos.

A. Ataques por análisis temporal

Los ataques por análisis temporal (*timing attack*) pretenden obtener información sobre un criptosistema midiendo el tiempo que el dispositivo tarda en realizar las operaciones del algoritmo que implementa [12].

A modo que ejemplo, supóngase que se desea conocer la clave privada de un criptosistema de clave asimétrica mientras se ejecuta la exponenciación modular mediante el algoritmo de elevar al cuadrado y multiplicar. Es sabido que el tiempo de ejecución de la multiplicación es constante, pero si el resultado de la multiplicación es mayor que el módulo considerado, se debe hacer una reducción adicional y el tiempo de ejecución aumenta.

Mediante esta hipótesis, es claro que se obtiene información acerca de los tamaños de los números que intervienen en cada paso de la operación, lo que ayuda a la determinación de la clave si se ejecutan un gran número de muestras de texto claro.

Es sabido que la memoria caché almacena copias de los datos usados con mayor frecuencia. Cuando el procesador necesita leer una celda de la memoria principal, primero comprueba si el dato está ya en la caché. Si ése es el caso (*cache hit*), el procesador usa el dato de modo inmediato sin acceder a la memoria principal; en caso contrario (*cache miss*), el dato se

lee de la memoria y se almacena en la caché.

Este comportamiento se aprovecha en los ataques denominados por análisis de la caché, que se suelen emplear en criptosistemas de clave simétrica, dado que su tiempo de ejecución no presenta suficiente correlación con los valores cifrados. En este caso se hace uso del hecho de que el tiempo requerido para acceder a la caché cuando los datos están presentes en ella es mucho menor que cuando no están [10].

Otro tipo de ataque por análisis temporal contra las CPU de los ordenadores se denomina por análisis de la predicción de saltos (*branch prediction analysis*). En este caso se hace uso del procesado determinístico de las unidades de predicción de saltos de las modernas CPU, que predicen la secuencia de instrucciones esperada antes de obtener el resultado real de la directiva de salto. El ataque aprovecha la penalización en tiempo (ciclos extra de reloj) que supone un error en la predicción de salto en el flujo de un programa cuya secuencia depende de los datos utilizados [1]. La relación entre la penalización en tiempo y los datos procesados es utilizada por el atacante para obtener información sobre dichos datos, entre los que se encuentra la clave buscada.

B. Ataques por análisis de potencia

Dado que el consumo de potencia de los microprocesadores actuales está muy relacionado con el número de bits que cambian en memoria o registros, un atacante puede aprovechar este hecho para intentar adivinar un valor secreto utilizado en una operación criptográfica observando la curva de consumo de potencia o traza. A continuación se comentan de forma resumida los principales ataques que emplean este método.

Los ataques por análisis simple de potencia (*Simple Power Analysis*, SPA) emplean trazas de consumo de potencia medidas durante el funcionamiento del dispositivo criptográfico. Estas medidas se determinan con un osciloscopio digital que mide la caída de tensión en una resistencia que se conecta en serie con el dispositivo, a la alimentación. Para que el atacante pueda obtener la clave más o menos directamente a partir de una determinada traza (o de un conjunto de unas pocas trazas), lo más normal es que necesite un conocimiento detallado del algoritmo criptográfico que está siendo ejecutado y, a ser posible, de las características de su implementación en el dispositivo atacado. En algunas ocasiones, un análisis visual de la traza permite obtener información relevante sobre el tipo de criptosistema empleado.

A modo de ejemplo, en la Fig. 1 se muestra una traza de potencia de una ejecución de un DEA (*Data Encryption Algorithm*), donde se aprecia la repetición de un patrón 16 veces.

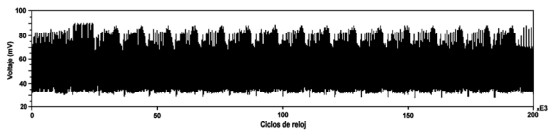

Figura 1. Traza de potencia de un DEA.

Si la relación entre la potencia consumida y la clave criptográfica no es clara, la señal que se obtiene suele ser muy débil y los ataques SPA no dan información suficiente para obtener la clave. En este caso, se suelen emplear técnicas estadísticas y ejecutar un ataque por análisis diferencial de potencia (*Differential Power Analysis*, DPA). Para llevar a cabo este ataque se hacen muchos cifrados con diferentes entradas, se miden y almacenan las correspondientes curvas de consumo de potencia y se sincronizan las medidas (ver Fig. 2 es decir, se procede a un alineamiento de las trazas de modo que se garantice la comparación de valores de potencia consumida en los mismos instantes a lo largo de la ejecución del algoritmo [13].

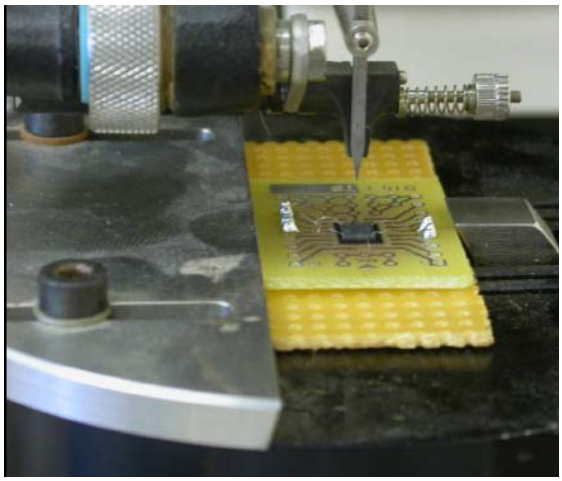

Figura 2. Cableado para medida del consumo de potencia del chip.

Los ataques por correlación de la potencia consumida (*Correlation Power Analysis*, CPA) consideran la correlación entre los datos procesados y la medición de la potencia instantánea consumida por el dispositivo [4]. Dado que esta correlación suele ser muy pequeña, se llevan a cabo muchas medidas para disponer de muchas trazas y utilizar después métodos estadísticos para comparar dichas trazas con las salidas de un determinado modelo del dispositivo.

Finalmente, señalar que los ataques DPA se pueden generalizar a los llamados ataques por análisis diferencial de orden superior (*Higher-Order Differential Power Analysis*, HODPA) en los que no se usa un único punto de la traza de potencia sino varios. Se trata de analizar si el consumo en su conjunto está correlado con el valor del bit según la conjetura que se haya hecho sobre la clave. En general, un ataque DPA de orden n hace uso simultáneamente de n muestras que corresponden a n valores intermedios diferentes (medidos en diferentes instantes) de la misma traza.

C. Ataques por análisis de emanaciones electromagnéticas

Los ataques por análisis de emanaciones electromagnéticas (*ElectroMagnetic Analysis*, EMA) consideran las emanaciones electromagnéticas emitidas por un circuito debidas, según las ecuaciones de Maxwell, al desplazamiento de cargas a lo largo de las pistas de las capas de metal del circuito. Estas emanaciones son especialmente significativas cuando los transistores conmutan de estado [16]. En este tipo de ataque se colocan sondas en las cercanías del chip para medir el campo electromagnético (ver Figura 3).

Figura 3. Medidas con sonda para un ataque EMA.

Aunque en la práctica se puede medir tanto el campo eléctrico como el magnético, para los ataques por canal lateral se obtienen mejores resultados haciendo uso del campo magnético. Por otra parte, es posible aumentar significativamente la determinación de este campo si se lleva a cabo un decapado previo del chip, permitiendo acercar más la sonda de medida a las zonas del chip donde están las fuentes de emanación.

La información recogida se analiza de forma similar a como se hace con las trazas de potencia y entonces se habla de análisis electromagnético simple (*Simple ElectroMagnetic Analysis*, SEMA) o de análisis electromagnético diferencial (*Differential ElectroMagnetic Analysis*, DEMA).

Los ataques de tipo EMA pueden proporcionar más información que los análisis de potencia consumida, pues, por ejemplo, pueden señalar la distribución de los componentes del chip si se conoce la orientación del campo electromagnético medido. En ocasiones puede darse el caso de que es posible medir el campo electromagnético y no el consumo de potencia.

D. Ataques por inducción de fallos

Los ataques por inducción de fallos son ataques activos que intentan manipular un dispositivo criptográfico para alterar su funcionamiento normal, de modo que dicha manipulación proporcione como salida del dispositivo un resultado erróneo en un cálculo, un mensaje de error, etc. Se trata, en definitiva, de obtener algún tipo de información no prevista originalmente en su implementación y explotar dicha información.

Para realizar este tipo de ataque puede ser necesario conocer la distribución de los componentes del chip dentro del mismo, es decir, en qué posición del chip se encuentra el criptoprocesador, determinadas celdas de memoria, etc., de tal manera que sea posible inducir el fallo en una localización específica del chip (ver Fig. 4). Para ello suele ser necesario un considerable trabajo de ingeniería inversa.

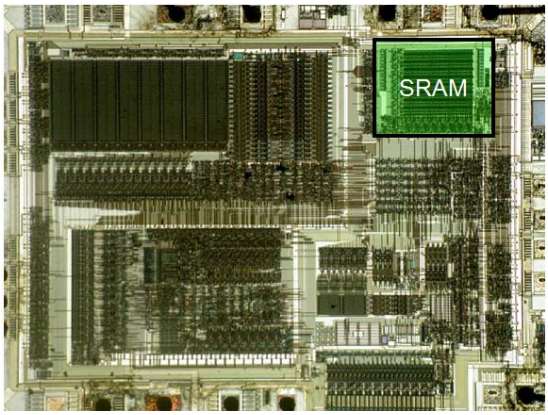

Figura 4. Identificación de regiones sensibles por ingeniería inversa.

Algunas de las técnicas para inducir fallos más comúnmente utilizadas son las siguientes.

Por calor o radiación infrarroja, de modo que si se consigue hacer funcionar un circuito fuera de los márgenes de la temperatura señalada por el fabricante, se puede explotar el hecho de que los umbrales de temperatura para la lectura y escritura no coinciden en la mayoría de las memorias no volátiles.

Los picos de tensión por encima del nivel de tolerancia de los dispositivos pueden provocar un resultado erróneo en un cálculo o fallos en la memoria.

Las variaciones en la frecuencia de reloj, al igual que en el caso anterior, pueden provocar fallos en la memoria o ejecuciones incorrectas de los programas.

Si fuera posible acceder a las capas de silicio de un chip, mediante el uso de rayos láser (rojos o verdes) o luz ultravioleta convenientemente enfocada, se pueden destruir algunas de sus estructuras o alterar valores de la memoria (ver Fig. 5).

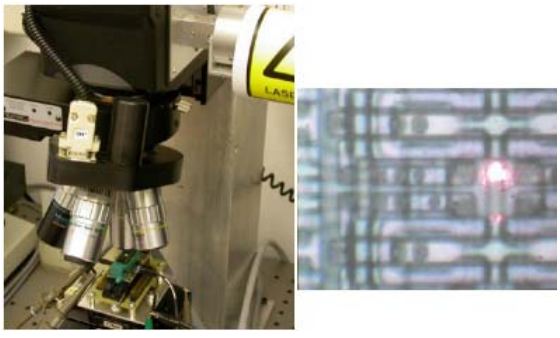

Figura 5. Disparo de luz láser contra un área de memoria.

El paso de corriente eléctrica por una bobina produce un campo magnético, de modo que si se coloca la bobina cerca de la superficie de un chip se inducen en él corrientes de Foucault que pueden afectar al correcto funcionamiento de un transistor o de un bloque de memoria.

En otras ocasiones se emplean haces de iones con el fin de perforar agujeros en las capas del chip e introducir por ellos material conductor de modo que sea posible acceder con equipos de medida a elementos individuales del chip.

Por otra parte, además de las técnicas de fallo, se suelen considerar modelos de fallo, que son representaciones de cómo

el fallo ha influido en el comportamiento del dispositivo, de modo que se hace una hipótesis de cuántos bits han sido perturbados y cuál ha sido el impacto de la perturbación sobre tales bits.

E. Ataques por métodos combinados y contramedidas

Además de llevar a cabo los ataques ya presentados, es posible realizar ataques en los que intervengan varios de ellos de modo que la potencia del ataque se vea incrementada. De este modo, es posible combinar dos o más ataques utilizando, por ejemplo, el consumo de potencia y emanaciones electromagnéticas simultáneamente.

Es fácil suponer que cada vez que se publica un determinado tipo de ataque que pueda afectar a una serie de dispositivos o de algoritmos implementados en ellos, enseguida se plantea la necesidad de estudiar cómo los dispositivos pueden prevenir dichos ataques. Se trata, en definitiva, de implementar determinadas contramedidas que hagan inviable o, al menos, dificulten, tales ataques.

En todo caso, la decisión de implementar o no la contramedida, si es que se encuentra, dependerá de la valoración entre el coste de la contramedida y el valor de lo que se desea proteger. Las principales consideraciones que se suelen tener en cuenta para elaborar contramedidas son las que se mencionan a continuación.

Para evitar ataques por canal lateral se utilizan estrategias, tanto en hardware como software, como decorrelar las trazas de salida de ejecuciones individuales introduciendo retrasos temporales aleatorios y estados de espera, insertando instrucciones inútiles, haciendo aleatoria la ejecución de ciertas operaciones, etc.

Otras estrategias consisten en cambiar instrucciones críticas por otras cuya traza de consumo de potencia sea difícil de analizar; rediseñar la circuitería encargada de las operaciones aritméticas o de las transferencias a memoria; modificar los algoritmos criptográficos para que los ataques resulten ineficientes, como por ejemplo enmascarando de modo aleatorio datos y claves, etc.

III. Diseño de Herramientas para Ataques por Canal Lateral

Una vez que se han presentado los principales ataques por canal lateral y por fallos, consideraremos de modo especial los ataques de tipo DPA para los que estará especialmente diseñado el conjunto de herramientas presentado en este trabajo.

Como ya se ha mencionado anteriormente (ver §II-B), en los ataques DPA, al contrario de lo que sucede con los ataques SPA, un atacante no necesita un conocimiento detallado del dispositivo a atacar. Además, con estos ataques es posible obtener la clave secreta, incluso cuando las trazas de potencia sean ruidosas. Por otro lado, este tipo de ataque requiere una gran cantidad de trazas de potencia.

Un ataque DPA consta de los siguientes cinco pasos [14].

1) *Elegir un resultado intermedio del algoritmo ejecutado*. Este primer paso consiste en elegir un resultado intermedio del algoritmo criptográfico que ha sido ejecutado por el dispositivo a atacar. Dicho valor se obtiene a partir de una función que utiliza como entradas parte de las claves criptográficas y datos conocidos.
Dicho de otro modo, el valor intermedio es el resultado de una función $f(c, k)$, siendo c un valor no constante conocido y k una parte pequeña de la clave buscada. En general, c es el texto claro o el texto cifrado que se combina con la clave o con una clave derivada.
2) *Medición del consumo de potencia*. En el segundo paso se mide el consumo de potencia del dispositivo criptográfico mientras cifra o descifra n bloques de datos (textos claros o textos cifrados).
Para cada una de estas n ejecuciones de cifrado o descifrado, el atacante necesita conocer el valor, c, correspondiente que está implicado en el cálculo del resultado intermedio mencionado en el paso 1. Estos datos conocidos pueden escribirse como un vector $C = (c_1, \ldots, c_n)$, donde c_i denota el dato en la i-ésima ejecución del cifrado/descifrado.
Durante cada una de estas ejecuciones, el atacante almacena una traza de potencia. Se denota por $T_i = (t_{i,1}, \ldots, t_{i,m})$ la traza correspondiente al bloque de datos c_i, siendo m la longitud de la traza. El atacante mide una traza para cada uno de los n bloques, de modo que las trazas se pueden representar como una matriz $T = (t_{i,j})$ de tamaño $n \cdot m$:
$$T = \begin{pmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,m} \\ t_{2,1} & t_{2,2} & \cdots & t_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ t_{n,1} & t_{n,2} & \cdots & t_{n,m} \end{pmatrix}, \quad (1)$$
siendo $t_{i,j}$ el valor del voltaje medido por el osciloscopio al ejecutar el dato c_i en el instante j.
Para un ataque DPA, es importante que las trazas medidas estén correctamente alineadas, es decir, los valores del consumo de potencia de cada columna de T deben corresponder a la misma operación. Para obtener este alineamiento en las trazas, la señal del disparador (*trigger*) del osciloscopio necesita ser generada de modo que el osciloscopio registre el consumo de potencia de modo exacto de cada secuencia de operaciones durante cada proceso de cifrado o descifrado.
En el caso de no disponer de la señal del disparador, se debe realizar una alineación de las trazas mediante coincidencia de patrones.
3) *Cálculo de valores intermedios hipotéticos*. En este paso se determina un valor intermedio hipotético para cada elección de k, denotándose por $K = (k_1, \ldots, k_l)$ las posibles elecciones y por l su número. Los elementos del vector K suelen llamarse claves hipotéticas.
Dado el vector de datos C y la clave hipotética K, un atacante puede calcular fácilmente los valores hipotéticos intermedios para las n ejecuciones de cifrado y para las l claves hipotéticas. Estos valores determinan una matriz $V = (v_{i,j})$, de tamaño $n \cdot l$, cuyos elementos vienen dados por:
$$v_{i,j} = f(c_i, k_j), \quad 1 \le i \le n, \quad 1 \le j \le l, \quad (2)$$
es decir, se tiene que

$$V = \begin{pmatrix} v_{1,1} & v_{1,2} & \cdots & v_{1,m} \\ v_{2,1} & v_{2,2} & \cdots & v_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ v_{n,1} & v_{n,2} & \cdots & v_{n,m} \end{pmatrix}. \quad (3)$$

Las columnas de V contienen los resultados intermedios calculados y el objetivo del ataque DPA es determinar cuál de las columnas de V ha sido procesada durante las n rachas de cifrado o descifrado. En cuanto se conozca qué columna ha sido procesada en el dispositivo atacado, se conocerá la clave del dispositivo.

4) *Asignación de valores intermedios a los valores de consumo de potencia*. Este paso asigna los hipotéticos valores intermedios de V a una matriz $H = (h_{i,j})$, de tamaño $n \cdot l$, de valores hipotéticos de consumo de potencia. Para ello, el atacante utiliza diferentes técnicas de simulación.
   La calidad de la simulación depende en gran medida del conocimiento que el atacante tenga del dispositivo. Los modelos de potencia más utilizados para asignar V a H son los modelos basados en la distancia y el peso de Hamming.
5) *Comparación de los valores hipotéticos de consumo de potencia con las trazas de potencia*. El último paso del ataque consiste en comparar los valores hipotéticos del consumo de potencia con los valores del consumo de potencia medidos.
   Cada columna de H se compara con todas las columnas de T, es decir, el atacante compara los valores hipotéticos de consumo de potencia de cada clave hipotética con las trazas almacenadas en cada posición. El resultado de esta comparación es otra matriz $R = (r_{i,j})$, de tamaño $l \cdot m$, de modo que cada elemento $r_{i,j}$ contiene los resultados de la comparación entre la columna i-ésima de H y la j-ésima de T. Dependiendo del modelo de potencia utilizado, podrá existir una relación entre el consumo de potencia hipotético y el consumo de potencia real. Para determinar esta relación se utilizan métodos estadísticos.

Es claro que cuantas más trazas mida un atacante, más elementos habrá en cada columna de las matrices H y T, en cuyo caso será más probable que en R aparezcan datos destacados puesto que las relaciones entre las columnas serán más fáciles de determinar.

A. Obtención de trazas y calibración del osciloscopio

En los ataques por canal lateral se emplea un osciloscopio digital para obtener las trazas cuando el dispositivo está ejecutando operaciones criptográficas. Los osciloscopios pueden ser configurados de diferentes maneras, conforme a las características de los elementos a medir. Los principales parámetros que deben ser tenidos en cuenta para el almacenamiento de las trazas son los siguientes:

1) *Sensibilidad vertical*. Proporciona el rango de valores verticales (voltaje) que mide el osciloscopio, que suele ser un rango alrededor del cero, típicamente, ±100 mV, ±200 mV, ±500 mV. Cada osciloscopio posee un valor por defecto de estos valores que el usuario debe ajustar a las necesidades del problema a resolver.
2) *Frecuencia de muestreo*. Este valor especifica la velocidad a la que se sincroniza el convertidor analógico a digital en el osciloscopio para digitalizar la señal de entrada. Tal valor puede ser especificado en una frecuencia de captura (1 GS/s) o en un intervalo de tiempo entre capturas (1 ns).
3) *Resolución*. Los osciloscopios digitales discretizan los valores verticales (voltaje), de modo que la resolución proporciona el número de valores intermedios que pueden ser distinguidos por el osciloscopio, dentro del rango del valores del voltaje configurado por la sensibilidad vertical. Este valor se mide en bits y el número de valores intermedios se puede calcular como $2^{\text{resolution}}$. Por ejemplo, con una sensibilidad vertical de ±100 mV y 8 bits de resolución (256 valores intermedios), los valores adyacentes tienen una separación de 200/256 mV.
4) *Tamaño de memoria*. Los osciloscopios digitales almacenan valores de entrada en una memoria interna. Dependiendo de la cantidad de entradas capturadas por unidad de tiempo (sensibilidad horizontal), los datos pueden ser transferidos directamente a un PC en flujo (*streaming*) o en bloque (*block*). En este último caso, la longitud de la traza capturada está limitada por la memoria interna del osciloscopio.
5) *Offset de corriente continua (DC)*. El offset DC permite al usuario añadir un voltaje de corriente continua constante a la señal de entrada. Como ya se ha dicho, la sensibilidad vertical siempre está centrada en el valor cero; no obstante, cuando la señal es pequeña y distante de este valor, el offset puede ser utilizado para llevar la señal al rango adecuado para conseguir la sensibilidad vertical óptima.

Antes de iniciar el proceso de captura de las trazas, se deben especificar los parámetros a utilizar. Una vez que la captura finaliza, los datos son transferidos desde la memoria del osciloscopio a la memoria del PC. Las trazas capturadas son vectores de muestra y el tamaño de cada muestra es igual al parámetro de resolución. Este valor de muestra se presenta en bruto y el valor del voltaje se puede calcular como

$$V = S \cdot \frac{Raw}{Max} + O, \quad (4)$$

donde S es el máximo de la sensibilidad vertical, es decir, 50 mV, 100 mV o 200 mV; Raw es el valor en bruto; Max es el valor máximo representado por la resolución y O es el offset DC.

B. Método de programación

El primer paso a considerar en la implementación software es la determinación del método de programación que se va a utilizar, en función de los objetivos planteados y de las clases a lograr. Así, a la hora de establecer cuál es el mejor método de programación, se trata de decidir entre dos paradigmas de la programación para la reutilización y extensibilidad de código, el uso de herencia o el uso de plantillas (*templates*). En teoría, la herencia permite hacer desarrollo de código más limpio pero menos eficiente.

Para evaluar hasta qué punto penaliza el uso de herencia, se han desarrollado dos versiones de una parte crítica del código,

con herencia y plantilla, y se han medido tiempos de procesamiento en cada una de las versiones. En la sección IV se indicarán con más detalle las versiones de código desarrollado y los resultados de las pruebas.

En la implementación de la versión con plantilla, ha sido necesario trasladar este método a varias clases. Se han realizado varias optimizaciones pero aun así el código no ha quedado lo suficientemente limpio. En la sección IV se comentarán los resultados de las pruebas y la decisión final sobre la implementación.

C. Representación de las trazas y su almacenamiento en memoria

Dado que la herramienta básica para obtener datos sobre los que trabajar en los ataques DPA son las trazas, se ha estudiado, en primer lugar, la manera más eficiente de almacenar en memoria los valores de las trazas capturadas. Uno de los principales objetivos es el de intentar minimizar la memoria usada, por lo que cada elemento de la traza deberá ocupar un espacio lo más cercano posible a la resolución del osciloscopio. Esto lleva consigo la ventaja de que además de almacenar más datos, se procesan más rápidamente, gracias al menor número de fallos de caché.

Se ha implementado otro punto de referencia para comprobar si es mejor representar las matrices que definen las trazas mediante columnas o filas para ser almacenadas en memoria y con el fin de determinar qué efectos tiene dicha representación sobre los fallos en la caché, es decir, analizar su rendimiento.

D. Implementación

En el desarrollo de la implementación, las clases mas importantes representadas son las siguientes:

CTrace: clase para contener una traza. Los elementos de la traza pueden tener una resolución de 8, 16 o 32 bits (CTrace8, CTrace16, CTrace32, respectivamente). Esta clase abstrae a otras clases de la resolución de los elementos (ver Fig. 6).

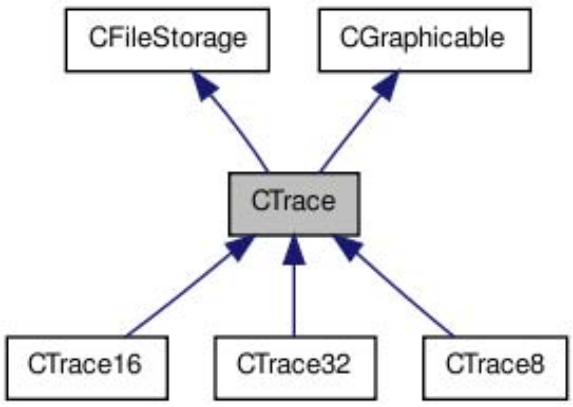

Figura 6. Clase CTrace.

Esta clase hace uso de las siguientes funciones.

- getVoltConversion(): devuelve el atributo voltConversion, es decir, el multiplicador para pasar las mediciones de *Raw* a voltaje.
- getDisplacement(): devuelve el atributo displacement, que es valor del offset de DC aplicado en las muestras.
- getSamplingRate(): devuelve el atributo samplingRate, esto es, el tiempo entre muestras.
- getValue(size_t pos)}: devuelve el valor de la traza, en *mV*, en la posición especificada.
- getRawValue(size_t pos): devuelve el valor obtenido en el osciloscopio.
- setValue(uint32_t value, size_t pos): establece la posición de la traza especificada con el valor dado.
- size_t size(): obtiene el tamaño de la traza (número de valores).
- eraseLast(): borra el último elemento de la traza.
- sizeElem(): obtiene la resolución de los valores en bruto, en Bytes.

CTimeSlice: clase que representa los valores de múltiples trazas en un instante de tiempo. Sobre estas trazas se realizará la correlación por distintos métodos, con el fin de obtener la clave u otros datos secretos (CTimeSlice8, CTimeSlice16, CTimeSlice32). Esta clase también requiere abstracción de la resolución de los elementos (ver Fig. 7).

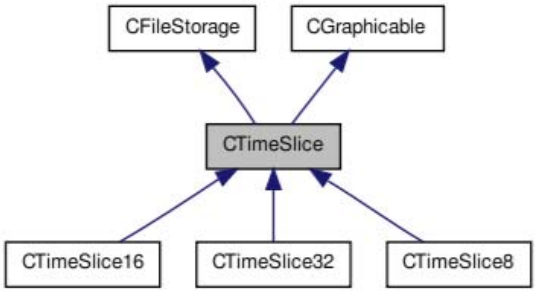

Figura 7. Clase CTimeSlice.

Las funciones incluidas en esta clase son las siguientes.

- getVoltConversion: ver CTrace.
- getDisplacement: ver CTrace.
- getValue: ver CTrace.
- getRawValue: ver CTrace.
- setValue: ver CTrace.
- size: ver CTrace.
- sizeElem: ver CTrace.
- getValue: ver CTrace.
- toDisk(int fd): almacena el objeto en el descriptor del fichero (ver §IV-B).
- fromDisk(int fd): recupera el objeto almacenado en el descriptor del fichero (ver §IV-B). El objeto deberá haber sido almacenado previamente en el fichero señalado por el descriptor mediante la función toDisc.
- toPng(char*fileName, list<string> *gnuplotCmds=NULL): produce un fichero de tipo png con la interpretación gráfica del objeto (ver §IV-C).

CTraceSet: clase que contiene un conjunto de trazas (ver Fig. 8). De esta clase se puede extraer CTimeSlice indicando el instante de tiempo. Además, se pueden obtener valores

estadísticos, como la media y la varianza, del conjunto de trazas, útil para obtener, por ejemplo, la relación señal/ruido. Pueden ser preprocesables (alineamientos). Se han implementado dos algoritmos de alineamiento: mediante integración (*Integration*) y mediante reconocimiento de patrones (*Square match pattern*).

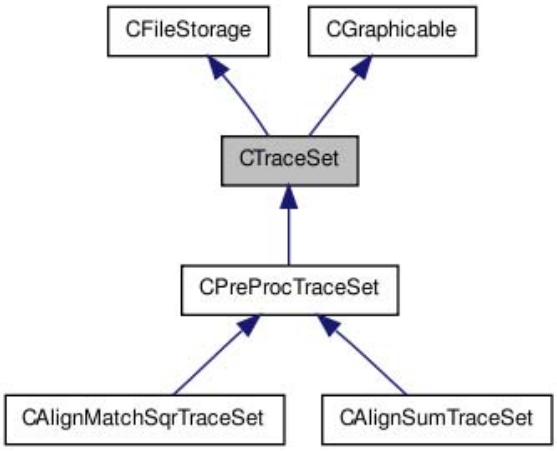

Figura 8. Clase CTraceSet.

Las funciones que se emplean en esta clase se describen a continuación.

- getLenTrace(): devuelve el número de elementos de cada traza.
- getLenSlice(): devuelve el número de elementos en cada instante de tiempo.
- addTrace(CTrace *trace): inserta una traza al final del conjunto de trazas. Las trazas se almacenan como referencias. El destructor CTraceSet libera la memoria. Este método sólo puede ser utilizado en el modo de alineamiento.
- statMode(): cambia la representación interna al modo estadístico. Mejora la ejecución de las operaciones que requieren el uso de instantes de tiempo.
- getNTrace(int pos): devuelve una copia de la traza almacenada en la posición especificada. El que la llama debe liberar el valor devuelto cuando se requiera (es más rápido en el modo de alineamiento).
- getNSlice(int pos): devuelve una copia del instante de tiempo almacenado en la posición especificada. El que la llama debe liberar el valor devuelto cuando se requiera (es más rápido en el modo estadístico).
- getNTraceRef(int pos): devuelve el puntero de la traza almacenada en la posición especificada. El que la llama no debe liberar el valor devuelto (es más rápido en el modo de alineamiento).
- getNSliceRef(int pos): devuelve un puntero al instante de tiempo almacenado en la posición especificada. El que la llama no debe liberar el valor devuelto (es más rápido en el modo de alineamiento).
- meanTraces(): devuelve un vector (CStatTrace) con el valor de la media aritmética de todas las trazas almacenadas. El que la llama debe liberar el valor devuelto cuando se requiera (es más rápido en el modo estadístico).
- varianceTraces(CStatTrace *mean=NULL): devuelve un vector (CStatTrace) con el valor de la varianza de todas las trazas almacenadas. El que la llama debe liberar el valor devuelto cuando se requiera (es más rápido en el modo estadístico).
- toDisk: ver CTimeSlice.
- fromDisk}: ver CTimeSlice.
- toPng: ver CTimeSlice.

CStatTrace: esta clase proporciona algunos datos estadísticos obtenidos a partir de varias trazas como la media, varianza, etc. (ver Fig. 9).

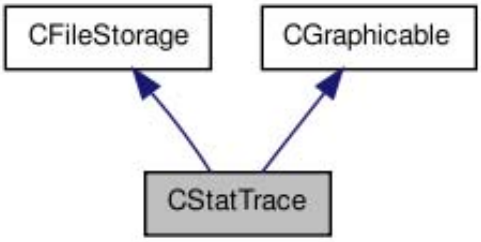

Figura 9. Clase CStatTrace.

Las principales funciones que emplea esta clase son:

- getSamplingRate: ver CTrace.
- getValue: ver CTrace.
- setValue: ver CTrace.
- size: ver CTrace.
- toDisk: ver CTimeSlice.
- fromDisk: ver CTimeSlice.
- toPng: ver CTimeSlice.

Como se puede ver en las Figs. 6-9, las clases anteriores hacen uso de otras dos clases, CFileStorage y CGraphicable. La primera de ellas contiene las funciones toDisk y fromDisk y especifica las subclases que pueden ser almacenadas en un fichero (ver Fig. 10). Por su parte, la segunda (ver Fig. 11) contiene la función toPng y señala las subclases que pueden ser imprimidas, si está instalado gnuplot.

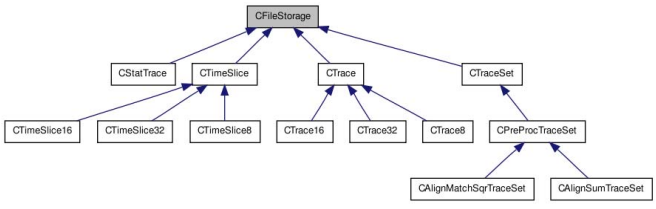

Figura 10. Clase CFileStorage}.

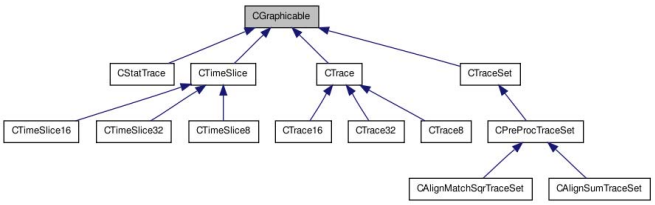

Figura 11. Clase CGraphicable.

IV. RESULTADOS EXPERIMENTALES

Durante el diseño de este conjunto de herramientas se han evaluado varias opciones con el fin de resolver distintos problemas.

En primer lugar, y dado que tales herramientas se pretenden poner a disposición de los investigadores interesados, se ha estudiado la mejor forma de abstraer al usuario de propiedades de bajo nivel a la hora de trabajar con las trazas. De forma análoga, se ha analizado cómo se almacenarían las mismas en un soporte físico de forma óptima, como un disco duro; o qué clases podrían tener representación gráfica y qué tipo de representación gráfica se amoldaría mejor a cada clase.

Por todo ello, se han realizado los siguientes estudios.

A. Comparación entre plantillas y herencias

A la hora de tratar el diseño del software de almacenamiento de trazas (ver sección III), se han implementado dos posibles soluciones con el fin de analizar las ventajas e inconvenientes de cada una de ellas. Estas dos soluciones resuelven el problema de la abstracción en la precision de las trazas de dos modos distintos.

La implementación mediante plantillas proporciona mejores tiempos teóricos de ejecución que los correspondientes a las herencias, dado que las herencias tienen que resolver los métodos *virtuales* en tiempo de ejecución. No obstante, en la práctica, las optimizaciones de los compiladores pueden igualar dichos tiempos. En teoría la penalización de la versión con herencia es del 10%, si bien, en la práctica, el compilador con las opciones de optimización resuelve las llamadas a métodos virtuales, con lo que la diferencia es mínima.

Dado que se trata de una *toolbox*, es decir, un conjunto de herramientas que cualquier usuario puede utilizar y que está diseñado para ser extendido a su conveniencia, es fundamental elegir un método que ofrezca una alta claridad en el código. Con relación a este tema, hay que diferenciar entre las clases de más bajo nivel y las clases de más alto nivel que, por su parte, hacen uso de las primeras:

- *Clases de bajo nivel con plantillas*. La codificación mediante plantillas (ver Tabla I) sólo requiere una clase, ya que la resolución de la traza viene dada por la plantilla (template<class T> class CTrace). Cuando se ejemplifica dicha clase, se especifica la resolución denotando el tipo que se le asignará a T de la siguiente manera: CTrace<uint8_t> trace;. Este tipo puede ser uint8_t, uint16_t o uint32_t, representando resoluciones de 8, 16 y 32 bits, respectivamente.

TABLA I. EJEMPLO DE CÓDIGO DE BAJO NIVEL CON PLANTILLA.

```
template <class T> class CTrace : public valarray<T>{
protected:
    uint64_t samplingRate;
    double voltConversion;
public:
    CTrace (int fd);
    CTrace (uint64_t samplingRate, double voltConversion,
            T* vals, unsigned int lengh);
    CTrace (uint64_t samplingRate, double voltConversion);
    CTrace (CTrace<T> &trace, unsigned int begin,
            unsigned int len);
    double getVoltConversion (){return voltConversion;};
    uint64_t getSamplingRate (){return samplingRate;};
    int toDisk (int fd);
    int toPng (char *fileName, list<string> *gnuplotCmds
            = NULL);
    };
```

- *Clases de bajo nivel con herencias*. La codificación usando herencias requiere de una clase por cada resolución posible, por ejemplo, CTrace8, CTrace16 y CTrace32 (ver Tabla II para una resolución de 8 bits) y una clase abstracta que sea superclase de las anteriores (ver Tabla III).

TABLA II. EJEMPLO DE CÓDIGO DE BAJO NIVEL CON HERENCIA.

```
class CTrace8 : public CTrace{
protected:
    vector <uint8_t> values;
public:
    CTrace8 (uint64_t samplingRate, double voltConversion,
            double displacement, uint8_t* values, size_t size);
    CTrace8 (uint64_t samplingRate, double voltConversion,
            double displacement, size_t size);
    CTrace8 ();
    float getValue (size_t pos){return float(values[pos]) *
            voltConversion + displacement;};
    uint32_t getRawValue (size_t pos) {return uint32_t
            (values[pos]);};
  int setValue (uint32_t value, size_t pos)
            {values[pos] = value;};
    size_t size (){return values.size();};
    void eraseLast (){values.pop_back();};
    size_t sizeElem (){return 8;};
    int toDisk (int fd);
    int fromDisk (int fd);
    int toPng (char *fileName, list<string> *gnuplotCmds
            = NULL);
};
```

TABLA III. EJEMPLO DE CÓDIGO DE BAJO NIVEL CON PLANTILLA.

```
class CTrace : virtual public CFileStorage,
            virtual public CGraphicable{
protected:
    uint64_t samplingRate;
    double voltConversion;
    double displacement;
public:
    CTrace (uint64_t samplingRate, double voltConversion,
            double displacement);
    CTrace ();
       // get methods for class parameters
    double getVoltConversion (){return voltConversion;};
    double getDisplacement (){return displacement;};
    uint64_t getSamplingRate (){return samplingRate;};
    virtual float getValue (size_t pos) = 0;
```

```
    virtual uint32_t getRawValue (size_t pos) = 0;
        // returns the oscilloscope value.
    virtual int setValue (uint32_t value, size_t pos) = 0;
        // set oscilloscope value.
    virtual size_t size () = 0; // returns its size.
    virtual void eraseLast () = 0;
        // erases last element (req. by CTraceSet::statMode)
    virtual size_t sizeElem () = 0;
        // used to know the value resolution (subClass used).
};
```

- *Clases de alto nivel con plantillas*. Cuando se emplean plantillas, las clases de alto nivel que hacen uso de clases de más bajo nivel, también se han de implementar con plantillas, de forma que se les pueda indicar la resolución de las trazas representadas por las clases de bajo nivel (ver Tabla IV). De este modo, las plantillas se extienden por gran parte del código de la toolbox.

TABLA IV. EJEMPLO DE CÓDIGO DE ALTO NIVEL CON PLANTILLA.

```
template <class T> class CTraceSet {
protected:
    vector < CTrace <T> > traces;
    int lenTrace;
    uint64_t samplingRate;
    double voltConversion;
public:
    CTraceSet (uint64_t samplingRate, double voltConversion,
               int lenTrace);
    CTraceSet (vector <CTrace<T> > &traces);
    CTraceSet (CTraceSet<T> &set, unsigned int begin,
               unsigned int len);
    CTraceSet (int fd);
    int getLenTrace (){return lenTrace;};
    uint64_t getSamplingRate (){return samplingRate;};
    double getVoltConversion (){return voltConversion;};
    int size (){return traces.size();};
    int addTrace (const CTrace<T> &trace);
    CTrace <T> getNTrace (int pos);
    CTimeSlice <T> getNSlice (int pos);
    virtual CTrace <double> meanTraces ();
    virtual CTrace <double> varianceTraces (CTrace <double>
                                            *mean = NULL);
    virtual int alignTraces (){};
    int toPng (char *fileName);
    int toDisk (int fd);
};
```

- *Clases de alto nivel con herencias*. En este caso, las clases de alto nivel hacen uso de la clase abstracta que es una superclase de las clases que especifican la resolución de las trazas (CTrace8, CTrace16, CTrace32), lo cual permite abstraerse de la subclase que está siendo utilizada (ver Tabla V).

TABLA V. EJEMPLO DE CÓDIGO DE ALTO NIVEL CON HERENCIA.

```
class CTraceSet : virtual public CFileStorage,
                  virtual public CGraphicable{
protected:
    vector <CTrace*> traces;
    vector <CTimeSlice*> slices;
    uint64_t samplingRate;
    size_t lenTrace;
    size_t lenSlice;
    bool StatMode;
    float getValue (size_t NTrace, size_t NSlice);
public:
    CTraceSet ();
    CTraceSet (vector <CTrace*> &traces);
    CTraceSet (CTraceSet &set, size_t begin, size_t len);
    ~CTraceSet ();
    uint64_t getSamplingRate (){return samplingRate;};
    bool getStatMode (){return StatMode;};
    int getLenTrace (){return lenTrace;};
    int getLenSlice (){return lenSlice;};
    int addTrace (CTrace *trace); // only in alignment mode.
    void statMode ();
    CTrace* getNTrace (int pos);
    CTimeSlice* getNSlice (int pos);
    CTrace* getNTraceRef (int pos); // only in alignment mode.
    CTimeSlice* getNSliceRef (int pos);
        // only in statistical mode.
    virtual CStatTrace* meanTraces ();
    virtual CStatTrace* varianceTraces (CStatTrace* mean
                = NULL);
    int toPng (char *fileName, list<string> *gnuplotCmds
                = NULL);
    int toDisk (int fd);
    int fromDisk (int fd);
};
```

Para la implementación final, se han elegido las herencias, ya que se genera una codificación más limpia para las clases de alto nivel, que son más susceptibles de modificación o ampliación por la comunidad científica (por ejemplo, nuevos métodos estadísticos). Además, consideramos que la penalización que introduce el uso de herencia es asumible. Las clases de bajo nivel son simples y una vez optimizadas, es menos probable que se plantee su modificación.

B. Interfaz para almacenamiento en disco

Esta interfaz, cuyo diagrama de herencias se puede encontrar en la Fig. 10, permite almacenar en disco los objetos representados por las clases que la usan: CTrace, CTimeSlice, CTraceSet y CStatTrace.

La necesidad de esta interfaz surge por el coste de tiempo necesario para capturar el número de trazas requerido para el análisis estadístico. El almacenamiento permite capturar trazas en distintas tandas, así como hacer uso de ellas en momentos o localizaciones diferentes.

Dada la cantidad de información a almacenar, ha sido preciso intentar optimizar el espacio requerido para ello. Con este fin los atributos se almacenan de forma secuencial, ajustándolos al mínimo número de bits requerido para cada uno de ellos.

A modo de ejemplo, un objeto de la clase CTrace almacenaría la siguiente información.

- *Muestreo* (samplingRate): indica el muestreo, en picosegundos, que el osciloscopio utilizó para tomar la traza.
- *Conversión* (voltConversion): es el valor de la sensibilidad vertical que utilizó el osciloscopio para tomar la traza.
- *Desplazamiento* (displacement): señala el desplazamiento que hay que añadir para obtener el valor

de voltaje real.

- *Resolución* (bytesData): es el número de bits que ocupa cada dato (punto de la traza).
- *Tamaño* (sizeData): corresponde al número de datos que contiene la traza.
- *Datos* (Data): son los datos expuestos de forma secuencial.

Cada una de las clases almacena los datos correspondientes de modo secuencial como sigue:

- CTrace: samplingRate || voltConversion || displacement || bytesData || sizeData || Data …
- CTimeSlice: voltConversion || displacement || bytesData || sizeData || Datav
- CTraceSet: samplingRate || lenTrace || lenSlice || StatMode || data resolution || if (StatMode) Data Slice 0 … || Data Slice 1 … else Data Trace 0 … || Data Trace 1 …
- CStatTrace: samplingRate || sizeData || Data …

C. *Interfaz para representaciones gráficas*

Esta interfaz ha sido implementada para las clases que pueden ser representadas gráficamente. Como se puede comprobar en el diagrama de herencias de la Fig. 11, las clases que la implementan son CTrace, CTimeSlice, CTraceSet y CStatTrace.

Dependiendo de cada clase, el tipo de representación gráfica será diferente:

- CTrace: se representa como una gráfica en dos dimensiones. En el eje de abscisas se muestra el tiempo, mientras que en el eje de ordenadas se presenta el voltaje. En la Fig. 12 se muestra la representación de una traza de longitud 1000, habiéndose tomado medidas cada 2 ns.

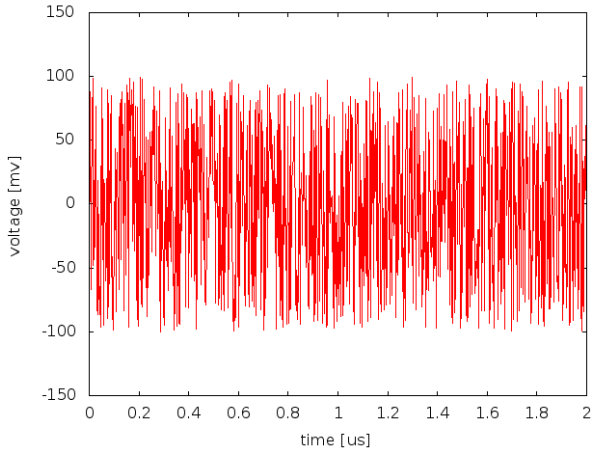

Figura 12. Gráfica proporcionada por CTrace.

- CTimeslice: se muestra como un histograma, por lo que hace referencia al número de ocurrencias de distintos valores del voltaje en un instante de tiempo determinado. La Fig. 13 presenta el histograma de los valores de la traza de la Fig. 12 para el primer instante de tiempo.

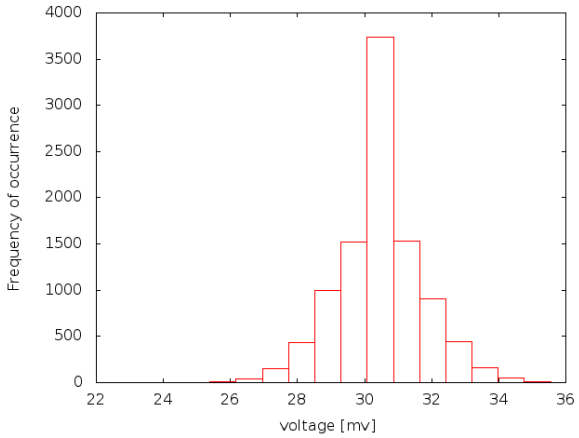

Figura 13. Histograma proporcionado por CTimeSlice.

- CTraceSet: se muestra como una gráfica en tres dimensiones. En realidad, la gráfica aparece en dos dimensiones, estando la tercera dimensión representada por la intensidad del color. Para realizar dicha gráfica se superponen varias trazas representadas como en CTrace. Es claro que cuanto mayor sea el número de trazas que pasen por un determinado punto, más intenso será el color en dicho punto. En la Fig. 14 se puede observar la salida de esta clase para la traza de la Fig. 12 y permite hacerse una idea del valor medio y de la varianza en los distintos instantes de tiempo.

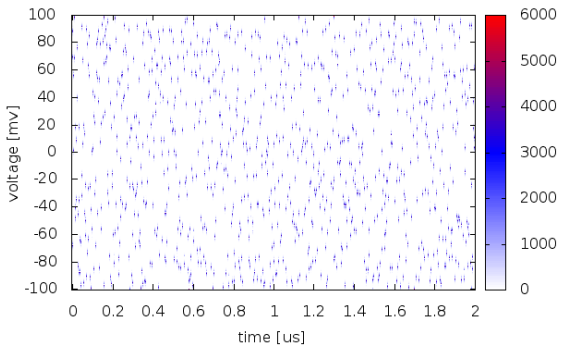

Figura 14. Gráfica proporcionada por CTraceSet.

V. Conclusiones

A partir del diseño y de los resultados experimentales presentados en las secciones III y IV, se ha llegado a las siguientes conclusiones con el fin de avanzar en la implementación de las herramientas software para realizar ataques por canal lateral.

Con relación al uso del método de programación (ver §III-B), la Tabla VI muestra una comparación de estas dos posibles soluciones. Según lo señalado anteriormente, consideramos que, globalmente, desarrollar el código utilizando herencia aporta mayores ventajas que hacerlo utilizando plantillas, ya que la penalización de tiempos es inferior a un 10% y el código

sigue una estructura orientada a objetos, lo cual es más apropiado con vistas a desarrollar una caja de herramientas o toolbox.

TABLA VI. COMPARACIÓN ENTRE PLANTILLAS Y HERENCIAS.

	ALTO NIVEL	BAJO NIVEL
PLANTILLAS	Las plantillas tienden a extenderse por la implementación	Solo requiere una clase
HERENCIAS	Permite abstraerse de los detalles de bajo nivel	Es necesario crear una clase por resolución

Como consecuencia, para la implementación final se han elegido las herencias, ya que se genera una codificación más limpia para las clases de alto nivel y son más susceptibles de modificación o ampliación por la comunidad científica. La inclusión de nuevos métodos o valores estadísticos, por ejemplo, es más sencilla de este modo. Además, las clases de bajo nivel son simples y una vez optimizadas es menos probable que se plantee su modificación en el futuro (ver §IV-A).

En lo que hace referencia a la forma de representar las trazas para su optimización a la hora de ser almacenadas en memoria, se ha decidido implementar los dos métodos de representación mencionados en §III-C: uno para tareas de alineación y preprocesado (ordenamiento por filas) y otro para realizar labores estadísticas (ordenamiento por columnas). El paso de un modo de representación a otro se lleva a cabo mediante un método.

Ligado a este tema y con relación al uso del procesador, también se ha evaluado el uso de múltiples hilos (alineamiento concurrente de trazas, cálculos estadísticos concurrentes, etc.) de forma que se maximice el uso de los núcleos del procesador.

Con las clases presentadas y comentadas en §III-D, se tendría resuelta la representación de las medidas tomadas, es decir, se habría determinado cómo trabajar y preprocesar la matriz T mencionada en la sección III. Además, se habrían evaluado las distintas alternativas de implementación de modo que se maximice la eficiencia y claridad del código.

Finalmente, señalar que se han desarrollado dos interfaces específicas, una para el almacenamiento en disco (ver §IV-B) y otra para las representaciones gráficas (§IV-C).

AGRADECIMIENTOS

Este trabajo ha sido parcialmente subvencionado por el Ministerio de Economía y Competitividad bajo el proyecto ProCriCiS, TIN2014-55325-C2-1-R, y por la Comunidad de Madrid bajo el proyecto S2013/ICE-3095-CIBERDINE-CM.

REFERENCIAS

[1] O. Aciiçmez, J.P. Seifert, and Ç.K. Koç, "Predicting secret keys via branch prediction", Lecture Notes in Comput. Sci., vol. 4377, pp. 225-242, 2007.

[2] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov. "Cryptographic processors-A survey". Proc. IEEE, vol. 94, 2, pp. 357-369, feb. 2006.

[3] D. Boneh, R.A. DeMillo, and R.J. Lipton, "On the importance of checking cryptographic protocols for faults", Lecture Notes in Comput. Sci., vol. 1233, pp. 37-51, 1997.

[4] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model", Lecture Notes in Comput. Sci., vol. 3156, pp. 16-29, 2004.

[5] B. Chor and R.L. Rivest, "A knapsack-type public key cryptosystem based on arithmetic in finite fields", IEEE Transactions on Information Theory, vol. 34, pp. 901-909, 1988.

[6] J. Daemen and J. Rijmen, "The Design of Rijndael: AES-The Advanced Encryption Standard", Springer Verlag, Berlín, Germany, 2002.

[7] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", IEEE Transactions on Information Theory, vol. 31, pp. 469-472, 1985.

[8] A. Fúster Sabater, L. Hernández Encinas, A Martín Muñoz, F. Montoya Vitini, and J. Muñoz Masqué, "Criptografía, protección de datos y aplicaciones. Una guía para estudiantes y profesionales", RA-MA, Madrid, Spain, 2012.

[9] D. Hankerson, A.J. Menezes, and S. Vanstone, "Guide to elliptic curve cryptography", Springer, New York, NY, USA, 2004.

[10] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side channel cryptanalysis of product ciphers", J. Comput. Secur., vol. 8, 2,3, pp. 141-158, 2000.

[11] A. Kerckhoffs. "La cryptographie militaire". Journal des sciences militaires, vol. IX, pp. 1-2, 5-38, 161-191, 1883.

[12] P.C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems", Lecture Notes in Comput. Sci., vol. 1109, pp. 104-113, 1996.

[13] P.C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis", Lecture Notes in Comput. Sci., vol. 1666, pp. 388-397, 1999.

[14] S. Mangard, E. Oswald, and T. Popp, "Power analysis attacks: Revealing the secrets of smart cards", Advances in Information Security, Springer Science+Business Media, NY, USA, 2007.

[15] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, "Handbook of Applied Cryptography", CRC Press, Inc., Boca Raton, FL, USA, 1997.

[16] J.-J. Quisquater and D. Samyde, "ElectroMagnetic Analysis (EMA): Measures and counter-measures for smart cards", Lecture Notes in Comput. Sci., vol. 2140, pp. 200-210, 2001.

[17] R.L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, vol. 21, 2, pp. 120-126, 1978.

[18] S.P. Skorobogatov, "Semi-invasive attacks-A new approach to hardware security analysis", PhD thesis, University of Cambridge, Darwin College, UK, 2005. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-630.pdf.

Alberto Fuentes received his MSc. degree in Computer Science from the KTH Kungliga Tekniska Högskolan, Stockholm, Sweden in 2007. He has been employed by INTA as computer security specialist from 2007 to 2011 evaluating products and systems with Common Criteria and ITSEC standards and from 2011 to 2013 in EADS as common criteria consultant and I+D researcher. He has wide experience in PKI platforms, ad-hoc application or system penetration testing and system vulnerability analysis, and is currently involved in the design and implementation of software tools to carry out side channel attacks to cryptographic devices, pursuing his Ph.D. degree.

Luis Hernández obtained his Graduate in Mathematics at the University of Salamanca (Spain) in 1980 and his Ph.D. in Mathematics at the same university in 1992. He is a researcher at the Department of Information Processing and Cryptography (TIC) at the Institute of Physical and Information Technologies (ITEFI), Spanish National Research Council (CSIC) in Madrid (Spain). He has participated in more than 30 research projects. He is author of 9 books, 9 patents, more than 150 papers, more than 100 contributions to workshops and conferences, he has supervised 3 doctoral thesis and has served as referee for different SCI journals and for many international conferences. His current research interests include Cryptography and Cryptanalysis of public key cryptosystems (RSA, ElGamal and Chor-Rivest), Cryptosystems based on elliptic and hyperelliptic curves, Graphic Cryptography, Pseudorandom number generators, Digital signature schemes, Authentication and Identification protocols, Crypto-Biometry, Secret sharing protocols, Side channel attacks, and Number Theory problems. Dr. Hernández belongs to the International Association for Cryptologic Research (IACR).

Agustín Martín obtained his Graduate in Physics at the Complutense University of Madrid (Spain) in 1988 and his Ph.D. in Physics at the same university in 1995. He is currently a researcher at the Department of Information Processing and Cryptography (TIC) of the Institute of Physical and Information Technologies (ITEFI), Spanish National Research Council (CSIC), in Madrid. He started his research career dealing with the development of numerical methods in electromagnetics to study the interaction of electromagnetic radiation, especially in the radiofrequency range, with different objects as satellites or biological tissues. His latest research interests are focused on the analysis of possible vulnerabilities of radiofrequency identification systems (RFID) and the study of techniques of physical attacks to cryptographic devices. He has published 20 research papers in international scientific journals and is co-author of a book and more than 60 contributions to peer reviewed workshops and conferences. He has participated in 20 research projects and contracts, has taught several courses and seminars, and is co-author of 3 patents. Dr. Martin is member of the International Association for Cryptologic Research (IACR).

Bernardo Alarcos received his Ph.D. degree in telecommunications from the University of Alcalá, Spain in 2004. He received his M.Sc. degree in Telecommunications Engineering from the UPM, Polytechnic University of Madrid, Madrid, Spain in 1979. Since 1999, he has been employed as an Associate Professor at the Alcalá University. The subject of the thesis was about proposal of security architecture in Programmable Networks. He has participated in more than 5 projects developing activities related with the cybersecurity. Last projects that is working on is about Honeynet services using virtualization. Since 2010, he has been a member of Research Institute for the Police Science (IUICP), in the area of Computer forensics.