UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Posgrado en Ciencia e Ingeniería de la Computación

ALGORITMOS DE EXPONENCIACIÓN MODULAR Y SU PROTECCIÓN EN SISTEMAS CRIPTOGRÁFICOS

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

DOCTOR EN CIENCIAS (COMPUTACIÓN)

P R E S E N T A:

DAVID TINOCO VARELA

VLADISLAV KHARTCHENKO
FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN

México D.F. a Julio de 2014

Resumen

Distintos tipos de ataques físicos tales como los Side Channel Attacks (SCA) y los Fault Attacks (FA) son comúnmente utilizados contra una gran cantidad de criptosistemas, principalmente contra aquellos basados en la exponenciación modular y contra aquellos utilizados en sistemas embebidos. Este tipo de ataques suelen utilizar la observación y el análisis del comportamiento de los distintos algoritmos de exponenciación dentro de lo sistemas electrónicos, mediante esta observación se obtiene información relevante acerca de las claves secretas utilizadas en las comunicaciones y transacciones realizadas a través de ellos.

Distintos tipos de SCA, basan su funcionamiento en el comportamiento de ciertos valores numéricos durante la ejecución dentro de un criptodispositivo, tal es el caso del ataque N-1, y otros ataques, como los FA, basan su funcionamiento en ciertos cálculos realizados con un valor erróneo de salida del criptosistema, tal es el caso de calcular el Símbolo de Jacobi a dichos valores.

La finalidad de esta tesis esta enfocada en evitar que SCA y FA puedan vulnerar la seguridad de los sistemas de comunicaciones, seguridad dada por medio de la exponenciación modular de un algoritmo criptográfico. Se presenta una técnica que es capaz de evitar tanto SCA como FA, de una forma simple y no costosa. La técnica presentada permitirá evitar ataques como el N-1 y ataques basados en el cálculo del Símbolo de Jacobi.

También se presenta un ataque realizado contra un algoritmo de exponenciación, ataque basado tanto en SCA como en FA.

Agradecimientos

Existen muchas personas a las cuales agradecer el apoyo brindado para la culminación de esta meta, que más quisiera que poder mencionar a todas y cada una de ellas, sin embargo el espacio es poco, por este motivo, solo podré mencionar a algunas cuantas.

A mis padres, que siempre muestran su apoyo hacia mi, aunque muchas veces no estén de acuerdo conmigo, y aunque yo no entienda que lo hacen.

A mi tía Guadalupe Silva, quién creyó desde un principio en que podía lograrlo.

A los miembros de l comité totoral, ya que estuvieron dispuestos a darme su tiempo cuando yo lo solicite.

A todas las secretarias tanto de IIMAS como de FES-C, que han estado en la mejor disposición para atender mis dudas o necesidades de la mejor manera posible.

Y principalmente y de forma muy específica quisiera darle las gracias a mi tutor, el Dr. Vladislav Khartchenko, por haberme aceptado como su alumno, por su tutoría y por todo lo que ha hecho por mí.

A la UNAM, mi casa de estudios y la cual me ha dado tanto.

A CONACyT por la beca otorgada para la realización de mis estudios.

Al IIMAS y a FES-C, por las instalaciones y facilidades brindadas.

Se que faltan personas por mencionar, pero, no por eso no les estoy agradecido, para todas esas personas que han estado ahí de una u otra forma, gracias.

Índice general

1. Introducción. **1**
- 1.1. Planteamiento. . . . 3
- 1.2. Pasos realizados. . . . 3
- 1.3. Visión sobre los sistemas de seguridad. . . . 5
  - 1.3.1. Desarrollo público y privado. . . . 5
  - 1.3.2. Ocultación de ataques reales. . . . 7
  - 1.3.3. TEMPEST. . . . 8
  - 1.3.4. Visualización de los side channel attacks en el mundo real. . . . 9
  - 1.3.5. Algunos comentarios. . . . 10
- 1.4. Algunas herramientas. . . . 11
- 1.5. Artículos publicados. . . . 11
- 1.6. Organización de la Tesis. . . . 12

2. Conceptos generales. **13**
- 2.1. Aritmética modular. . . . 13
  - 2.1.1. Congruencias y sus propiedades básicas. . . . 13
  - 2.1.2. Propiedades de las congruencias. . . . 14
  - 2.1.3. Inversos multiplicativo mod n. . . . 14
  - 2.1.4. Residuos cuadrados modulo n. . . . 15
  - 2.1.5. El símbolo de Jacobi. . . . 16
- 2.2. Sistemas criptográficos. . . . 18
  - 2.2.1. Diffie Hellman. . . . 18
  - 2.2.2. RSA. . . . 19
- 2.3. Exponenciación modular. . . . 20
- 2.4. Pequeños dispositivos y *Smart Cards*. . . . 20

3. Algoritmos de exponenciación modular. **22**
- 3.1. Algoritmos square and multiply y square and multiply always. . . . 22
- 3.2. Montgomery powering ladder. . . . 24
  - 3.2.1. Características del algoritmo Montgomery powering ladder. . . . 26
- 3.3. Atomicidad de un algoritmo. . . . 26
- 3.4. Algoritmo de Sun Da-zhi. . . . 28

4. Ataques físicos sobre la exponenciación modular. 30
4.1. Side channel attacks. 30
4.1.1. Análisis de potencia simple y diferencial. 32
4.1.2. Ataque $N-1$. 34
4.2. Fault attacks. 35
4.2.1. Test de coherencia para evitar FA. 37
4.2.2. FA usando el símbolo de Jacobi. 39
4.2.3. Ataque contra el algoritmo de Sun Da-Zhi. 42
4.2.4. Contramedidas para proteger al algoritmo de Da-Zhi. 44

5. Contramedida que puede ser implementadas en sistemas criptográficos reales. 49
5.1. Protección contra el ataque $N-1$. 49
5.1.1. Contramedidas existentes contra el ataque $N-1$. 51
5.1.2. Ventajas de la contramedida propuesta en comparación con las contramedidas existentes. 52
5.2. Explicación de la contramedida propuesta. 52
5.3. Algoritmo 10 protegido contra el ataque propuesto por Schmidt. 54
5.4. Comentarios acerca de las exponenciaciones intermedias pares. 56

6. Comentarios y conclusiones. 58

A. Otros ataques y contramedidas. 60
A.1. Ataques. 61
A.1.1. Side channel attacks. 61
A.1.2. Fault attacks. 62
A.1.3. Ataques invasivos. 62
A.2. Contramedidas. 63

B. Técnicas para mejorar el tiempo de ejecución. 65
B.1. Diversas representaciones binarias. 65
B.2. Técnicas para acelerar los algoritmos de exponenciación. 68

Índice de tablas

3.1. Comportamiento de k, en el algoritmo 7. 27
3.2. Pasoso intermedios del algoritmo 8. 29

4.1. Algoritmo 11 cuando es sometido a un ataque $N-1$. 36
4.2. Algoritmo 10 ejecutado con un FA, donde $d_{i+1} \neq d_i$. 40
4.3. Caso donde $d_{i+1} \neq d_i$. 40
4.4. Caso donde $d_{i+1} = d_i$. 41
4.5. Resultados del primer paso del ataque propuesto. 43
4.6. Resultados del segundo paso del ataque propuesto. 44
4.7. SJ de los valores de salida, cuando han sido colocados faults en el algoritmo 16. 46
4.8. SJ de los valores de salida, cuando han sido colocados faults en el algoritmo 17. 47
4.9. Características de los algoritmos modificados. 48

5.1. Algoritmo 19 ejecutado con un mensaje de entrada igual a $N-1$ y un exponente $d = 89 = 1011001$. 51
5.2. Caracteísticas de los algoritmos propuestos en comparación con otros algoritmos de interes. 56

B.1. Ejemplos de representaciones binarias signadas de un valor decimal. 68

Índice de algoritmos

1. Generación de claves en el esquema DH. 19
2. Generación de valores para RSA. 19
3. Square and multiply, left-to-right. 23
4. Square and multiply, right-to-left. 23
5. Square and multiply always, left-to-right. 24
6. Montgomery powering ladder, left-to-right. 25
7. SaM protegido por medio de atomicidad. 27
8. Algoritmo de Da-zhi. 28
9. Algoritmo KQ. 33
10. Algoritmo de Fumaroli y Vigilant. 33
11. BRIP. 34
12. Versión rápida de RSA. 36
13. Square and multiply always con test de coherencia. 38
14. Algoritmo propuesto en [1]. 39
15. Ataque propuesto en [2]. 40
16. Algoritmo de Da-zhi con atomicidad. 45
17. Algoritmo de Da-zhi con atomicidad, segunda versión. 46
18. Square and multiply always, left-to-right modificado. 50
19. BRIP modificado. 50
20. Algoritmo FV modificado. 55
21. Convierte una cadena binaria a su representación NAF. 66
22. Representación canónica de una representación binaria. 67
23. Convierte una cadena binaria a su representación MOF. 67

Capítulo 1

Introducción.

A security system is only as strong as its weakest link.
(Ferguson, Schneier)

Hoy en día, vivimos en la era de las telecomunicaciones, es decir, estamos en un mundo donde podemos transmitir y recibir cualquier tipo de dato en forma global. Esto ha permitido un intercambio de información más dinámico y eficiente, lo que a su vez ha logrado que podamos enterarnos de eventos que están sucediendo en otra parte del planeta, realizar transacciones monetarias a través de bancos internacionales, realizar compras en tiendas que están a kilómetros de distancia sin salir de casa, e incluso poder ver y conversar con una persona que se encuentra en una posición geográfica completamente distinta a la nuestra, todo esto de forma casi instantánea.

Pero no todo es perfecto, con la llegada de este tipo de comunicaciones, también han aparecido entes que buscan interceptar los datos transmitidos por los canales de intercambio, para usarlos con fines personales y/o malintencionados. En el mejor de los casos, estos intrusos solamente buscan husmear en las vidas de otras personas en forma ociosa, pero en casos mas críticos, pueden reducir los fondos de una cuenta bancaria o pueden poner en riesgo un sistema comercial completo. Es precisamente por este punto, que es necesario crear comunicaciones seguras, donde los datos transmitidos no sean fácilmente observables por un posible atacante. La protección hacia estos sistemas de datos, puede ser dada por medio del uso de sistemas criptográficos.

Un sistema criptográfico, básicamente, es un sistema que esta desarrollado en base a algoritmos de encriptación y desencriptación de datos, con la finalidad de protegerlos de entes malintencionados. Desde la antigüedad se han utilizado sistemas criptográficos (tal como el cifrado César, la escitala, etc), sin embargo, con la llegada de las computadoras, ha aparecido lo que conocemos como criptografía moderna. La criptografía moderna (llamada solamente criptografía de aquí en adelante) ha sido de gran interés y utilidad en los campos militares y políticos, sin embargo, en la actualidad es utilizada en prácticamente todo campo que requiera un intercambio de información electrónica. Estos sistemas criptográficos, los podemos encontrar en elementos tan distintos de la vida cotidiana tales como Internet, cuentas bancarias, teléfonos celulares, etc.

Diariamente, todo tipo de personas utilizan dispositivos electrónicos para realizar transacciones monetarias, comerciales y personales. Dichos dispositivos están lejos de ser grandes computadoras o servidores resguardados en un lejano edificio y custodiados por personal capacitado, por el contrario, están a la mano de cualquier persona, incluyendo posibles atacantes. Debido a esta facilidad de obtención, la protección y resguardo de este tipo de dispositivos debe de ser un tema central en cuestiones de seguridad. Actualmente, estos dispositivos contienen procesos criptográficos en su diseño, pero lamentablemente, en la década de los 90's esos pequeños elementos electrónicos mostraron tener vulnerabilidades a cierto tipo de ataques, hoy conocidos como *side channel attacks* y *fault attacks*. Estos ataques, buscan obtener las claves secretas de los criptosistemas, no por medio de cálculos matemáticos o procesos meramente computacionales, sino más bien, por medio de la medición de señales físicas que emite un dispositivo electrónico cuando este ejecuta un algoritmo criptográfico dentro de él.

Los ataques contra pequeños *criptodispositivos*, han inducido a una gran cantidad de investigadores a tratar de proteger los dispositivos mencionados, sin embargo, la protección de este tipo de micro computadoras engloba varios campos del conocimiento, tales como la ingeniería, las matemáticas y la informática, lo que hace complicado un diseño seguro e integral que incluya tanto la implementación de software dentro del aparato, así como el diseño electrónico.

Según *Ferguson* y *Schneier* [3], realizar un sistema criptográfico *completo*, que incluya *software* y *hardware* así como canales y usuarios, es sumamente complejo, por lo que cada uno de los elementos de un sistema completo, debe ser dividido en bloques y cada bloque debe de ser desarrollado con la idea de la seguridad en mente, ya que un algoritmo criptográfico, no importando que tan bien diseñado este, no va a funcionar si su entorno no es seguro. Este par de autores, también defienden la idea de que todo sistema criptográfico debe de ser un sistema sencillo, ya que entre más complejo sea un sistema, más posibles fallos en la seguridad puede contener, y cada uno de estos posibles fallos podrá ser utilizado por un atacante, más aún, si a la ecuación le sumamos los *usuarios finales*, un sistema complejo traerá aún más hoyos en la seguridad del mismo.

El campo de la criptografía, es un campo verdaderamente extenso, por lo que es difícil poder abarcar todo el conocimiento existente, debido a esto, este trabajo de tesis esta enfocado solamente a buscar protección de sistemas criptográficos que utilicen la exponenciación modular como núcleo de los mismos y que sean utilizados en dispositivos pequeños, dispositivos que un atacante puede fácilmente obtener, manejar y estudiar para extraer información sensible a partir de ellos.

Los algoritmos presentados en este trabajo, no están enfocados a grandes computadoras con capacidades enormes de almacenamiento y procesamiento, computadoras que difícilmente podrían ser adquiridas por un atacante y tener libertad total de examinarlas y estudiarlas, por el contrario, dichos algoritmos están enfocado a dispositivos pequeños, con capacidades mínimas de almacenamiento y procesamiento. Por lo tanto, los algoritmos aquí presentados deben mantener la sencillez necesaria para cumplir con las ideas de

Ferguson y Schneier. y para no exceder las capacidades de los dispositivos en donde sean implementados.

1.1. Planteamiento.

Originalmente, la seguridad de un sistema criptográfico dependía del concepto matemático sobre el cual estaba basado, aunado a esto, también dependía de la dificultad de encontrar los valores secretos por medio de esfuerzos computacionales. Si sumábamos el concepto matemático fuerte más su dificultad de ser resuelto computacionalmente, en teoría, teníamos un sistema altamente seguro. Pero, ¿Que sucede cuando el criptosistema pasa del papel al mundo real?, ¿Que sucede cuando el sistema se implementa en dispositivos electrónicos reales, que están compuestos de elementos que tienen un comportamiento especifico?, ¿Que pasa cuando la forma del programa informático que realiza la encriptación o desencriptación de datos es, por decirlo de alguna forma, irregular? Desafortunadamente, estas interrogantes fueron contestadas por *Kocher* en 1996, el demostró que cuando estos sistemas son implementados en dispositivos reales, los tiempos de ejecución y el consumo de potencia que utiliza el dispositivo cuando ejecuta un algoritmo criptográfico, pueden ser fácilmente observados por medio de un equipo de medición, tal como un osciloscopio, y a partir de estas mediciones un atacante puede obtener la clave secreta del criptosistema, todo esto sin realizar un cálculo matemático complejo o utilizar un programa computacional que tarde años en poder encontrar la clave secreta.

Debido al problema mencionado, la finalidad de este tema de tesis es encontrar formas seguras de implementar algoritmos criptográficos, basados en la exponenciación modular, en sistemas reales, formas que permitan que un dato pueda ser calculado correctamente sin dar información del mismo a un atacante, y ese es el punto de partida de este trabajo.

1.2. Pasos realizados.

En esta sección, solo se hablara en forma general de los pasos a seguir para llegar a los resultados obtenidos en este trabajo de tesis. No se profundizara mucho en explicaciones, debido a que la metodología general realizada, es la metodología del método científico, es decir, observación, generación de una idea o teoría, experimentación y comprobación. A continuación se enumeran los pasos seguidos a través de todo el proceso de investigación:

1. El primer paso de este proceso, fue la recolección de información relacionada al tema de estudio, desde el inicio se tenía claro el campo al que se enfocaba la investigación, en este caso, hacía la exponenciación modular, el como había sido atacada, y las medidas que se habían utilizado para evitar dichos ataques.

2. Teniendo recolectada una lista de trabajos relacionados al tema, se realizó la búsqueda de problemas que permanecieran abiertos, encontrando algunos muy interesantes como el algoritmo 10, el ataque $N-1$ (explicado más adelante) y ataques basados

en el símbolo de Jacobi. Es importante hacer notar, que los temas mencionados atrajeron la atención del autor desde que aparecieron en la bibliografía recolectada, obviamente, este interés permitió que se siguiera trabajando sobre dichos tópicos.

3. Una parte importante en el entendimiento y estudio de la bibliografía consultada, fue la cuestión práctica. Para este fin, se programaron los algoritmos de exponenciación modular publicados previamente. A partir de tales programas computacionales, se empezó a estudiar el comportamiento de cada uno de ellos.

4. Teniendo las primeras nociones acerca del comportamiento computacional de dichos algoritmos, se inició con el estudio y programación de smart cards. Para este propósito, se adquirió un lector de smart cards de la marca *Zeit control* compatible con Windows, así como también, se adquirieron smart cards programables de la misma marca que el lector de tarjetas, modelos ZC3.1 y ZC3.3 con capacidades de memoria EEPROM de 2KB y 8KB respectivamente, y ambas con una memoria RAM de 256 bytes. Con las tarjetas ya programadas, se realizaron ataques básicos por medio de la medición de potencia del lector de tarjetas con la ayuda de un osciloscopio digital *PicoScope* de la marca *Pico Technology*, modelo 2203, con frecuencia de muestreo de 40MSPS y un ancho de banda de 5MHz.

5. Después de la recolección de información, de la búsqueda de problemas abiertos, de la programación y el estudio del comportamiento computacional y matemático de los distintos algoritmos de exponenciación analizados, ya se tenía una idea, si no clara al menos si visible sobre el como buscar la solución a los problemas abiertos encontrados.

   Al analizar los problemas que queríamos resolver, la primer situación fue el visualizar cual era el problema realmente, es decir, el problema general es que se tenía alguna vulnerabilidad en el algoritmo, pero el problema real era el saber en base a que se tenía esa vulnerabilidad. El camino a tomar, consistió en observar cual era el problema real y buscar la solución en base a conceptos contrarios al esquema de ataque. Como ejemplo, un problema manejado, consistía en la existencia de dos tipos de exponenciaciones intermedias dentro del algoritmo, por lo que la idea contraria era que solo se realizara un tipo de exponenciación a través de la ejecución del algoritmo.

6. Parecería que después del punto anterior la solución se había obtenido, sin embargo, la realidad es diferente. Después de tener las ideas acerca del como evitar el problema, el siguiente paso consistió en buscar la forma de implementar esas ideas en los algoritmos reales, pero no solamente era implementar las ideas, sino que también era necesario que esas implementaciones fueran eficientes en forma temporal, espacial y monetaria.

   En este punto, comienza una etapa de prueba y error, ya que se realizaron distintas formas para implementar la idea en una manera eficiente. A modo de ejemplificación, una de esas implementaciones, consistía en el defasamieno de una operación dentro

del algoritmo, es decir, una determinada operación se realizaba en la iteración $i-1$ en vez de realizarse en la iteración i que es en donde originalmente se ejecutaba.

Obviamente, no todas las implementaciones cumplían con lo requerido, es decir, seguridad y eficiencia. Se realizaron pruebas sobre varias implementaciones, pero aunque en algunos casos se evitaba la vulnerabilidad contra la cual se estaba desarrollando la contramedida, esa misma "solución" colocaba nuevas vulnerabilidades, no solo eso, en distintas ocasiones también aumentaba el número de registros de los algoritmos, el número de operaciones y por consiguiente el tiempo de ejecución.

7. Al obtener las versiones que se consideraban seguras, se sometían a diferentes esquemas de ataques simulados, con la intención de observar si presentaba alguna vulnerabilidad conocida. No solamente se utilizaron simulaciones computacionales para detectar los ataques, también se implementaron posibles ataques sobre papel para buscar vulnerabilidades no conocidas.

   Las simulaciones de ataques, se realizaron por medio de programas computacionales. Estos simuladores, eran generadores de errores aleatorios sobre distintos puntos del algoritmo, en el caso de la simulación de fault attacks, y graficadores de valores en el caso de side channel attacks.

8. Solo después de la comprobación de seguridad y eficiencia, se consideraba que un algoritmo estaba listo para funcionar de forma práctica.

1.3. Visión sobre los sistemas de seguridad.

Un sistema de seguridad, debe de estar protegido contra el mayor número posibles de ataques, y cada que se tenga el conocimiento de un nuevo tipo de ataque, es necesario erradicarlo completamente. Esta sería una visión ideal acerca de cualquier tema relacionado con la seguridad, sin embargo, en la vida cotidiana existen una gran cantidad de intereses comerciales, políticos, bélicos, etc. que no siempre permiten que esta característica se cumpla. En esta sección, se hablara en forma sintetizada de las distintas visiones que se tienen con respecto a la implementación y protección de sistemas de seguridad en el mundo real, se presentaran algunos ataques que han sido realizados de forma práctica y la reacción de los distintos puntos de vista ante estas circunstancias.

1.3.1. Desarrollo público y privado.

En forma general, todo intercambio de datos por medio de canales de información utiliza sistemas de seguridad y/o criptográficos, sin embargo, no todos los algoritmos e ideas son conocidos públicamente. Los sistemas de seguridad, son básicamente vistos desde dos enfoques completamente diferentes: el público y el privado. La investigación pública relacionada a criptosistemas, esta realizada principalmente en el mundo académico, por otro lado, la investigación privada esta concentrada principalmente en los mundos políticos y militares.

Un ejemplo muy claro del choque entre ambas tendencias, es el nacimiento de la criptografía de clave pública, este tipo de criptografía fue dado a conocer por los investigadores del *Instituto Tecnológico de Massachusetts* (MIT, por sus siglas en inglés): *Rivest, Shamir* y *Adleman*, con su sistema RSA en 1977. Sin embargo, después de la publicación de este sistema, los criptógrafos ingleses *Ellis, Cocks* y *Williamson* del *Grupo de Seguridad de Comunicaciones Electrónicas del Gobierno del Reino Unido / Dirección General de Comunicaciones Gubernamentales* (CESG/GCHQ, por sus siglas en inglés) afirmaron que ellos habían descubierto secretamente la encriptación de clave pública años antes que los investigadores estadounidenses. Al haber realizado la investigación de forma secreta y al no haber dado a conocer los resultados a través de un medio accesible para cualquier persona, el reconocimiento por haber desarrollado la criptografía de clave pública es dado a los tres investigadores del MIT, y también es gracias a la critica y análisis de investigadores de todas partes del mundo, que estos sistemas han llegado a evolucionar para estar preparados contra una gran cantidad de posibles vulnerabilidades.

No solamente en cuestiones gubernamentales se da este tipo de situaciones, muchas empresas comerciales también mantienen sus propios sistemas de seguridad resguardados en forma secreta, lo que puede ocasionar que dichos sistemas no tengan el nivel de seguridad necesario que los consumidores requieran, debido principalmente a que dichos sistemas no son estudiados ni evaluados en una forma abierta, por lo tanto, un posible ataque permanecerá inobservable por los desarrolladores, pero un atacante puede percatarse de él. Un ejemplo de este tipo de situaciones, lo podemos observar con la empresa *Microchip Systems* y su algoritmo *Keeloq*. Esta empresa, se encarga de la manufactura de sistemas de seguridad ampliamente utilizados en la protección de carros y acceso a compuertas por medio de mandos a distancia. Para ofrecer seguridad a los consumidores, se utiliza el algoritmo Keeloq, que es un algoritmo de encriptación ligero por bloques que utiliza claves pequeñas de 32 bits y 64 bits. El problema aquí, fue que este algoritmo permaneció oculto, sin embargo, en 2006 los datos técnicos de este sistema fueron robados de la empresa mencionada, estos datos aparecieron en una web rusa y de ahi cualquier persona podía obtenerlos y estudiarlos[1]. Afortunadamente, un equipo de científicos[2] de 3 distintos grupos de investigación (*computer science department of the Technion*, de Israel; *research group COSIC of the Katholieke Universiteit Leuven*, de Bélgica; y *math department of the Hebrew University*, de Israel) se encargaron de analizarlo, y en pocos días ya tenían el esquema de un ataque básico contra el algoritmo, antes de publicar sus resultados en [4], los científicos hablaron con la compañía manufacturera para darle a conocer las debilidades del sistema.

Este caso es un caso que puede considerarse afortunado, ya que un grupo de investigadores dio con el algoritmo y trataron de brindar una mejoría a la seguridad del mismo, antes de que este sistema fuera atacado por enemigos reales.

Los *Side channel attacks*, que son los ataques que principalmente competen a este trabajo,

[1] http://www.wired.com/threatlevel/2007/08/researchers-cra/

[2] http://redtape.nbcnews.com/$_{n}ews$/2007/08/28/6345961 − *researchers* − *say* − *theyve* − *hacked* − *car* − *door* − *locks*?*lite*

también han estado inmersos en las dos vertientes. Según *Wright* [5], en las décadas de los 50's y 60's, el MI5, *servicio de seguridad del Reino Unido*, ya utilizaba este tipo de ataques en los campos políticos y militares, obviamente mantenidos en estricto secreto. Entre este tipo de ataques se menciona el realizado en la operación ENGULF, que medía el sonido de los rotores de una maquina cifradora *Hagelin* que era usada en la embajada egipcia, todo esto realizado durante la *crisis de Suez*. También se menciona la operación STOCKADE, que explotaba las emanaciones electromagnéticas de la entrada de un teleimpresor en la salida de una maquina cifradora usada en la embajada francesa.

Es importante el hacer mención, que a pesar de que este tipo de ataques ya eran utilizados de forma práctica desde los años 50's, no fue hasta 1985 cuando ataques del tipo side channel, fueron conocidos en forma pública, esto debido a que *van Eck* [6] realizó un experimento en donde se podían obtener las emanaciones electromagnéticas de un monitor de computadora a distancia, y a partir de dichas emanaciones era posible reconstruir la imagen desplegada en el monitor atacado, en un segundo monitor observado por el atacante. Pero fue hasta 1996, donde este tipo de ataques ya fue utilizado para vulnerar las implementaciones de algoritmos criptográficos en dispositivos electrónicos.

En contraparte a las operaciones secretas ENGULF y STOCKADE, tenemos un side channel attack realizado de forma práctica y pública llevado a cabo en 2008. *Eisenbarth* y otros autores [7], presentaron un ataque del tipo side channel realizado de forma práctica. Estos autores, mediante la medición de consumos de potencia y emanaciones electromagnéticas, lograron encontrar la clave secreta del dispositivo y la clave secreta del fabricante almacenadas ,respectivamente, en el transmisor y en el receptor pertenecientes a sistemas comerciales de protección vehicular y manipulación de entradas a distancia, sistemas que utilizan el algoritmo Keeloq. Este es un trabajo de interés, debido a que como ya se menciono, rompieron la seguridad de sistemas que se venden comercialmente, y en palabras de los mismos autores: "Los ataques físicos, no deben de ser considerados como relevantes solo en la industria de las smart cards o como meros ejercicios académicos".

1.3.2. Ocultación de ataques reales.

No solamente los algoritmos se mantienen en secreto en cuestiones de seguridad, en la práctica, muchos ataques realizados contra diversas instituciones comerciales y no comerciales, son ocultados por las mismas instituciones, debido principalmente a que estos ataques pueden dar una percepción popular de ser una institución débil e insegura, lo que puede provocar perdidas económicas.

Para ejemplificar un poco la situación mencionada en el párrafo anterior. En 2008, un grupo de estudiantes de la *Radboud University Nijmegen* dirigidos por el profesor *B. Jacobs*, mostraron que el chip RFID (*Radio Frequency IDentification*) *Mifare clásico* (el cual es utilizado en una gran cantidad de aplicaciones, como acceso a edificios o acceso a sistemas de transporte), muestra vulnerabilidades. En el ataque que ellos presentaron, se intercepta una señal de la comunicación entre una tarjeta RFID y el lector de tarjetas,

y a partir de esa señal es posible obtener las claves criptográficas. Lo interesante de este estudio, es que se llevo a cabo de forma exitosa en sistemas reales de transporte, tal como el metro de Londres donde se utiliza la tarjeta llamada *Oyster*. Este grupo de estudiantes, dieron a conocer sus resultados tanto al gobierno Holandés como a la empresa manufacturera de las tarjetas: *NXP*, con la intención de que se realizaran los cambios necesarios a la seguridad de este sistema, sin embargo, NXP decidió poner una demanda judicial contra el grupo de investigadores, así como contra la institución universitaria, para que estos no publicaran los resultados obtenidos[3]. Afortunadamente, el sistema judicial fallo a favor de la Universidad y sus estudiantes/investigadores, permitiendo que dicha información pudiera ser publicada[4] en [8].

En 2010, salio a la luz la tesis de *Omar Choudary* [9], estudiante de *Cambridge*. En dicha tesis, el autor construye lo que el llama *smart card detective*. Este dispositivo, se creo originalmente para servir como un detector de claves PIN incorrectas y permitir transacciones más seguras. Sin embargo, es un dispositivo que también permite realizar un ataque de forma práctica, donde un atacante puede colocarse como "intermediario" entre una tarjeta comercial y el lector de tarjetas, y de esta forma engañar a la terminal lectora para que crea que cualquier PIN introducido es correcto. Según el autor de dicho trabajo, se realizaron pruebas en tiendas aleatorias de Cambridge para demostrar su efectividad en el mundo real. Este trabajo, al igual que el mencionado anteriormente, pudo haber sido un buen paso para mejorar los protocolos de autenticación en sistemas comerciales, sin embargo, la *United Kingdom Cards Association* (UKCA), que representa a los más grandes bancos de aquel país, solicito a la universidad de Cambridge que sacaran del dominio público los datos obtenidos por Omar. Cambridge, se nego a retirar dicha publicación, enviando una carta a UKCA, haciéndoles ver que por encima de los intereses monetarios, se encuentra el conocimiento (la carta enviada por *Ross Anderson* a la UKCA se puede consultar en http://www.cl.cam.ac.uk/∼rja14/Papers/ukca.pdf).

En los dos ejemplos anteriores, los datos obtenidos a partir de una investigación han salido a la luz pública, aun cuando grandes instituciones se han opuesto a ello, sin embargo, no todos los conflictos de este tipo tienen el mismo resultado y es aquí donde se oscurece la obtención de información relacionada a ataques implementados realmente, ya que las empresas/instituciones bloquean este tipo de datos.

1.3.3. TEMPEST.

Un caso que tiene mención aparte, es el llamado TEMPEST (hoy en día también llamado EMSEC, *Emission Security*), mucho se ha especulado de que la palabra TEMPEST es un acrónimo de *Telecommunications Electronics Material Protected from Emanating Spurious Transmissions*, o un acrónimo para *Transient Electromagnetic Pulse Emanation Standard* entre muchos otros, sin embargo, oficialmente solo se toma como un palabra

[3]http://www.sos.cs.ru.nl/applications/rfid/main.html
[4]http://www.sos.cs.ru.nl/applications/rfid/pressrelease-courtdecision.en.html

clave usada por los Estados Unidos sin acrónimo existente.

Desde la década de los 50's, el gobierno estadounidense se dio cuenta que las emanaciones electromagnéticas de equipos computacionales podían ser capturadas, y a partir de las señales capturadas, se podía reconstruir información de inteligencia, por este motivo se dio importancia a la investigación relacionada a la emisión y recepción de estas emanaciones, a este campo de investigación se le denomino TEMPEST. Todo lo relacionado a TEMPEST, se mantuvo en secreto por el gobierno norteamericano. Hoy en día, la NSA (*National Security Agency*) controla el uso de tecnología TEMPEST, tanto para equipos de monitoreo, como para que los equipos electrónicos no puedan ser monitoreados, o al menos reducir el riesgo de monitoreo. El uso de dispositivos de monitoreo de señales electromagnéticas, esta prohibido por el gobierno de los Estados Unidos, y solamente pueden diseñar este tipo de equipos, laboratorios autorizados, y el único que puede comprar estos aparatos es el gobierno estadounidense.

Este tipo de investigaciones, han sido de gran impacto dentro de ámbitos gubernamentales y militares, esto se puede comprobar con el simple hecho de que la NSA mantiene estándares de diseño para dispositivos electrónicos, basados en la emanación de señales electromagnéticas, buscando que tales dispositivos no emitan una mayor cantidad de señales que las permitidas por la ley. La NSA, ha clasificado los dispositivos protegidos contra TEMPEST[5], en tres categorías: categoría 1, extremadamente segura y solo disponible para el gobierno de los Estados Unidos y entes autorizados; categoría 2, menos segura que la categoría 1 pero requiere aprobación gubernamental para su uso; y la categoría 3, que es la seguridad para uso comercial.

1.3.4. Visualización de los side channel attacks en el mundo real.

Los side channel attacks, como muchos tópicos de seguridad, pertenecen a un tema que puede parecer un objeto meramente académico, sin embargo, los side channel attacks, son tomados muy en serio en el mundo real, ya que se han realizado laboratorios dentro de grandes compañías, tal como SIEMENS, para evitar este tipo de ataques en sus equipos. No solamente se han realizado laboratorios como una sección dentro de una compañía, también existen compañías, tal como la *Cryptography Research Inc.*, enteramente dedicadas a la solución de distintos problemas de seguridad, ofreciendo evaluaciones de seguridad y servicios especializados.

Cryptography Research Inc., realiza evaluaciones de seguridad sobre chips y otros dispositivos criptográficos. Entrando en el tema de esta tesis, esta compañía realiza pruebas de seguridad contra side channel attacks por medio de su *Cryptography Research DPA Workstation*. Según palabras de la empresa "DPA Workstation es la más potente y flexible plataforma en el mundo para el análisis de ataques del tipo side channel", es decir, a través de esta terminal se realizan evaluaciones de seguridad de distintos dispositivos

[5]http://www.nsa.gov/applications/ia/tempest/index.cfm

electrónicos, para dar una mayor certidumbre de que dichos dispositivos son resistentes contra side channel attacks. Este es un buen ejemplo para mostrar la importancia que los side channel attacks pueden tener en el mundo real.

1.3.5. Algunos comentarios.

Todo lo expuesto en esta sección, es mencionado para mostrar de forma general, el como los sistemas de seguridad son tratados desde distintos puntos de vista. Lamentablemente, existe una gran tendencia oscurantista con respecto a este tema, básicamente, nadie quiere dar información acerca de sus sistemas de seguridad. Esta idea podría ser aparentemente lógica, puesto que si queremos proteger datos, lo más lógico es que no se de a conocer la forma en la que los protegemos, sin embargo, el ocultar todo este tipo de información conlleva a sistemas no evolucionados y no mejorados. Cuando un sistema de seguridad es mostrado en forma pública, si bien es cierto que posibles atacantes pueden tener acceso a él, también es cierto que investigadores alrededor del mundo van a observarlo, analizarlo, criticarlo, y lo más importante, van a mejorarlo, es decir, la comunidad relacionada a este tópico va a encontrar vulnerabilidades en dicho sistema pero también se van a dar soluciones a tales vulnerabilidades. Este esquema público, permite que los sistemas de seguridad vayan mejorando y siendo un poco más seguros cada nuevo día, situación que no puede llevarse a cabo bajo un estándar privado.

Lo mencionado en el párrafo anterior, podría ser "justificable" bajo el supuesto de que la compañía tenga miedo a que su sistema sea vulnerado al ser público, y que esta situación pueda afectar a sus consumidores, sin embargo, los ejemplos expuestos anteriormente nos muestran intereses menos altruistas por parte de varias instituciones, es decir, hay una preocupación mayor por mantener un sistema de ganancias, sin la intención de mejorar los productos ofrecidos, ya que cuando investigadores (más no atacantes reales) han demostrado las vulnerabilidades que los productos comerciales presentan, las compañías no han intentado mejorar sus productos, sino más bien, han intentado eliminar la información relacionada a dichas vulnerabilidades, lo cual también evita la evolución y mejoramiento de los sistemas criptográficos privados.

Como ya se ha mencionado en esta sección, los side channel attacks se han presentado en el mundo real, y lo más importante, tienen un verdadero campo de aplicación en sistemas comerciales. Debido a esta situación, en este trabajo de tesis se dara un enfoque hacia el diseño de algoritmos de exponenciación seguros contra distintos ataques del tipo side channel y que puedan ser implementados en sistemas reales, todo esto sin necesidad de cambiar las características del hardware para su uso.

1.4. Algunas herramientas.

Para un sistema criptográfico seguro, es necesario el uso de números grandes, es decir, números que puedan tener 1024 o 2048 bits de longitud en su forma binaria, lamentablemente los lenguajes de programación C y C++ no cuentan con un tipo de datos que soporte esta clase de valores, por lo que se ha utilizado la librería GMP de distribución libre. GMP, es una librería específicamente desarrollada para trabajar con sistemas criptográficos y con conceptos de teoría de números, esta librería permite el uso de datos tan grandes como la memoria de la computadora permita, lo que la hace adecuada para el tema en cuestión.

GMP esta desarrollada para trabajar bajo lenguajes C y C++, lo que ha sido de gran utilidad, ya que se han programado todos los algoritmos en C, con la ayuda del editor *Dev-cpp*, también de licencia libre.

Para estudiar el funcionamiento de los ataques descritos a lo largo de esta tesis, se desarrollaron simuladores de ataques, que básicamente consistían en generadores de elementos aleatorios colocados sobre posiciones específicas dentro de la ejecución de los algoritmos probados.

Toda la programación de los algoritmos, y la exposición de los mismos a ataques simulados, se llevo a cabo mediante una computadora con un procesador *Core(TM)2 Duo* T5550 a 1.3 GHz, dos Giga Bytes de memoria RAM y bajo un sistema XP, con *service pack* 2.

Como mención especial, voy a hablar acerca de la cuenta de acceso a BiDi UNAM, ya que esta fue de gran utilidad para la búsqueda y consulta de textos especializados, tal como libros, tesis y artículos. Menciono esta herramienta, debido a que en muchas ocasiones los alumnos en los distintos niveles de educación, desconocemos que existen este tipo de apoyos por parte de la UNAM, por lo que yo recomiendo a un futuro lector de este trabajo, la solicitud y uso de estas cuentas que ayudarán en más de un problema académico.

1.5. Artículos publicados.

Durante este periodo de trabajo, se han podido realizar publicaciones en revistas internacionales, las publicaciones realizadas se encuentran numeradas a continuación:

1. Tinoco Varela D. Attack against an efficient exponentiation algorithm used in cryptosystems. *Proceedings of the 5th International Conference MSAST 2011 of IMBIC*, pp. 196-204. 2011-2012.

2. Tinoco Varela D. Blinded Montgomery powering ladder protected against the Jacobi symbol attack. *International Journal of Security*, Vol. 6, Issue 3, pp. 15-27. 2012.

3. Tinoco Varela D. How to avoid the N-1 attack without costly implementations. *International Journal of Network Security and Its Applications*, Vol. 4, No. 4, pp.109-122. 2012.

1.6. Organización de la Tesis.

En el capítulo 2, se explicarán a grandes rasgos los conceptos necesarios para el entendimiento del presente trabajo, estos conceptos son tanto matemáticos como técnicos.

En el capítulo 3, se presentarán los algoritmos de exponenciación modular básicos, y aquellos que son relevantes para el trabajo de esta tesis.

En el capítulo 4, se explicarán los distintos tipos de ataques físicos existentes y como se han utilizado contra los sistemas criptográficos, y específicamente, contra la exponenciación modular, así como también, se presentan versiones posteriores de los algoritmos de exponenciación para evitar este tipo de ataques. En cuanto a los ataques físicos, básicamente nos estaremos enfocando a ataques que miden el tiempo y el consumo de potencia de un dispositivo, así como de ataques que utilizan pequeñas cargas eléctricas para provocar un funcionamiento erróneo de dicho dispositivo. En este capítulo, también se presenta una propuesta de ataque contra el algoritmo de exponenciación modular de *Sun Da-zhi*, para determinar la cadena binaria del exponente.

En el capítulo 5, se presentará la contramedida propuesta para evitar ataques por medio del *símbolo de Jacobi* y ataques con valores de entrada igual a $N - 1$, el por que de este valor se explicará en dicho capítulo.

En el capítulo 6, se darán las conclusiones y los comentarios finales del trabajo presentado.

En el apéndice A, se dará una pequeña reseña de algunos ataques físicos diferentes al de medición de consumo de potencia, a su vez, también se resumirán algunas implementaciones que se han realizado en *hardware* para evitar dichos ataques. Las contramedidas en hardware no son parte de nuestro trabajo, sin embargo, es de gran interés el conocer un poco acerca de ellas, ya que están contenidas dentro de un sistema de seguridad completo.

En el apéndice B, se presentarán algunas técnicas empleadas para mejorar la velocidad de los algoritmos de exponenciación.

Como última nota, los capítulos 3 y 4 están sumamente ligados, es difícil hablar de una contramedida sin mencionar el ataque que quiere evitar, así mismo, es difícil hablar de un ataque sin saber el algoritmo al que fue dirigido, estos dos capítulos pudieron haber sido uno solo, que explicara en forma cronológica sobre el desarrollo de ataques y sus contramedidas, sin embargo, se realizaron dos capítulos distintos, con la finalidad de englobar conceptos similares por capítulo y esperando que de esta forma no exista una confusión entre los ataques y sus contramedidas.

Capítulo 2

Conceptos generales.

Este capítulo, esta enfocado a la presentación de los conceptos necesarios, tanto matemáticos como técnicos, para el desarrollo del temario a lo largo del presente trabajo. Los conceptos aquí presentados, son ampliamente conocidos y ya han sido definidos y desarrollados en una cantidad incontable de textos, por lo que estos conceptos son dados solamente a modo de referencia y no se profundizará demasiado en ellos.

A modo de aclaración, los tópicos expuestos en este capítulo no son los tópicos fundamentales del trabajo de tesis, cada uno de los conceptos fundamentales se ira tratando en capítulos sucesivos, y a cada uno de ellos si se les dará una mayor explicación.

2.1. Aritmética modular.

La aritmética modular, es de gran importancia e interés en campos como la criptografía, donde se trabajan números grandes y es necesario el poder realizar los cálculos de una forma rápida y simplificada. El aporte esencial de esta aritmética, es que podemos reducir un conjunto infinito de números a un conjunto finito, este conjunto finito se obtiene al remplazar cualquier entero t por su residuo, cuando t ha sido dividido entre un valor n, denominado *modulo*.

Cuando trabajamos sobre un modulo n, solamente estaremos trabajando con valores que van desde 0 hasta $n-1$, en lugar de trabajar con un número infinito de valores, esta reducción de valores es lo que nos permite obtener la rapidez mencionada. Cuando trabajamos con aritmética modular, se utiliza una nomenclatura llamada congruencia, definida por el símbolo $\equiv$.

2.1.1. Congruencias y sus propiedades básicas.

La notación para representar a las congruencias fue introducida por *C. F. Gauss* (1777-1855) en su trabajo *Disquisitiones Arithmeticae* que fue publicado en 1801.

Para la definición de este término, tenemos los valores enteros $a,\ b$ y $n \in \mathbb{N}$, tal que $n \neq 0$. Con los valores mencionados, se puede definir una congruencia $a \equiv b(mod\ n)$ (se leé a es congruente con b modulo n) en el sentido que $n|(a-b)$ y $a = b + kn$ para algún valor k, o dicho de otra forma, existe congruencia entre a y b, si el residuo de ambos valores es el mismo al ser divididos, independientemente, por n. Si $n \not|(a-b)$, significa que los valores a y b no tienen el mismo residuo cuando son divididos por n, por lo tanto no son congruentes $mod\ n$.

Para ejemplificar el concepto de congruencia, tenemos que los valores 13, 24 y 35 son congruentes mod 11, debido a que los tres valores tienen el mismo residuo al ser divididos por 11, es decir, $13 \equiv 24 \equiv 35 \equiv 2(mod\ 11)$, para la congruencia $35 \equiv 2(mod\ 11)$ tenemos que: $11|(35-2)$ ya que $11|(33)$, y se cumple que $a = b + kn$ pues $35 = 2 + 3 \cdot 11$ para un valor $k = 3$.

Todos los valores que sean congruentes modulo n, forman una *clase de equivalencia*. Dentro de un modulo n, existen exactamente n clases de equivalencia, y todo valor entero pertenece a una clase de equivalencia modulo n. Para ejemplificar este termino, tenemos que 13, 24, -9, -31, 2 pertenecen a la misma clase de equivalencia modulo 11 y 16, 38, -6, -28, 5 pertenecen a otra clase de equivalencia modulo 11.

2.1.2. Propiedades de las congruencias.

Las congruencias tienen las siguientes propiedades:

1. $a \equiv b\ (mod\ n)$, es siempre verdadera si $n = 1$.
2. Si $a \equiv b\ (mod\ n)$, implica que $b \equiv a\ (mod\ n)$.
3. Si $a \equiv b$ y $b \equiv c\ (mod\ n)$, entonces $a \equiv c\ (mod\ n)$. Para $a - b = kn,\ b - c = jn$ implica $a - c = (k + j)n$.
4. Si $a \equiv b$ y $c \equiv d\ (mod\ n)$ entonces $a + c \equiv b + d$ y $a - c \equiv b - d\ (mod\ n)$. Para $a - b = kn,\ c - d = jn$ implica $(a + c) - (b + d) = (k + j)n$.
5. Si $a \equiv b\ (mod\ n)$ entonces, para algun entero c, $ac \equiv bc\ (mod\ n)$. Para $a - b = kn$ implica $ac - bc = kcn$.
6. Si $a \equiv b$ y $c \equiv d\ (mod\ n)$, entonces $ac \equiv bd\ (mod\ n)$. Para $ac \equiv bc$ y $bc \equiv bd$.
7. $a \equiv b\ (mod\ n)$ si y solo si $ac \equiv bc\ (mod\ nc)$.

2.1.3. Inversos multiplicativo mod n.

Si tenemos dos valores enteros a y b tal que $gcd(a, b) = 1$, decimos que a y b son números primos relativos, es decir, no tienen factores primos comunes. Ahora bien, si tenemos que $gcd(a, n) = 1$, donde a es cualquier valor y n es el modulo dentro de una operación de

congruencia, significa que a tiene un inverso multiplicativo en el modulo n, es decir, existe un valor k tal que $a \cdot k \equiv 1 \ (mod\ n)$, o lo que es lo mismo $a \equiv k^{-1} \ (mod\ n)$.

Como ejemplo de ese concepto, si tenemos un modulo $n = 1765$, y tenemos un valor $a = 1131$, es necesario buscar un valor k que multiplicado por a nos de un resultado congruente con 1 *mod* n. El valor encontrado es $k = 2976$, ya que $(1131)(2976) \equiv 1\ mod\ 1765$.

Cuando el modulo es un valor primo p, todos los elementos dentro del modulo, exceptuando el 0, tienen inverso multiplicativo modulo p, ya que todos los elementos dentro del modulo son coprimos con p. Obviamente, esta característica no se cumple para elementos dentro de un modulo compuesto n.

2.1.4. Residuos cuadrados modulo n.

Si tenemos los valores enteros $n \geq 2$ y $a \in \mathbb{Z}$ con $gcd(a, n) = 1$, decimos que a es un *residuo cuadrado* modulo n si $a \equiv x^2 \ (mod\ n)$ para algún valor $x \in \mathbb{Z}$. Si a satisface $gcd(a, n) = 1$ pero no es un residuo cuadrado modulo n, este es llamado un *residuo no cuadrado*.

Para ejemplificar este concepto, vamos a utilizar un modulo $n = 17$, los residuos modulo 17 de $1^2, 2^2, ..., 16^2$ son 1, 4, 9, 16, 8, 2, 15, 13, 13, 15, 2, 8, 16, 9, 4, 1, por lo tanto, los residuos cuadrados modulo 17 son 1, 2, 4, 8, 9, 13, 15, 16. Para $n = 20$, los residuos cuadrados son 0, 1, 4, 5, 9, 16, y para $n = 7$ los residuos cuadrados son 1, 2, 4.

Una observación importante de mencionar, es que si $n = p$, donde p es un número primo, existe el mismo número de residuos cuadrados y de no cuadrados, tal es el caso de $n = 17$ donde hay 8 residuos cuadrados y 8 residuos no cuadrados, este es un comportamiento que se da de forma general para todo número primo p. Para determinar si un valor a es residuo cuadrado modulo p, se utiliza el *símbolo de Legendre*.

El símbolo de Legendre queda definido de la siguiente forma: para un número primo $p \geq 3$ y un entero a tenemos que

$$\left(\frac{a}{p}\right) = \begin{cases} 1, \ si\ a\ es\ un\ residuo\ cuadrado\ modulo\ p \\ -1, \ si\ a\ es\ un\ residuo\ no\ cuadrado\ modulo\ p \\ 0, \ si\ a\ es\ un\ multíplo\ de\ p \end{cases} \quad (2.1)$$

donde la representación $\left(\frac{a}{p}\right)$ es llamada el *símbolo de Legendre* de a y p.

El símbolo de Legendre, es fácilmente calculado por medio del *criterio de Euler*. El criterio de Euler nos dice que si tenemos un valor primo p, un valor a es residuo cuadrado si y solo si $a^{(p-1)/2} \equiv 1 \ (mod\ p)$, este resultado proviene a su vez del *pequeño teorema de*

Fermat que nos dice que la congruencia $a^{p-1} \equiv 1 \ (mod\ p)$ cumple si y solo si p es primo y $a \neq 0$, lo que nos lleva a $(a^2)^{(p-1)/2} = (a)^{(2)((p-1)/2)} = (a)^{(p-1)}$, por otro lado, es claro que $a^{(p-1)/2} \equiv 0 \ (mod\ p)$ si a es un múltiplo de p.

El símbolo de Legendre satisface algunas reglas, que ayudan a realizar el cálculo rápidamente. Estas reglas son enumeradas a continuación:

1. $\left(\frac{a \cdot b}{p}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{b}{p}\right)$, para todo $a,\ b \in \mathbb{Z}$.
2. $\left(\frac{a \cdot b^2}{p}\right) = \left(\frac{a}{p}\right)$, para $a\ \in \mathbb{Z}$, donde $p \not| b$.
3. $\left(\frac{a+cp}{p}\right) = \left(\frac{a}{p}\right)$, para todos los enteros a y c.
4. $\left(\frac{-1}{p}\right) = 1$, si y solo si $p \equiv 1 \ (mod\ 4)$.

2.1.5. El símbolo de Jacobi.

Una generalización del símbolo de Legendre es el llamado símbolo de Jacobi (SJ), el SJ queda definido de la siguiente forma:

Si tenemos un entero impar $n \geq 3$ con una descomposición de factores primos tal que $n = p_1 \cdots p_r$, y tenemos un entero a, entonces podemos usar la expresión

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right) \cdots \left(\frac{a}{p_r}\right), \tag{2.2}$$

donde $\left(\frac{a}{n}\right)$ es llamado el símbolo de Jacobi.

Es fácil ver, que los símbolos de Legendre y Jacobi son representados de la misma forma, esto debido a que si el valor n es un número primo, el SJ es el mismo que el símbolo de Legendre y la representación es irrelevante.

En el caso del SJ, es necesario hacer notar que el resultado de (a/n), puede ser algunas veces igual a 1 y no necesariamente significa que a sea residuo cuadrado de n, tal es el caso del siguente ejemplo $\left(\frac{18}{25}\right) = 1$, done a pesar de que el SJ de 18 con respecto a 25 es igual a 1, 18 no es un residuo cuadrado de 25. Otros casos donde a pesar de que tienen un SJ igual a 1, no son residuos cuadrados son: $\left(\frac{5}{21}\right) = 1$, $\left(\frac{33}{35}\right) = 1$, $\left(\frac{68}{6541}\right) = 1$, $\left(\frac{238}{7615}\right) = 1$, $\left(\frac{1007}{23121}\right) = 1$. Por otro lado, si el SJ de (a/n) es igual a -1, se tiene la certeza de que a es un residuo no cuadrado de n. Si $\left(\frac{a}{n}\right) = 0$, uno de los p_i's divide a a, por lo tanto a y n tienen un factor común.

Así como el símbolo de Legendre, el SJ satisface algunas características básicas:

1. $\left(\frac{a \cdot b}{n}\right) = \left(\frac{a}{n}\right) \cdot \left(\frac{b}{n}\right)$;

2. $\left(\frac{a\cdot b^2}{n}\right) = \left(\frac{a}{n}\right)$, si $gcd(b,n) = 1$;

3. $\left(\frac{a}{n\cdot m}\right) = \left(\frac{a}{n}\right)\cdot\left(\frac{a}{m}\right)$;

4. $\left(\frac{a}{n\cdot m^2}\right) = \left(\frac{a}{n}\right)$, si $gcd(a,m) = 1$;

5. $\left(\frac{a+cn}{n}\right) = \left(\frac{a}{n}\right)$, para todos los enteros c;

6. $\left(\frac{a^{2k}\cdot a}{n}\right) = \left(\frac{a}{n}\right)$, y $\left(\frac{a^{2k+1}\cdot a}{n}\right) = \left(\frac{2}{n}\right)\cdot\left(\frac{a}{n}\right)$, para $k \geq 1$;

7. $\left(\frac{-1}{n}\right) = (-1)^{(n-1)/2}$;

8. $\left(\frac{0}{n}\right) = 0$ y $\left(\frac{1}{n}\right) = 1$.

Aparentemente, el SJ puede ser una tarea difícil de calcular, debido a que n es un valor compuesto y la descomposición de n en sus factores primos sería necesaria, sin embargo, en base a la generalización de la *ley de reciprocidad cuadrada*, en donde, si $a, n \geq 3$ son enteros impares, tenemos

$$\left(\frac{a}{n}\right) = \begin{cases} \left(\frac{n}{a}\right) \ si\ n \equiv 1\ o\ a \equiv 1\ (mod\ 4), \\ -\left(\frac{n}{a}\right) \ si\ n \equiv 3\ y\ a \equiv 3\ (mod\ 4), \end{cases} \quad (2.3)$$

y en base a

$$\left(\frac{2}{n}\right) = \begin{cases} 1,\ si\ n \equiv 1\ o\ n \equiv 7\ (mod\ 8) \\ -1,\ si\ n \equiv 3\ o\ n \equiv 5\ (mod\ 8) \end{cases}$$

donde $n \geq 3$ es un entero impar, se puede realizar el cálculo del SJ sin realizar pruebas de primalidad o de factorización de enteros, por lo que de esta forma es fácilmente computable, y esto lo convierte en una herramienta práctica. Las propiedades para calcular el SJ de una forma eficiente son enumeradas a continuación:

1. Si a no esta en el intervalo $\{1, ..., n-1\}$, el resultado es $\left(\frac{a\ mod\ n}{n}\right)$.

2. Si $a = 0$, el resultado es 0.

3. si $a = 1$, el resultado es 1.

4. Si $4|a$, el resultado es $\left(\frac{a/4}{n}\right)$.

5. Si $2|a$, el resultado es $\left(\frac{a/2}{n}\right)$ si $n\ mod\ 8 \in \{1,7\}$ y $-\left(\frac{a/2}{n}\right)$ si $n\ mod\ 8 \in \{3,5\}$.

6. Si $a > 1$ y $a \equiv 1$ o $n \equiv 1\ (mod\ 4)$, el resultado es $\left(\frac{n\ mod\ a}{a}\right)$.

7. Si $a \equiv 3$ y $n \equiv 3\ (mod\ 4)$, el resultado es $-\left(\frac{n\ mod\ a}{a}\right)$.

Para ejemplificar el uso de las propiedades mencionadas anteriormente, vamos a calcular el SJ de $\left(\frac{1375}{45771}\right)$:

$$\left(\frac{1375}{45771}\right) \overset{(7)}{=} - \left(\frac{396}{1375}\right) \overset{(4)}{=} - \left(\frac{99}{1375}\right) \overset{(7)}{=} (-)(-)\left(\frac{88}{99}\right) \overset{(4)}{=} \left(\frac{22}{99}\right) \overset{(5)}{=} - \left(\frac{11}{99}\right) \overset{(7)}{=} (-)(-)\left(\frac{0}{11}\right) \overset{(2)}{=} 0$$

En este ejemplo vemos que el resultado es 0, esto es debido a que ambos valores tienen el factor común 11. Vamos a realizar otro ejemplo, en este caso vamos a calcular el SJ de $\left(\frac{6143}{73511}\right)$:

$$\left(\frac{6143}{73511}\right) \overset{(7)}{=} - \left(\frac{5938}{6143}\right) \overset{(5)}{=} - \left(\frac{2969}{6143}\right) \overset{(6)}{=} - \left(\frac{205}{2969}\right) \overset{(6)}{=} - \left(\frac{99}{205}\right) \overset{(6)}{=} - \left(\frac{7}{99}\right) \overset{(7)}{=} (-)(-)\left(\frac{1}{7}\right) \overset{(3)}{=} 1$$

En este otro ejemplo, el resultado es igual a 1, lo que significa que 6143 puede ser un residuo cuadrado modulo 73511.

2.2. Sistemas criptográficos.

Los sistemas criptográficos, son aquellos sistemas desarrollados para proteger información por medio de algoritmos de encriptación de datos. Estos sistemas son de principal utilidad e interés en los campos económicos y militares, pero no por eso menos importantes en campos sociales y personales.

Existe una gran cantidad de sistemas criptográficos, sin embargo, en este trabajo solo mencionaremos algunos sistemas que utilizan la exponenciación modular como operación central, ya que esta operación es la que nos interesa.

2.2.1. Diffie Hellman.

Diffie y *Hellman* en [10], dieron las bases para lo que se conoce como criptosistemas de clave pública, sin embargo, ellos no desarrollaron el primer criptosistema de clave pública propiamente dicho, ya que su propuesta no buscaba encriptar mensajes como tal, sino más bien, ellos desarrollaron un esquema para la generación de claves secretas a partir de datos conocidos y un canal de comunicación inseguro. Este esquema se muestra en el algoritmo 1.

El sistema DH, esta basado en la dificultad de encontrar el valor x a partir del resultado $g^x \ mod \ p$, aún conociendo los valores p y g de la exponenciación. Este esquema permite que los valores p y g sean conocidos por cualquier persona, ya que esto no vulnerará la seguridad.

Si una atacante *Eve*, intercepta ambos mensajes g^x y g^y, esta no podrá obtener la clave privada k, debido a que lo más que ella puede obtener es el valor $g^x \cdot g^y = g^{x+y}$ que es diferente a la clave $k = g^{xy}$.

Algoritmo 1 Generación de claves en el esquema DH.

1: **Entradas**: Generador g y un modulo primo p.
2: **Salida**: Clave k para Bob y Alice.
3: Alice busca un valor aleatorio y seguro x.
4: Alice calcula y envía el valor g^x a Bob.
5: Bob busca un valor aleatorio y seguro y.
6: Bob calcula y envía el valor g^y a Alice.
7: Alice recibe el valor g^y de Bob y lo eleva a x, $(k = g^y)^x = g^{xy}$.
8: Bob recibe el valor g^x de Alice y lo eleva a y, $(k = g^x)^y = g^{xy}$.
9: Alice y Bob ahora tienen la misma clave secreta.

2.2.2. RSA.

Este criptosistema es, tal vez, el más conocido y utilizado hoy en día, es el primer criptosistema desarrollado para trabajar en el esquema de clave pública. Fue diseñado en 1977 por tres investigadores del MIT: *Rivest, Shamir* y *Addleman* [11]. Este sistema, basa su seguridad en la dificultad que existe de factorizar números compuestos grandes en factores primos, cuando estos factores son parecidos en términos de tamaño. Este algoritmo utiliza varios valores, algunos públicos y otros privados, para realizar la encriptación y desencriptación de forma segura. El como se generan estos valores, puede verse en el algoritmo 2.

Algoritmo 2 Generación de valores para RSA.

1: Se buscan 2 números primos grandes p y q que sean similares en tamaño.
2: Se obtiene el modulo N del criptosistema por medio de $N = p \cdot q$.
3: Se calcula el valor $\phi(N)$ por medio de $\phi(N) = (p-1) \cdot (q-1)$.
4: Se escoge un valor aleatorio e, tal que $gcd(e, \phi(N)) = 1$.
5: Se calcula la clave privada d por medio de $d = e^{-1}\ mod\ \phi(N)$.
6: Los valores N y e son valores públicos.
7: Los valores p, q, $\phi(N)$ y d son valores privados.

RSA utiliza para su funcionamiento dos claves distintas, una privada d y una pública e. Con la clave pública e, cualquier persona puede encriptar un mensaje m y enviarlo al dueño de las claves, y solamente el destinatario es capaz de desencriptar el mensaje encriptado E con el uso de su clave privada d. La encriptación esta dada por $E = m^e\ mod\ N$ y la desencriptación esta dada por $m = E^d\ mod\ N$, donde N es el modulo del sistema obtenido a partir de dos números primos grandes p y q.

Como podemos ver en los criptosistemas RSA y DH, los valores privados en ambos sistemas son el exponente en una exponenciación modular, por lo que si un atacante determina el exponente en dicha operación, este habrá roto completamente la seguridad.

2.3. Exponenciación modular.

La definición más general de la exponenciación modular, es dada como una exponenciación que se ejecuta sobre un modulo. Esta operación, como ya se mencionó, es la operación central de varios criptosistemas, teniendo como exponente la clave secreta de los mismos.

Los algoritmos que calculan esta operación en sistemas computacionales, realizan el cálculo principalmente a partir de la representación binaria del exponente y no de la representación decimal, y a partir de los valores 0 y 1, es calculado el valor correcto de la exponenciación. Existen dos tipos de algoritmos que calculan la exponenciación, los algoritmos *Left to Right* y los algoritmos *Right to Left*, la diferencia de estos dos tipos de algoritmos radica en que unos inician el cálculo desde el bit más significativo de la cadena binaria y los otros lo inician desde el bit menos significativo.

Debido a la forma en que los algoritmos calculan la exponenciación, esta ha sido ampliamente atacada por medio de ataques físicos. Los ataques contra esta operación no están enfocados en obtener la clave por medio de una operación matemática, sino por el contrario, están enfocados en obtener la representación binaria del exponente por medio de la observación de señales emitidas por un dispositivo electrónico, mientras se calcula la exponenciación dentro de él. De esta forma, un atacante puede obtener información sensible del criptosistema.

Por el motivo ya mencionado, sabemos que es necesario el generar algoritmos que calculen la exponenciación modular, de forma que puedan brindar una mayor seguridad a los criptosistemas que la requieran como operación central, tal como: RSA, DH, ECC (*elliptic curve cryptosystem*).

Para el caso del ECC, no se utiliza la exponenciación modular como operación central, en este caso se utiliza la multiplicación escalar, sin embargo, ambas operaciones son análogas y los algoritmos que calculan ambas operaciones son, en términos generales, los mismos. La diferencia de ambos esquemas es meramente de representación, es decir, si un algoritmo de exponenciación usa una multiplicación y una elevación al cuadrado, un algoritmo de multiplicación escalar utilizará una suma y una multiplicación por 2, respectivamente.

2.4. Pequeños dispositivos y *Smart Cards*.

El caso de estudio del presente trabajo, se obtiene cuando sistemas criptográficos son implementados en dispositivos que pueden ser fácilmente obtenidos por un atacante, y este puede tener control total sobre dichos dispositivos. Estos dispositivos, generalmente son pequeños y constan de pocas capacidades de almacenamiento y procesamiento. Entre estos elementos se encuentran los Tokens, RFDI, y principalmente las *Smart cards* (SC).

Las Smart cards, son pequeños dispositivos electrónicos ampliamente utilizados en prácti-

camente todos los sectores de la sociedad, ya que estas se pueden encontrar en: tarjetas bancarias, sistemas de televisión de paga, identificadores personales, teléfonos celulares, sistemas de transporte, identificadores corporativos e incluso, en algunos países, pueden ser utilizados para monitorear el expediente médico de una persona.

Existen diferentes tipos de SC, ya que estas pueden ser solo de memoria, de procesamiento y criptográficas, estas últimas son, básicamente, tarjetas de procesamiento pero con funciones criptográficas en su núcleo.

Una SC promedio, puede contener un procesador de 8 bits, sin embargo, algunos nuevos diseños se comienzan a enfocar a procesadores de hasta 32 bits. La memoria de estos dispositivos esta, como ya se ha hecho referencia, muy limitada ya que en general la memoria ROM cuenta con 16 KB, la EEPROM cuenta con 16 bytes y la memoria RAM cuenta con 256 bytes de almacenamiento. Estos valores, son valores promedio, sin embargo se pueden encontrar tarjetas con distintos valores dependiendo del fabricante y precio en el mercado.

Como ya se mencionó, las SC pueden ser obtenidas fácilmente por un atacante y obtener datos secretos o sensibles a partir de su manipulación, por lo que es de suma importancia el generar SC con características que las hagan difíciles de vulnerar, tanto en el *software* como en el *hardware*.

Capítulo 3

Algoritmos de exponenciación modular.

En este capítulo, se presentan diferentes algoritmos de exponenciación modular y como estos han sido modificados para obtener un mayor grado de seguridad en sus implementaciones. Existe un gran número de algoritmos de este tipo, por lo que es muy difícil el poder hablar de todos y cada uno de ellos en este trabajo, debido a esta situación, solo se mencionarán los algoritmos que tienen mayor importancia sobre los resultados presentados en el capítulo 5.

Es un tanto complicado el hablar de las modificaciones realizadas a los algoritmos, sin tener el conocimiento del o los ataques que originaron dichas modificaciones, por lo tanto, en este capítulo se mencionaran los ataques, o razones, que generaron cada nueva versión de un algoritmo, pero es en el capítulo 4 en donde se explicarán en una forma más detallada cada uno de esos ataques.

3.1. Algoritmos square and multiply y square and multiply always.

El algoritmo clásico de exponenciación modular, es el algoritmo conocido como *square and multiply* (SaM), este algoritmo puede encontrarse en dos formas: *left-to-right* (LtoR) y *right-to-left* (RtoL), algoritmos 3 y 4 respectivamente.

Tanto el algoritmo 3 como el 4, calculan la operación de exponenciación de forma binaria, es decir, ambos utilizan la representación binaria del exponente para realizar el cálculo correcto de la exponenciación. Estos algoritmos, utilizan un ciclo *for* para ir escaneando cada uno de los bits de la cadena binaria y su valor correspondiente (0 o 1), es fácil observar que cuando el valor del bit es igual a 0, estos algoritmos realizan una multiplicación modular, sin embargo, cuando el valor del bit es igual a 1, estos algoritmos ejecutan dos multiplicaciones modulares (a partir de aquí vamos a tomar el cálculo de la exponenciación cuadrada modular como una multiplicación modular).

Algoritmo 3 Square and multiply, left-to-right.

1: **Entrada** m, d, N, con $d = (d_{n-1} \cdots d_0)_2$
2: **Salida** $S = m^d \ mod \ N$
3: $R[0] \leftarrow 1$
4: **for** $n-1$ to 0 **do**
5: $\quad R[0] \leftarrow R[0]^2 \ mod \ N$
6: $\quad$ **if** $d_i = 1$ **then**
7: $\quad\quad R[0] \leftarrow R[0] \cdot m \ mod \ N$
8: $\quad$ **end if**
9: **end for**
10: **Return** $R[0]$

Algoritmo 4 Square and multiply, right-to-left.

1: **Entrada** m, d, N, con $d = (d_{n-1} \cdots d_0)_2$
2: **Salida** $S = m^d \ mod \ N$
3: $R[0] \leftarrow 1; \ R[1] \leftarrow m$
4: **for** 0 to $n-1$ **do**
5: $\quad$ **if** $d_i = 1$ **then**
6: $\quad\quad R[0] \leftarrow R[0] \cdot R[1] \ mod \ N$
7: $\quad$ **end if**
8: $\quad R[1] \leftarrow R[1]^2 \ mod \ N$
9: **end for**
10: **Return** $R[0]$

La condicional *if* dentro del ciclo *for*, provoca un comportamiento irregular entre los tiempos de ejecución de las iteraciones, es decir, si el valor del bit que se este evaluando es igual a 1, el tiempo de ejecución será mayor que cuando el valor del bit sea igual a 0, aunado a esto, el consumo de potencia, cuando el algoritmo es implementado en un dispositivo electrónico, también será mayor cuando el valor del bit sea igual a 1 que cuando sea igual a 0 (las consecuencias de esta irregularidad serán explicadas en el capítulo 4). Kocher en 1996, se dio cuenta de este comportamiento, y en base a él, desarrollo los primeros *side channel attacks*.

Con la intención de evitar los side channel attacks, *Coron* en 1999 [12] desarrolló el algoritmo conocido como *square and multiply always* (SaMA), representado como algoritmo 5. El SaMA, es un algoritmo regular, lo que significa que realiza el mismo número de operaciones en cada iteración del algoritmo, sin tomar en cuenta el valor del bit que se este ejecutando. Este algoritmo, utiliza operaciones *dummy* para su funcionamiento. Una operación dummy, es una operación que no es necesaria para calcular el valor correcto de salida de un algoritmo.

Algoritmo 5 Square and multiply always, left-to-right.

1: **Entrada** m, d, N, con $d = (d_{n-1} \cdots d_0)_2$
2: **Salida** $S = m^d \ mod \ N$
3: $R[0] = 1$
4: **for** $n - 1$ to 0 **do**
5: $\quad R[0] = R[0]^2 \ mod \ N$
6: $\quad R[1] = R[0] \cdot m \ mod \ N$
7: $\quad R[0] = R[d_i] \ mod \ N$
8: **end for**
9: **Return** $R[0]$

El SaMA parecería una buena idea, sin embargo, el uso de operaciones dummy permite atacar a este algoritmo por medio de *fault attacks*, convirtiéndolo en un algoritmo inseguro.

3.2. Montgomery powering ladder.

Un nuevo tipo de algoritmo de exponenciación llamado *Montgomery powering ladder* (algoritmo 6), fue propuesto en 2002 por *Joye* y *Yen* [13], este algoritmo es un algoritmo regular, sin embargo, este algoritmo no utiliza operaciones dummy y esta basado en una idea diferente a cualquiera existente hasta el momento de su publicación. Para la ejecución de este algoritmo se utilizaban los valores

$$L_j = \sum_{i=j}^{t-1} d_i 2^{i-j} \ y \ H_j = L_j + 1,$$

donde los valores de L_j y H_j cumplen las igualdades de la ecuación (3.1):

$$L_j = 2L_{j+1} + d_j = L_{j+1} + H_{j+1} + d_j - 1 = 2H_{j+1} + d_j - 2 \tag{3.1}$$

A partir de la ecuación (3.1), se obtiene el comportamiento de la ecuación (3.2)

$$(L_j, H_j) = \begin{cases} (2L_{j+1}, L_{j+1} + H_{j+1}) \;\; si\ d_j = 0 \\ (L_{j+1} + H_{j+1}, 2H_{j+1}) \;\; si\ d_j = 1 \end{cases} \tag{3.2}$$

Teniendo el comportamiento definido en la ecuación (3.2), suponemos que en cada iteración del algoritmo, un primer registro $R[0]$ es usado para contener el valor m^{L_j} y un segundo registro $R[1]$ es usado para contener el valor de m^{H_j}, lo que da como resultado las expresiones (3.3) y (3.4)

$$(m^{L_j}, m^{H_j}) = ((m^{L_{j+1}})^2,\ m^{L_{j+1}} \cdot m^{H_{j+1}})\ si\ d_j = 0 \tag{3.3}$$

$$(m^{L_j}, m^{H_j}) = (m^{L_{j+1}} \cdot m^{H_{j+1}},\ (m^{H_{j+1}})^2)\ si\ d_j = 1 \tag{3.4}$$

Algoritmo 6 Montgomery powering ladder, left-to-right.

1: **Entrada** m, d, N, con $d = (d_{n-1} \cdots d_0)_2$
2: **Salida** $S = m^d\ mod\ N$
3: $R[0] \leftarrow 1$
4: $R[1] \leftarrow m$
5: **for** $n-1$ hasta 0 **do**
6: $\quad R[\bar{d}_i] \leftarrow R[0] \cdot R[1]\ mod\ N$
7: $\quad R[d_i] \leftarrow R[d_i] \cdot R[d_i]\ mod\ N$
8: **end for**
9: **Return** $R[0]$

Las expresiones (3.3) y (3.4), son las que dan el nacimiento y comportamiento del algoritmo Montgomery powering ladder, pero estas a su vez fueron derivadas de la ecuación (3.1). Para evitar confusiones, vamos a mostrar la forma en que se obtiene la primera igualdad de dicha ecuación, es decir, $L_j = 2L_{j+1} + d_j$.

Primero que nada, representamos L_j en la forma

$$L_j = \sum_{i=j}^{t-1} d_i 2^{i-j} = d_j 2^0 + d_{j+1} 2^1 + d_{j+2} 2^2 + \cdots + d_{t-2} 2^{t-2-j} + d_{t-1} 2^{t-1-j} \tag{3.5}$$

Ahora, necesitamos obtener la representación de L_{j+1}, dada por la ecuación (3.6):

$$L_{j+1} = \sum_{i=j+1}^{t-1} d_i \cdot 2^{i-(j+1)} = d_{j+1} 2^0 + d_{j+2} 2^1 + d_{j+3} 2^2 + \cdots + d_{t-2} 2^{t-2-(j+1)} + d_{t-1} 2^{t-1-(j+1)} \tag{3.6}$$

Ya teniendo la representación binaria de L_{j+1}, la multiplicamos por 2, obteniendo la ecuación (3.7):

$$2L_{j+1} = d_{j+1}2^1 + d_{j+2}2^2 + d_{j+3}2^3 + \cdots + d_{t-2}2^{t-2-j} + d_{t-1}2^{t-1-j} \tag{3.7}$$

Observando y comparando las ecuaciones (3.5) y (3.7), podemos llegar al resultado

$$L_j = 2L_{j+1} + d_j 2^0 = 2L_{j+1} + d_j$$

De la ecuación (3.1), solo nos interesa explicar el como se llego a la primera igualdad, ya que las otras dos igualdades se cumplen por medio de sustitución de variables sin un mayor problema.

3.2.1. Características del algoritmo Montgomery powering ladder.

El algoritmo Montgomery powering ladder, presenta muchas características que lo hacen atractivo para su implementación en sistemas reales:

1. El algoritmo es regular, por lo que evita los *simple power analysis* (SPA, explicados en el capítulo 4).

2. Una característica muy interesante, y que proviene directamente del concepto inicial en el cual se baso, es el hecho de que los dos registros del algoritmo tienen la siguiente relación: $R[0]/R[1] = m$, la cual es invariante en cada iteración del algoritmo. Enfocando este punto a nuestro trabajo, podríamos decir que en cada una de las iteraciones del algoritmo, un registro va a almacenar un valor m elevado a un exponente intermedio par y el otro registro va a contener un valor m elevado a un exponente intermedio impar, este dato lo utilizaremos más adelante.

3. Las dos multiplicaciones realizadas en cada iteración comparten un operador, por lo que se puede usar la técnica denominada *Common-multiplicand multiplication* [14] para acelerar la ejecución del algoritmo.

4. Las dos operaciones realizadas en cada iteración son independientes, por lo que pueden ejecutarse de forma paralela.

3.3. Atomicidad de un algoritmo.

Chevallier y otros autores, propusieron la técnica de atomicidad en 2004 [15]. Esta técnica, según sus propios autores, es un esquema genérico y virtualmente aplicable a cualquier algoritmo. Esta idea dice que un proceso puede verse como una secuencia de instrucciones, y cada una de ellas es equivalente a todas las demás instrucciones, es decir, no pueden diferenciarse a través de un análisis del tipo side channel attack.

Un algoritmo de exponenciación modular, puede ser representado como una secuencia de procesos indistinguibles unos de otros, de esta forma, un algoritmo con estas características es seguro contra side channel attack. En [15], es presentado el algoritmo 7, el cual es

una versión segura y atomizada del algoritmo SaM.

Algoritmo 7 SaM protegido por medio de atomicidad.

1: **Entrada** m, d, N, con $d = (d_{n-1} \cdots d_0)_2$
2: **Salida** $S = m^d \ mod \ N$
3: $R[0] \leftarrow 1$
4: $R[1] \leftarrow m$
5: $k = 0$
6: $i = n - 1$
7: **while** $i \geq 0$ **do**
8: $\quad R[0] \leftarrow R[0] \cdot R[k] \ mod \ N$
9: $\quad k \leftarrow k \oplus d_i$
10: $\quad i \leftarrow i - \overline{k}$
11: **end for**
12: **Return** $R[0]$

Este algoritmo, ejecuta una sola multiplicación modular por cada iteración del loop *while*, lo que hace que cada iteración sea similar a todas las demás, este comportamiento se obtiene gracias a la variable k. Vamos a utilizar la tabla 3.1, para explicar como la variable k ayuda a regularizar cada iteración del algoritmo 7.

$d_i = 0$, $k = 0$	$d_i = 1$, $k = 0$
$R[0] \leftarrow R[0] \cdot R[0]$	$R[0] \leftarrow R[0] \cdot R[0]$
$k \leftarrow 0 \oplus 0 = 0$	$k \leftarrow 0 \oplus 1 = 1$
$i = i - 1$	$i = i - 0$
	———————
	$R[0] \leftarrow R[0] \cdot R[1]$
	$k \leftarrow 1 \oplus 1 = 0$
	$i = i - 1$

Tabla 3.1: Comportamiento de k, en el algoritmo 7.

La tabla 3.1, muestra el comportamiento del algoritmo 7 en la posición i de la cadena binaria, en esta tabla suponemos que existen dos posibilidades: donde $d_i = 0$, en la primera columna; y donde $d_i = 1$, en la segunda columna. Podemos observar que cuando $d_i = 0$, el algoritmo solo ejecuta una iteración dentro del ciclo *while*, y vemos que en base al valor de k, pasa a la siguiente posición de i, en este caso $i - 1$. Ahora, cuando $d_i = 1$, vemos que el algoritmo ejecuta dos iteraciones dentro del ciclo *while*, lo que significa el cálculo de dos multiplicaciones. Solo despues de que la segunda multiplicación se ha ejecutado, la variable k permite pasar a la posición $i - 1$ de la cadena binaria. En esta manera, este algoritmo va ejecutando una sola multiplicación por cada iteración realizada dentro del ciclo *while*, y a su vez, va calculando el valor correcto de la operación.

Es necesario hacer notar, que en ambos casos mostrados en la tabla 3.1, la variable k es igual a 0 al inicio de cada posición i, debido a que para que el algoritmo cambie a la siguiente posición en la cadena binaria, es decir de d_{i+1} a d_i, la variable k tiene que ser igual a 0. También es necesario mencionar, que en este algoritmo el numero de iteraciones no es igual al número de bits de la cadena binaria, como se ha presentado en algoritmos anteriores, en este caso, se presentarán en promedio 1,5n iteraciones.

3.4. Algoritmo de Sun Da-zhi.

En 2007, fue propuesto un algoritmo ingenioso y eficiente (algoritmo 8) por *Sun Da-Zhi* y otros autores [16], la principal característica de este algoritmo, es que separa la cadena binaria del exponente d de n bits, en dos cadenas secundarias d_1 y d_2 de $\lceil n/2 \rceil$ bits, en base a estas dos cadenas, se determina que registro utilizar para almacenar la operación realizada en cada iteración. Por medio de d_1 y d_2, sabemos que se pueden utilizar cuatro registros diferentes $C_0 \cdots C_3$. Si $d_{i,1}$ y $d_{i,2}$ (donde $i,1$ es el bit i de la cadena d_1 y $i,2$ es el bit i de la cadena d_2) son iguales a 0, el registro a utilizar es C_0, si $d_{i,1} = 1$ y $d_{i,2} = 0$, el registro a utilizar es C_1 y así sucesivamente para cada registro. Este algoritmo, es un algoritmo regular, por lo que se presentó como un algoritmo seguro contra side channel attacks. En este algoritmo, la operación $sq^n(x)$, significa que el valor x es elevado al cuadrado n veces.

Algoritmo 8 Algoritmo de Da-zhi.

Input m, d, N con $d_1 = d_{\lceil n/2 \rceil,1} \cdots d_{1,1}$ y $d_2 = d_{\lceil n/2 \rceil,2} \cdots d_{1,2}$
Output $C = m^d \ mod \ N$
$s = m$
$C_0 = C_1 = C_2 = C_3 = 1$;
$C_{2 \cdot d_{1,2} + d_{1,1}} = s \cdot C_{2 \cdot d_{1,2} + d_{1,1}}$
for 2 to $\lceil n/2 \rceil$ **do**
    $s = s \cdot s$
    $C_{2 \cdot d_{i,2} + d_{i,1}} = s \cdot C_{2 \cdot d_{i,2} + d_{i,1}}$
end for
$C_1 = C_1 \cdot C_3$
$C_2 = C_2 \cdot C_3$
$C = sq^{(\lceil n/2 \rceil)}(C_1) \cdot C_2$

Este algoritmo, mostrando el fundamento dado por los autores, se basa en el hecho de que si tenemos dos cadenas binarias d_1 y d_2, la ecuación 3.8 se cumple

$$d_1 \ AND \ (NOT \ d_2) + d_1 \ AND \ d_2 = d_1 \tag{3.8}$$

ya que $d_1 \ AND \ (NOT \ d_2 + \ d_2) = d_1 \ AND \ \mathbf{1} = d_1$.

Ahora bien, un comportamiento similar se da para la ecuación 3.9

$$(NOT\ d_1)\ AND\ d_2\ + d_1\ AND\ d_2 = d_2 \tag{3.9}$$

ya que $(NOT\ d_1 +\ d_1)\ AND\ d_2 = d_2\ AND\ \mathbf{1} = d_2$.

En ambas ecuaciones tenemos que $d_i = 1$, $(NOT\ d_i) = 0$ y $\mathbf{1}$=111$\cdots$.

Ahora bien, explicando un poco más a detalle el funcionamiento del algoritmo. En cada iteración se va a almacenar una operación dentro de uno de los cuatro registros C. Cuando en una de las iteraciones del algoritmo tenemos que $d_{i,1} = 1$ y $d_{i,2} = 0$, en esta iteración se almacena el punto donde se cumple la operación lógica $d_1\ AND\ (NOT\ d_2)$ en el registro C_1. Si se da el caso $d_{i,1} = 0$ y $d_{i,2} = 1$, entonces se almacena el punto donde se cumple la operación lógica $(NOT\ d_1)\ AND\ d_2$ en el registro C_2 y si tenemos que $d_{i,1} = 1$ y $d_{i,2} = 1$, el punto que se almacena en C_3, es donde se cumple $d_1\ AND\ d_2$.

En los pasos 10 y 11 del algoritmo 8, vemos que se realizan las operaciones $C_1 = C_1 \cdot C_3$ y $C_2 = C_2 \cdot C_3$, respectivamente. Con estas operaciones, logramos que los valores guardados, a partir de este punto en C_1 y C_2 tengan las formas $m^{d_1\ AND\ (NOT\ d_2) + d_1\ AND\ d_2} = m^{d_1}$ y $m^{(NOT\ d_1)\ AND\ d_2\ + d_1\ AND\ d_2} = m^{d_2}$, en cada caso.

Vamos a dar un ejemplo numérico del comportamiento de este algoritmo. Suponemos que $d = 697 = 1010111001$, ahora en base al algoritmo, descomponemos d en $d_1 = 10101 = 21$ y $d_2 = 11001 = 25$. Las operaciones intermedias están representadas en la tabla 3.2.

i	**Cálculo intermedio**	**i**	**Cálculo intermedio**
1	$s = m$ $C_3 = m^1$	4	$s = m^8$ $C_2 = m^8$
2	$s = m^2$ $C_0 = m^2$	5	$s = m^{16}$ $C_3 = m^{17}$
3	$s = m^4$ $C_1 = m^4$		

Tabla 3.2: Pasoso intermedios del algoritmo 8.

Ya teniendo los valores almacenados en todos los registros C, realizamos la operación $C_1 = C_1 \cdot C_3 = m^4 \cdot m^{17} = m^{21}$, o lo que es lo mismo $C_1 = C_1 \cdot C_3 = m^{00100} \cdot m^{10001} = m^{00100+10001} = m^{10101}$. También realizamos la operación $C_2 = C_2 \cdot C_3 = m^8 \cdot m^{17} = m^{25}$, representada por $C_2 = C_2 \cdot C_3 = m^{01000} \cdot m^{10001} = m^{01000+10001} = m^{11001}$.

Ejecutamos la operación $sq^{(\lceil n/2 \rceil)}(C_1) = sq^5(m^{21}) = m^{672} = m^{1010100000}$, y multiplicamos por C_2, obteniendo $C = m^{672} \cdot m^{25} = m^{697}$, o expresado en forma binaria $C = m^{1010100000} \cdot m^{11001} = m^{1010100000+11001} = m^{1010111001}$. De esta forma, la exponenciación correcta de m^d se ha obtenido.

Capítulo 4

Ataques físicos sobre la exponenciación modular.

En este capítulo, son presentados y explicados los principales ataques físicos que han sido utilizados para vulnerar la seguridad de dispositivos electrónicos, que ejecutan algoritmos criptográficos en su interior. Algunos de estos ataques ya han sido mencionados en el capítulo anterior, pero es en este capítulo, en donde se da la explicación, motivación y funcionamiento de dichos ataques.

En este capítulo, también son presentados los algoritmos que han sido obtenidos con la intención de evitar un cierto ataque o tipo de ataque especifico. Los algoritmos aquí presentes, en general, son versiones mejoradas obtenidas a partir de los algoritmos descritos en el capítulo 3.

4.1. Side channel attacks.

Kocher en 1996, se da cuenta de que usando un equipo de medición, tal como un osciloscopio, se pueden obtener los tiempos de ejecución que realiza un algoritmo cuando este es implementado en un dispositivo electrónico [17], más aún, el se da cuenta de que no solo los tiempos de ejecución podían ser vistos, sino que también, otro tipo de señales analógicas se podían observar, tal como el consumo de potencia [18], esto dio como resultado un tipo de ataques, o técnicas de criptoanálisis, desconocidos hasta ese momento, denominados *side channel attacks* (SCA).

Los SCA, son ataques potentes y de fácil implementación, ya que en términos generales, el atacante solo necesita observar el consumo de potencia, o el tiempo de ejecución correspondiente a los cálculos realizados dentro del dispositivo, para así obtener información relevante acerca de la clave privada del criptosistema. Es necesario mencionar, que cada señal de consumo de potencia obtenida es representativa de los cálculos realizados, en base a esto, es fácil determinar los datos que un atacante busca.

El atacante que quiere obtener información relevante de un criptosistema basándose en

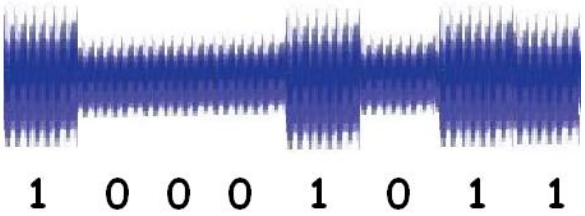

Figura 4.1: Variación del consumo de potencia en una señal, dependiendo de los valores del bit que un algoritmo de exponenciación este ejecutando, señal no real solo explicativa.

los SCA, necesita seguir, en forma general, los siguientes pasos de ataque:

1. Primero que nada, necesita tener acceso al dispositivo que va a ser observado.
2. Necesita tener libertada total sobre el dispositivo, es decir, el puede colocar valores de entrada, realizar tantas ejecuciones del algoritmo como el las requiera, poder desarmarlo si el así lo requiere, etc.
3. Ya teniendo el dispositivo, el atacante va a registrar las señales de consumo de potencia que el dispositivo genera cuando el criptosistema se esta ejecutando dentro de él, las señales son obtenidas a partir de un equipo de medición. Dependiendo del tipo de ataque, pueden ser registradas las señales de una o de varias ejecuciones del algoritmo.
4. El atacante va a estudiar los registros de consumo de potencia. En el estudio de las señales, el atacante puede buscar principalmente: Diferenciaciones en los valores de consumo de potencia, similitudes de la señal en diferentes puntos del registro completo y comportamientos específicos de las formas de onda.
5. Ya habiendo estudiado las señales obtenidas, el atacante solo necesita relacionar los diferentes puntos de la señal completa con los bits de la cadena binaria del exponente.
6. A partir del estudio del consumo de potencia, se pueden determinar los valores secretos del sistema.

Como se mencionó en el capítulo 3, los algoritmos de exponenciación clásicos trabajan de forma irregular ya que realizan un número distinto de cálculos dependiendo del valor del bit ejecutado, por lo que los primeros SCA se enfocaron en atacar dicha operación. Si en un algoritmo de exponenciación, el valor del bit escaneado es igual a 1, el consumo de potencia es mayor que si el valor del bit fuera igual a 0, observando estas variaciones entre los consumos de potencia, se obtiene la representación binaria total del exponente. La figura 4.1, es una ejemplificación de este tipo de ataque, esta imagen no es una muestra real, solo sirve para hacer comprender el esquema de ataque.

Estos ataques fueron de gran impacto, debido a que antes de su publicación, el criptoanálisis estaba enfocado a cuestiones matemáticas y computacionales, pero no se había prestado

atención a la implementación real de los sistemas criptográficos, y por lo consiguiente, no se habían realizado medidas de protección contra este tipo de vulnerabilidades. Debido a esta situación, los SCA atrajeron la atención de muchos investigadores, generando una gran cantidad de ataques de este tipo y buscando la forma de evitarlos.

El primer esfuerzo por evitar este tipo de ataques, fue descrito por Coron [12], diseñando un algoritmo regular en su funcionamiento.

4.1.1. Análisis de potencia simple y diferencial.

Los ataques que miden el consumo de potencia de un dispositivo, pueden ser clasificados en dos tipos: *análisis de potencia simple* (SPA, por sus siglas en ingles) y *análisis de potencia diferencial* (DPA, por sus siglas en ingles) [18]. Los SPA, son ataques que solo requieren una o pocas mediciones del consumo de potencia de los dispositivos que ejecuten un algoritmo en su interior, por otro lado, los DPA son ataques que requieren que el atacante recolecte muchas mediciones de potencia del dispositivo y después realice cálculos estadísticos para determinar la cadena binaria del exponente (que es básicamente la clave privada). Por estos motivos, es más fácil llevar a cabo ataques por medio de SPA que de DPA.

Aleatorización del mensaje de entrada.

Contrario a lo que podría suponerse, se han obtenido mejores resultados en la forma de evitar DPA que en evitar SPA. Las técnicas que mostraron ser eficientes contra los DPA, están basadas en la aleatorización del mensaje de entrada m, es decir, al inicio de la ejecución de un algoritmo, el valor de entrada m es protegido con un valor aleatorio r, r provoca que cada valor de entrada sea independiente de cualquier otro valor, por lo que un análisis estadístico no será de utilidad, ya que no existe dependencia de datos.

A continuación, se enumeran dos tipos de técnicas conocidas de aleatorización del mensaje para evitar los DPA:

1. Técnica propuesta por *Kim* y *Quisquater* (KQ) [19], representada en el algoritmo 9. Se tiene un valor $m \cdot r$ al principio del *loop for*, en cada iteración del loop se calculará primero a $m^{2^i} \cdot r^2$ y después $m^{2^i} \cdot r^2 \cdot r^{-1}$, por lo que r permanecerá constante en cada iteración, al final del ciclo se realizara el cálculo $m^d \cdot r \cdot r^{-1} = m^d$.

2. Técnica propuesta por *Fumaroli* y *Vigilant* (FV) [20], representada por el algoritmo 10. Este algoritmo proviene directamente del Montgomery powering ladder. En cada iteración del loop principal, el valor r almacenado en los registros $R[0]$ y $R[1]$ se va exponenciando al cuadrado, por lo que al final del algoritmo, en $R[0]$ existirá un valor r^{2^n}, para esta técnica tenemos un registro adicional $R[2]$ que contiene el valor r^{-2^n}, y al final del loop calculamos $m^d \cdot r^{2^n} \cdot r^{-2^n} = m^d$. Cabe mencionar que este algoritmo requiere n exponenciaciones cuadradas adicionales en comparación con el

Montgomery powering ladder.

Algoritmo 9 Algoritmo KQ.

Entrada $m,\ d,\ N,\ r,\ a = r^{-1}$, con $d = (d_{n-1} \cdots d_0)_2$
Salida $S = m^d \ mod\ N$
$C = r \ mod\ N$
$R[0] = a \ mod\ N$
$R[1] = m \cdot a \ mod\ N$
for $n - 1$ to 0 **do**
    $C = C \cdot C \ mod\ N$
    $C = C \cdot R[d_i] \ mod\ N$
end for
Return $C \cdot a$

Algoritmo 10 Algoritmo de Fumaroli y Vigilant.

Entrada m, d, N, con $d = (d_{n-1} \cdots d_0)_2$
Salida $S = m^d \ mod\ N$
$R[0] \leftarrow r$
$R[1] \leftarrow rm$
$R[2] \leftarrow r^{-1}$
for $n - 1$ to 0 **do**
    $R[\bar{d}_i] \leftarrow R[\bar{d}_i] \cdot R[d_i] \ mod\ N$
    $R[d_i] \leftarrow R[d_i] \cdot R[d_i] \ mod\ N$
    $R[2] \leftarrow R[2] \cdot R[2]$
end for
Return $R[0] \cdot R[2]$

Tanto la técnica KQ como la técnica FV, son útiles para proteger los algoritmos contra DPA, sin embargo, estos esquemas son atacados por un SPA y un *fault attack*, respectivamente (estos ataques particulares serán mencionados y explicados más adelante).

Otro ejemplo de algoritmo protegido contra DPA por medio de la aleatorización del mensaje m, es el algoritmo propuesto por *Mamiya* y otros autores en [21], este esquema lo podemos ver como el algoritmo 11.

Debido principalmente, a que las técnicas de aleatorización son usadas para evitar DPA de forma eficiente, en este trabajo solo nos enfocaremos en técnicas para evitar SPA, tal como el ataque $N - 1$, explicado en la sección 4.1.2.

Algoritmo 11 BRIP.

1: **Entrada** m, d, N, con $d = (d_{n-1} \cdots d_0)_2$
2: **Salida** $S = m^d \ mod \ N$
3: $R[0] = r$
4: $R[1] = r^{-1}$
5: $R[2] = m \cdot r^{-1}$
6: **for** $n - 1$ to 0 **do**
7: $R[0] = R[0]^2$
8: **if** $d_i = 0$ **then**
9: $R[0] = R[0] \cdot R[1]$
10: **else**
$R[0] = R[0] \cdot R[2]$
11: **end if**
12: **end for**
13: **Return** $R[0] \cdot R[1]$

4.1.2. Ataque $N - 1$.

En 2005, *Yen* y otros autores propusieron el ataque $N - 1$ en [22]. Ellos implementaron dicho ataque sobre los algoritmos de exponenciación modular 5 y 11. Este ataque, asume que las mediciones de potencia obtenidas desde un dispositivo son representativas de los cálculos realizados dentro de él, es decir, cada señal medida puede representar a un cálculo particular dentro del dispositivo, y a partir de este supuesto, dos segmentos de la señal completa pueden ser comparados para determinar si las operaciones realizadas en ambos puntos son iguales, aún si el adversario no sabe que valores están siendo calculados. Por ejemplo, si A y B son valores donde $A = B$, las colisiones de A^2 y B^2 pueden ser detectadas, aún cuando los valores de A y B son desconocidos por el atacante.

Este ataque esta basado en dos observaciones:

1. Si un criptosistema trabaja con modulo N, entonces $(N-1)^2 \equiv 1 \ mod \ N$, esta observación puede ser generalizada y obtener $(N-1)^j \equiv 1$, para cualquier entero par j.

2. Si un criptosistema trabaja con modulo N, entonces $(N-1)^1 \equiv N-1 \ mod \ N$, esta observación puede ser generalizada y obtener $(N-1)^i \equiv 1$, para cualquier entero impar i.

Si un atacante coloca un valor de entrada $m = N-1$ en un algoritmo, el algoritmo ejecuta n iteraciones en su ciclo *for*, y cada una de estas iteraciones realizará exponenciaciones intermedias pares o impares, dependiendo del valor del bit que se esta escaneando en cada una de ellas, a partir de este hecho, el valor de una exponenciación intermedia al final de cada una de las iteraciones del algoritmo atacado, puede tomar solamente uno de dos posibles valores: 1 si el valor del bit siendo procesado es igual a 0 (la exponenciación intermedia es par), o $N-1$ si el valor del bit siendo procesado es igual a 1 (la exponenciación intermedia es impar). De esta manera, un atacante puede observar solo dos patrones de

comportamiento a través de toda la medición de potencia, y así determinar la cadena binaria del exponente.

En [22], también se presenta una versión del ataque $N-1$ que se realiza en función de dos mensajes relacionados entre si, la vertiente de este ataque se explica de la siguiente forma:

1. Un atacante escoge el valor de entrada aleatorio m_1, y a partir de él, calcula el valor $m_2 = m_1 \cdot (N-1)$.

2. El atacante coloca los valores m_1 y m_2, de manera no consecutiva, como entrada a un algoritmo de exponenciación y obtiene el comportamiento completo del consumo de potencia de ambas ejecuciones del algoritmo.

3. Teniendo ambas mediciones, el atacante puede compararlas y encontrar *colisiones*. Las colisiones son puntos de la señal completa donde el comportamiento es el mismo, o dicho de otra forma, secciones en donde se presenta la misma forma de onda. Si existe una colisión entre ambas mediciones, el valor del bit d_i en ese punto es igual a 0, por el contrario, si no existe colisión, el valor del bit en ese punto es igual a 1, esto debido a que si $d_i = 0$, existe un comportamiento dado por:

$$(m_{2_i})^2 = (m_{1_i} \cdot (N-1))^2 = (m_{1_i})^2.$$

4. A partir de los puntos donde existen colisiones y donde no existen colisiones, el atacante deduce de forma rápida la cadena binaria del exponente.

En 2008, *Miyamoto* y otros autores [23] analizaron y demostraron la efectividad del ataque $N-1$ en la práctica, ellos obtuvieron mediciones de potencia en las cuales se identifican claramente los dos patrones observables.

Una de las características de este ataque, es que puede ser usado contra algoritmos que utilicen la protección del mensaje de entrada m por medio de un valor aleatorio r, tal es el caso del algoritmo 11. Este ataque es ilustrado en la tabla 4.1, la cual muestra el comportamiento del algoritmo 11 cuando es sometido a un ataque $N-1$. En este ejemplo, se asume que $m = N-1$ y $d = 89 = 1011001$.

4.2. Fault attacks.

Como ya se mencionó, los SCA fueron el primer acercamiento hacia los ataques físicos, pero a partir de ellos, se han desarrollado nuevos ataques que requieren un dispositivo real para su funcionamiento, los *fault attacks* (FA) fueron uno de esos nuevos ataques. Los FA, fueron originalmente propuestos por *Boneh, DeMilo* y *Lipton* en 1997 [24]. Los FA, son más agresivos que los SCA debido a que los FA interfieren y alteran el funcionamiento de

i	d_i	**Pasos Intermedios**	**Resultado(Patron)**
6	1	$R[0] = r^2$ R[0]=m$\cdot r$	$(N-1) \cdot r$
5	0	$R[0] = m^2 \cdot r^2$ R[0]=m$^2 \cdot r$	$(1) \cdot r$
4	1	$R[0] = m^4 \cdot r^2$ R[0]=m$^5 \cdot r$	$(N-1) \cdot r$
3	1	$R[0] = m^{10} \cdot r^2$ R[0]=m$^{11} \cdot r$	$(N-1) \cdot r$
2	0	$R[0] = m^{22} \cdot r^2$ R[0]=m$^{22} \cdot r$	$(1) \cdot r$
1	0	$R[0] = m^{44} \cdot r^2$ R[0]=m$^{44} \cdot r$	$(1) \cdot r$
0	1	$R[0] = m^{88} \cdot r^2$ R[0]=m$^{89} \cdot r$	$(N-1) \cdot r$

Tabla 4.1: Algoritmo 11 cuando es sometido a un ataque $N-1$.

un dispositivo electrónico por medio de pulsos eléctricos, magnéticos, térmicos, e inclusive lumínicos.

Cuando un dispositivo esta ejecutando un algoritmo dentro de si, un atacante coloca un error a dicho dispositivo por medio de un pulso eléctrico, este pulso eléctrico altera el funcionamiento del elemento electrónico y genera un error en el cálculo que se esta realizando en ese momento, esto a su vez, provoca que el valor guardado en un registro sea un valor erróneo z, y de esta forma el resultado de salida del algoritmo, es un resultado incorrecto. En base al valor de salida erróneo, el adversario puede fácilmente determinar los datos secretos del criptosistema atacado.

En [24], los autores muestran la forma de vulnerar una versión rápida del algoritmo RSA por medio de un fault. La versión rápida de RSA la podemos ver en el algoritmo 12, donde $d,\ p,\ q$ son los valores generados por el algoritmo RSA y $i_q = q^{-1}\ mod\ p$.

Algoritmo 12 Versión rápida de RSA.

1: **Entrada** $d,\ p,\ q,\ m$
2: **Salida** Mensaje m encriptado.
3: $d_p = d\ mod\ (p-1)$
4: $d_q = d\ mod\ (q-1)$
5: $S_p = m^{d_p}\ mod\ p$
6: $S_q = m^{d_q}\ mod\ q$
7: $S = ((S_p - S_q) \cdot i_q\ mod\ p) \cdot q + S_q$

Observamos que el algoritmo 12 realiza dos exponenciaciones modulares en su ejecución. Cuando se ataca este algoritmo con un FA, el atacante no busca obtener el valor del bit que se esta ejecutando, sino que el atacante busca generar un valor erróneo z en una de las operaciones de exponenciación modular, más no en ambas. Con el valor erróneo, el atacante puede determinar un factor primo del modulo N del criptosistema, y a partir de dicho valor, ir obteniendo todos los demás valores privados, con lo cual ha roto la seguridad completamente.

Explicando un poco más a detalle este ataque, cuando el atacante ha inducido un error

en la exponenciación de S_p, dando como resultado $\tilde{S_p}$, al sustituir este valor erróneo en la ecuación $S = ((\tilde{S_p} - S_q) \cdot i_q \ mod \ p) \cdot q + S_q$, obtendremos un valor de salida incorrecto $\tilde{S}$. Ya teniendo el valor incorrecto $\tilde{S}$, el atacante ejecuta nuevamente el algoritmo 12 sin colocar ningún tipo de fault y así obtiene el valor correcto S. Teniendo un valor de salida S y un valor $\tilde{S}$, es posible calcular el factor primo q por medio de $gcd((S - \tilde{S}), N)$ ya que

$$gcd((((\tilde{S_p} - S_q) \cdot i_q \ mod \ p) \cdot q + S_q) - (((S_p - S_q) \cdot i_q \ mod \ p) \cdot q + S_q), p \cdot q)$$
$$gcd((((\tilde{S_p} - S_q) \cdot i_q \ mod \ p) \cdot q + S_q) - ((S_p - S_q) \cdot i_q \ mod \ p) \cdot q - S_q, p \cdot q)$$
$$gcd(((\tilde{S_p} - S_q) \cdot i_q \ mod \ p) \cdot q - ((S_p - S_q) \cdot i_q \ mod \ p) \cdot q, p \cdot q)$$
$$gcd((\tilde{S_p} \cdot i_q \ mod \ p) \cdot q - (S_q \cdot i_q \ mod \ p) \cdot q - (S_p \cdot i_q \ mod \ p) \cdot q + (S_q \cdot i_q \ mod \ p) \cdot q, p \cdot q)$$
$$gcd((\tilde{S_p} \cdot i_q \ mod \ p) \cdot q - (S_p \cdot i_q \ mod \ p) \cdot q, p \cdot q)$$
$$gcd((\tilde{S_p} \cdot i_q \ mod \ p - S_p \cdot i_q \ mod \ p) \cdot q, p \cdot q) = q$$

En 2002, *Yen* y otros autores usaron los FA para atacar el algoritmo SaMA (algoritmo 5) [25], tomando ventaja del hecho de que este algoritmo utiliza operaciones dummy, es decir, se coloca un fault durante una iteración del algoritmo, lo que provoca un error z en el registro correspondiente a la operación calculada. Si el fault es colocado cuando el valor del bit es igual a 0, el valor z no altera el resultado final del cálculo, pero si el valor del bit es igual a 1, el resultado final es un resultado erróneo, de esta forma un atacante puede ir construyendo la cadena binaria del exponente, solo observando en que posiciones el fault genera un valor erróneo y en cuales no.

4.2.1. Test de coherencia para evitar FA.

Para evitar los FA, se idearon los *test de coherencia* en [26] y en [1] (algoritmos 13 y 14 respectivamente). Los test de coherencia, realizan un cálculo y una igualdad al final de la ejecución del algoritmo, este cálculo involucra a todos los registros utilizados a través de él, si la igualdad se cumple, significa que ningún registro ha sido alterado por un error o ataque y se devuelve el valor calculado, pero si la igualdad no se cumple, significa que un registro ha sido alterado, y por lo tanto, un ataque se ha llevado a cabo y no se devuelve el valor calculado.

A continuación se explica como funciona el test de coherencia sobre el algoritmo 13, explicación tomada de [26]:

- El valor $R[2]$ es independiente de d, por lo que al final del algoritmo, $R[2]$ satisface

$$R[2] = m^{2^n} \ mod \ N$$

- El valor $R[1]$ es el resultado de la exponenciación modular de m por el complemento binario de d, denotado $\overline{d}$, y satisface $\overline{d} = 2^n - d - 1$:

$$R[1] = m^{2^n - d - 1} \ mod \ N$$

Algoritmo 13 Square and multiply always con test de coherencia.

1: **Entrada** m, d, N, con $d = (d_{n-1} \cdots d_0)_2$
2: **Salida** $S = m^d \ mod \ N$
3: $R[0] \leftarrow 1$
4: $R[1] \leftarrow 1$
5: $R[2] \leftarrow m$
6: **for** 0 to $n-1$ **do**
7: $\quad R[\bar{d}_i] \leftarrow R[\bar{d}_i] \cdot R[2] \ mod \ N$
8: $\quad R[2] \leftarrow R[2] \cdot R[2] \ mod \ N$
9: **end for**
10: **if** $(m \cdot R[0] \cdot R[1] = R[2] \ mod \ N)$ && $(R[2] \neq 0)$ **then**
11: $\quad$ **Return** $R[0]$
12: **end if**

- Ya que $R[0] = m^d \ mod \ N$, la siguiente relación se mantiene con el contenido de $R[2]$ despues del loop:

$$m \cdot R[0] \cdot R[1] = m \cdot m^d \cdot m^{2^n - d - 1} = m^{2^n} = R[2] \ mod \ N \tag{4.1}$$

Para este algoritmo, si $R[2]$ es igual a 0, entonces $R[0]$ o $R[1]$ y $R[2]$ podrán ser nulos y el chequeo de coherencia fallara al detectar la inducción de la falla. Para prevenir tal ataque, se usa la verificación $R[2] \neq 0$.

Ahora bien, se dará una pequeña explicación de como funciona el test de coherencia sobre el algoritmo 14, explicación tomada de [1]:

Este algoritmo tiene tres registros que son iniciados con $R[0] = r,\ R[1] = r^{-1}$ y $R[2] = m$, donde r es un valor aleatorio que se va a encargar de proteger el mensaje.

Al final de la ejecución del algoritmo, los tres registros contendrán los valores $R[0] = r \cdot m^d,\ R[1] = r^{-1} \cdot m^{\overline{d}}$ donde $\overline{d}$ representa el complemento binario de d tal que $d + \overline{d} = 2^n - 1$ y $R[2] = m^{2^n}$. Por lo tanto, el cálculo del test de coherencia de $R[0] \cdot R[1] \cdot m$ es igual al registro $R[2]$.

$$r \cdot m^d \cdot r^{-1} \cdot m^{\overline{d}} \cdot m = m^d \cdot m^{\overline{d}} \cdot m = m^{d + \overline{d} + 1} = m^{2^n}$$

Los test de coherencia parecían una buena propuesta contra los FA, sin embargo, *Kim* y *Quisquater* en [27] demostraron de forma práctica que es posible realizar dos faults sobre una misma ejecución del algoritmo, un fault para alterar un registro y un segundo fault para saltarse el test de coherencia, lo que provoca que el valor erróneo sea devuelto y pueda ser utilizado por un atacante.

Otra desventaja de este tipo de test cuando se utilizan en algoritmos con operaciones dummy, es que si el test falla, significa que hubo un error y no se devuelve el resultado de salida, al no darse el resultado de salida o al realizarse nuevamente la operación, un

Algoritmo 14 Algoritmo propuesto en [1].

Entrada m, d, N, con $d = (d_{n-1} \cdots d_0)_2$
Salida $S = m^d \ mod \ N$
$R[0] \leftarrow r$
$R[1] \leftarrow r^{-1}$
$R[2] \leftarrow m$
for 0 to $n-1$ **do**
  $R[\bar{d}_i] \leftarrow R[\bar{d}_i] \cdot R[2] \ mod \ N$
  $R[2] \leftarrow R[2] \cdot R[2] \ mod \ N$
  if $(R[0] \cdot R[1] \cdot m = R[2] \ mod \ N)$ **then**
    Return $R[0] \cdot r^{-1}$
  end if
end for

atacante puede saber que el valor del bit que se estaba ejecutando en el momento del fault es 1, y si el resultado de la operación es devuelto, el bit es igual a 0.

4.2.2. FA usando el símbolo de Jacobi.

En 2006, *Boreale* propuso un nuevo tipo de ataques contra la exponenciación modular [28], implementado sobre el algoritmo 4. El coloca un fault z en $R[1]$ cuando una exponenciación cuadrada es ejecutada en la iteración $i-1$ del loop *for*, entonces, dependiendo del valor de $(\tilde{S}/N)$, donde $\tilde{S}$ es el valor de salida erróneo, es posible determinar el valor del bit d_i. Este esquema trabaja asumiendo que $(m/N) = 1$ y que si el valor del bit en la i-teaba iteración es igual a 0, el fault no afectara el valor del SJ de $(R[0]_i/N) = 1$, pero si $d_i = 1$, z afecta el SJ del registro $R[0]_i$, y este registro puede tener un SJ $(R[0]_i/N) = -1$, el cual afectara el SJ del resultado de salida. Por lo tanto, bajo un ataque de este tipo, existen dos posibilidades: si $(\tilde{S}/N)$ siempre es igual a 1, $d_i = 0$, pero si $(\tilde{S}/N)$ es igual a -1, $d_i = 1$. De esta forma, un atacante puede deducir fácilmente la cadena binaria del exponente.

En 2010, *Schmidt* [2] propuso un ataque que consiste en colocar un mensaje de entrada m, tal que $(m/N) = -1$, en el algoritmo 10, después de esto, por medio de un fault, la operación $R[d_i] = R[d_i]^2$ es saltada. Dependiendo del valor resultante $\tilde{S}$, es posible determinar si los valores de d_i y d_{i+1} son iguales entre si, o diferentes. Este ataque esta definido como el algoritmo 15.

Un ejemplo del ataque de Schmidt sobre el esquema de FV, puede ser visto en la tabla 4.2, en este ejemplo fue supuesto que $\left(\frac{m}{N}\right) = -1$ y $d = 19 = 10011$.

Para explicar un poco más el ataque de Schmidt, vamos a ver los casos cuando $d_i \neq d_{i+1}$ y cuando $d_i = d_{i+1}$ por medio de las tablas 4.3 y 4.4, para ambas tablas, se obtuvieron los productos intermedios a partir de los valores de entrada de la tabla 4.2.

Algoritmo 15 Ataque propuesto en [2].

Tenemos que encontrar $d = (d_{n-1}, \cdots, d_0)$.
Ssuponemos $d_{n-1} = 1$
for $n-2$ to 0 **do**
  Escoge $m \in \mathbb{Z}_N$ tal que $\left(\frac{m}{N}\right) = -1$
  Calcular $\tilde{S}$ con la i-teaba operación cuadrada saltada
  if $\left(\frac{\tilde{S}}{N}\right) = -1$ **then**
    $d_i = d_{i+1}$
  else
    $d_i = 1 \oplus d_{i+1}$
  end if
end for
Return d

i	d_i	Productos intermedios	Símbolo de Jacobi
4	1	$R[0] = m \cdot r^2$	$(R[0]/N) = (-1)(1) = -1$
		$R[1] = m^2 \cdot r^2$	$(R[1]/N) = (1)(1) = 1$
3	0	$R[0] = (m \cdot r^2)^2 = m^2 \cdot r^4 = FA$	$(R[0]/N) = (1)(1) = 1$
		$R[1] = m^3 \cdot r^4$	$(R[1]/N) = (-1)(1) = -1$
2	0	$R[0] = (m \cdot r^2)^2 = m^2 \cdot r^4$	$(R[0]/N) = (1)(1) = 1$
		$R[1] = m^3 \cdot r^4 \cdot m \cdot r^2 = m^4 \cdot r^6$	$(R[1]/N) = (1)(1) = 1$
1	1	$R[0] = m^6 \cdot r^{10}$	$(R[0]/N) = (1)(1) = 1$
		$R[1] = (m^4 \cdot r^6)^2 = m^8 \cdot r^{12}$	$(R[1]/N) = (1)(1) = 1$
0	1	$R[0] = m^{14} \cdot r^{22}$	$(R[0]/N) = (1)(1) = 1$
		$R[1] = (m^8 \cdot r^{12})^2 = m^{16} \cdot r^{24}$	$(R[1]/N) = (1)(1) = 1$

Tabla 4.2: Algoritmo 10 ejecutado con un FA, donde $d_{i+1} \neq d_i$.

	d_i	Productos intermedios
$i+1$	1	$R[0]_{i+1} = m \cdot r^2$
		$R[1]_{i+1} = m^2 \cdot r^2$
i	0	$R[0]_i = (R[0]_{i+1})^2 = (m \cdot r^2)^2 = m^2 \cdot r^4 = FA$
		$R[1]_i = R[1]_{i+1} \cdot R[0]_{i+1} = m^3 \cdot r^4$
$i-1$	0	$R[0]_{i-1} = (R[0]_{i+1})^2 = (m \cdot r^2)^2 = m^2 \cdot r^4$
		$R[1]_{i-1} = R[1]_i \cdot R[0]_{i+1} = m^3 \cdot r^4 \cdot m \cdot r^2 = m^4 \cdot r^6$

Tabla 4.3: Caso donde $d_{i+1} \neq d_i$.

Bajo circunstancias libres de ataques, en cada una de las iteraciones del algoritmo 6 (y por consiguiente del algoritmo 10), siempre se realiza una multiplicación modular y una exponenciación modular. El registro que guarda el resultado de la exponenciación modular,

	d_i	Productos intermedios
$i+1$	0	$R[0]_{i+1} = (R[0]_{i+2})^2 = (m \cdot r^2)^2 = m^2 \cdot r^4$ $R[1]_{i+1} = R[1]_{i+2} \cdot R[0]_{i+2} = m^3 \cdot r^4$
i	0	$R[0]_i = (R[0]_{i+1})^2 = (m^2 \cdot r^4)^2 = m^4 \cdot r^8 = FA$ $R[1]_i = R[1]_{i+1} \cdot R[0]_{i+1} = m^3 \cdot r^4 \cdot m^2 \cdot r^4 = m^5 \cdot r^8$
$i-1$	1	$R[0]_{i-1} = R[1]_i \cdot R[0]_{i+1} = m^5 \cdot r^8 \cdot m^2 \cdot r^4 = m^7 \cdot r^{12}$ $R[1]_{i-1} = (R[1]_i)^2 = (m^5 \cdot r^8)^2 = m^{10} \cdot r^{16}$

Tabla 4.4: Caso donde $d_{i+1} = d_i$.

siempre guarda un valor m elevado a un exponente par, por el contrario, el registro que guarda el resultado de la multiplicación modular siempre guarda un valor m elevado a un exponente impar, esto se debe a que la multiplicación se realiza con los valores anteriores almacenados en los dos registros utilizados (uno con exponente par y uno con exponente impar). Este comportamiento, nos permite mantener la relación $R[0]/R[1] = m$ en cada una de las iteraciones del algoritmo.

Es necesario hacer la siguiente observación: Cuando el valor del bit i que se esta ejecutando es igual a 0, el registro elevado al cuadrado es $R[0]$, y cuando el valor del bit es igual a 1, el registro elevado al cuadrado es $R[1]$.

En la tabla 4.3, vemos que la operación saltada en la iteración i es $R[0]_i = R[0]^2_{i+1}$, si ningún FA fuera ejecutado, el exponente intermedio correspondiente a esa operación sería un exponente par (en este caso m^2), ahora bien, en la iteración $i-1$ sabemos que el registro $R[1]_{i-1}$, guardaría el resultado de la multiplicación modular $R[0]_i \cdot R[1]_i$ que sería un valor con exponente impar. Cuando nos saltamos la exponenciación cuadrada por medio de un FA en la iteración i, el cálculo de la multiplicación modular en la iteración $i-1$ esta dada por $R[1]_i \cdot R[0]_{i+1}$, ambos registros con un valor m elevado a un exponente impar. El exponente impar obtenido en ambos registros, $R[0]_{i+1}$ y $R[1]_i$, se debe a que los valores de los bits en i y $i+1$ son diferentes, es decir, en la iteración $i+1$ el bit $d_{i+1} = 1$, por lo que el valor almacenado en $R[0]_{i+1}$ tiene un exponente impar; en la iteración i, el bit $d_i = 0$, por lo que el valor almacenado en $R[1]_i$ también es elevado a un exponente impar. Como sabemos, la multiplicación de dos exponentes impares da como resultado un exponente par, por tal motivo, en la iteración $i-1$ cualquier registro que almacene el resultado de la multiplicación modular, almacenara un valor m elevado a un exponente par, y el registro restante almacenara el resultado de la exponenciación cuadrada, que también es par, de este modo todos los cálculos que se realicen a partir de la iteración $i-1$ serán realizados solamente con valores m elevados a un exponente par, rompiendo la relación $R[0]/R[1] = m$ del algoritmo, y a partir de este punto todos los valores calculados tendrán un SJ igual a 1.

Ahora explicaremos el caso cuando $d_i = d_{i+1}$, basándonos en la tabla 4.4. En la iteración i nos saltamos la operación $R[0]_i = R[0]^2_i$, sin embargo, el tipo de exponente del valor m almacenado en $R[0]_{i-1}$ permanece siendo impar, esto se debe que para realizar la multiplicación modular son necesarios los valores anteriores de los dos registros utilizados,

ya hemos hecho notar que para realizar la correcta multiplicación, un registro debe contener un valor m elevado a un exponente par y el otro registro un valor m elevado a un exponente impar. Cuando $d_i = d_{i+1}$, el mismo registro en ambas iteraciones, $R[0]_{i+1}$ y $R[0]_i$, contiene un valor con un exponente par, por lo que no importa que sustituyamos $R[0]_{i+1}$ o $R[0]_i$ en la multiplicación modular de la iteración $i-1$, ya que la estructura de la multiplicación se va a mantener, es decir, un elemento elevado a un exponente par multiplicado por un valor elevado a un exponente impar. Esta situación rompe con la relación $R[0]/R[1] = m$, pero mantiene la relación entre los dos registros, en el aspecto de que un registro contiene un valor elevado a un exponente par y el otro un valor elevado a un exponente impar en cada iteración del algoritmo, por lo que el SJ se alterna en cada iteración entre -1 y 1.

Como sabemos, en sistemas tales como RSA, el exponente utilizado siempre es un número impar, por lo tanto al final de la ejecución del algoritmo de exponenciación, el SJ siempre será igual a -1 si $d_i = d_{i+1}$, pero este valor se vera alterado si $d_i \neq d_{i+1}$.

4.2.3. Ataque contra el algoritmo de Sun Da-Zhi.

El algoritmo de Sun Da-Zhi, fue presentado como un algoritmo eficiente y seguro, sin embargo, se demostró que es posible obtener hasta 3/4 partes de la representación binaria del exponente [29] por medio de FA y por medio del cálculo del SJ.

El algoritmo 8, puede ser atacado en una serie de tres pasos:

1. Cuando $d_{i,1} = d_{i,2} = 0$, el cálculo realizado y almacenado en C_0, no se vuelve a utilizar durante toda la ejecución del algoritmo, por lo que, si un fault ocurre en ese momento, el resultado final no se verá alterado y de esta forma el atacante puede reconocer cuando se uso C_0, y así reconocer los bits de la cadena binaria del exponente cuando $d_{i,1} = d_{i,2} = 0$, de esta forma obtenemos en promedio una cuarta parte de la representación binaria del exponente.

2. Ya teniendo los primeros bits de la cadena binaria, ahora nuevamente se colocan faults en las iteraciones del algoritmo, al final del algoritmo calculamos el SJ de $\tilde{S}$. Solamente cuando $d_{i,1} = 1$ y $d_{i,2} = 0$, el SJ siempre será igual a 1. El SJ igual a 1 siempre para C_1, se obtiene debido a que en el último paso del algoritmo 8, el valor m^{C_1} siempre es elevado a un exponente par, de esta forma se elimina todo SJ que pudiera tener un valor igual a -1, esta elevación a un exponente par solo se da para C_1, por lo que es fácil identificar los puntos donde este registro es utilizado. Identificando las posiciones de este registro, se determina otra cuarta parte de la cadena binaria total del exponente.

3. Observando el comportamiento de los registros, podemos ver que para que los registros C_2 y C_3 se utilicen, es necesario que $d_{i,2} = 1$ bajo cualquier circunstancia, por lo tanto podemos determinar que todos los valores de los bits restantes de la cadena binaria d_2 son igual a 1.

4. Hasta aquí, es posible conocer 3/4 partes, en promedio, de la cadena binaria del exponente, restando por encontrar solo la mitad de bits de la subcadena d_1.

Para mostrar este ataque daremos el siguiente ejemplo:

Para este ejemplo se tomo un valor aleatorio desconocido de dieciocho bits para el exponente, y un valor de entrada m tal que $(m/N) = 1$.

Se usaron los siguientes valores de entrada:

$m = 523489278217456019834593$
$N = 871258948162375898716235 81$

Antes que nada, se obtuvo el resultado correcto de la exponenciación, resultado dado por 84982614287872105537806915. Después, se colocaron FA sobre cada iteración ejecutada por el algoritmo, en el punto donde se realiza la multiplicación modular, obteniendo los resultados de la tabla 4.5.

Posición atacada	Resultado final
1	786770141613642574945049 19
2	560917410377094997017127 65
3	508060306368437484505569 41
4	84982614287872105537806915
5	4495242442647294430897836
6	84982614287872105537806915
7	447893522912747169485341 42
8	427213333524527893892312 67

Tabla 4.5: Resultados del primer paso del ataque propuesto.

Por medio de la tabla 4.5, sabemos que en las posiciones 4 y 6, los bits $d_{i,1} = d_{i,2} = 0$, ya que a pesar del error generado, el resultado no se vio alterado. De acuerdo con esta tabla, podemos determinar que $d_1 = xx0x0xxxx$ y $d_2 = xx0x0xxxx$ y por lo tanto $d = xx0x0xxxxxx0x0xxxx$.

Después de haber obtenido los primeros bits, volvemos a colocar *faults* en la ejecución del algoritmo y calculamos el símbolo de Jacobi para cada uno de los resultados de salida del mismo, con lo que obtenemos los resultados de la tabla 4.6.

La primera fila de la tabla 4.6 representa el número de posición que fue atacada. El bit 0, o columna 0, de la representación binaria no fue tomada en cuenta debido a que el cálculo de esta posición se realiza fuera del ciclo *if* principal.

0	1	2	3	4	5	6	7	8
0	1	-1	1	1	-1	1	-1	1
0	1	1	1	1	1	1	-1	1
0	1	1	1	1	1	1	1	1
0	1	-1	1	1	-1	1	1	1
0	1	-1	1	1	1	1	1	1
0	1	1	1	1	1	1	-1	1
0	1	1	1	1	1	1	-1	1
0	1	-1	1	1	1	1	-1	1
0	1	1	1	1	1	1	1	1
0	1	-1	1	1	-1	1	-1	1
0	1	-1	1	1	1	1	1	1

Tabla 4.6: Resultados del segundo paso del ataque propuesto.

En la tabla 4.6, vemos que las posiciones 1, 3, 4, 6, y 8 siempre tienen un SJ igual a 1, pero sabemos que las posiciones 4 y 6 corresponden a C_0. En el paso 12 del algoritmo 8, notamos que C_1 siempre es elevado al cuadrado, eliminando así todo símbolo de Jacobi igual a -1, por lo que podemos deducir que C_1 se esta ejecutando en las posiciones 1, 3 y 8 tal que $d_{i,1} = 1$ y $d_{i,2} = 0$, por lo tanto, podemos sustituir algunas x en $d_1 = 1x0x01x1x$ y $d_2 = 0x0x00x0x$, así $d = 1x0x01x1x0x0x00x0x$.

Para que los registros C_2 y C_3 se utilicen, es necesario que $d_{i,2} = 1$ bajo cualquier circunstancia, por lo que podemos determinar que $d_2 = 010100101$ y sustituir en $d = 1x0x01x1x010100101$, con lo cual solo nos resta determinar el valor de cada uno de los bits señalados con una x. Determinar estos bits restantes puede ser relativamente sencillo por medio de fuerza bruta, ya que si utilizamos una cadena de 1024 bits, solo se requerirá determinar 256 bits en promedio.

El exponente d para este experimento, fue elegido de forma aleatoria y oculta, es decir su valor fue desconocido por nosotros. El valor fue conocido hasta después de haber obtenido los bits ya mostrados de la cadena binaria, el valor del exponente esta dado por $d = 153253$ y su representación binaria esta dada por $d = 100101011010100101$, comparando la cadena binaria original de d y los bits obtenidos por medio de nuestro ataque, podemos observar que los valores obtenidos son correctos.

4.2.4. Contramedidas para proteger al algoritmo de Da-Zhi.

Con la intención de proteger el algoritmo de Da-Zhi se ha utilizado la ténica de atomicidad propuesta en [15].

Atomicidad en el algoritmo de Da-Zhi.

Buscando trabajar con todos los registros utilizados en el algoritmo 8, $C_0 ... C_3$, para así evitar que este algoritmo utilice operaciones dummy, se utilizó la técnica de atomicidad propuesta en [15]. Como ya se menciono, esta técnica busca que cada iteración del algoritmo sea igual a todas las demás en tiempo y consumo de potencia. Se modifico dicha técnica para hacerla funcional en el algoritmo 8, obteniendo de esta forma el algoritmo 16, donde $\oplus$ representa una operación binaria OR Exclusiva.

Algoritmo 16 Algoritmo de Da-zhi con atomicidad.

1: **Entrada** m, d, N con $d_1 = d_{\lceil n/2 \rceil,1} \cdots d_{1,1}$ y $d_2 = d_{\lceil n/2 \rceil,2} \cdots d_{1,2}$
2: **Salida** $C = m^d \ mod \ N$
3: $C_0 = m$
4: $C_1 = C_2 = C_3 = 1$;
5: $i = k = 1$;
6: **while** $i \leq \lceil i/2 \rceil$ **do**
7: $\quad k = k \oplus [d_{i,1} \ OR \ d_{i,2}]$;
8: $\quad C_{2kd_{i,2}+kd_{i,1}} = C_0 \cdot C_{2kd_{i,2}+kd_{i,1}}$;
9: $\quad i = i + k$;
10: **end while**
11: $C_1 = C_1 \cdot C_3$
12: $C_2 = C_2 \cdot C_3$
13: $C = sq^{(\lceil n/2 \rceil)}(C_1) \cdot C_2$

El algoritmo 16, utiliza todos los registros para realizar el cálculo del resultado correcto de la exponenciación, es decir, no importando que registro se altere, el resultado final siempre será incorrecto. Esto es una ventaja en comparación al algoritmo original, ya que si en el algoritmo original un fault era colocado en C_0, el resultado final no se veía alterado. Se puede observar que el registro C_0 se ejecuta en cada una de las iteraciones del algoritmo 16, lo cual es una característica importante de este.

Al realizarle pruebas de seguridad con respecto al SJ, nos dimos cuenta que esta mejoría contra un FA común, permitía un ataque por medio del cálculo del SJ. Este ataque se debe a que si un fault es colocado en C_0 cuando $d_{i,1} = d_{i,2} = 0$, la siguiente operación en la iteración $i + 1$ del algoritmo es $C_0 = C_0 \cdot C_0$, por lo tanto, si el valor erróneo tenia un SJ igual a -1 en i, en la iteración $i + 1$ dicho SJ siempre se convertirá en 1. Con las otra tres combinaciones de $d_{i,1}$ y $d_{i,2}$ no pasa esto, ya que el valor C_0 con SJ igual a -1 siempre alterara uno de los tres registros en el siguiente paso, tal que $C_{2kd_{i,2}+kd_{i,1}} = C_0 \cdot C_{2kd_{i,2}+kd_{i,1}}$ (donde $2kd_{i,2} + kd_{i,1} \neq 0$, para este caso). Sabiendo este dato, podemos determinar que si un fault se colocó cuando $d_{i,1} = d_{i,2} = 0$, el algoritmo siempre dará como resultado un SJ igual a 1 en el valor de salida. Para ejemplificar este ataque podemos observar la tabla 4.7, donde tenemos que $(m/N) = 1$, exponente $d = 2834761$ y las columnas tienen el formato $d_{i,1}d_{i,2}$.

01	**00**	**00**	**11**	**00**	**10**	**11**	**00**	**11**	**00**	**10**
0	1	1	-1	1	1	1	1	-1	1	1
0	1	1	-1	1	1	-1	1	1	1	1
0	1	1	1	1	1	-1	1	1	1	1
0	1	1	-1	1	1	-1	1	-1	1	1
0	1	1	1	1	1	1	1	-1	1	1
0	1	1	1	1	1	1	1	-1	1	1
0	1	1	-1	1	1	-1	1	1	1	1
0	1	1	-1	1	1	1	1	1	1	1
0	1	1	-1	1	1	1	1	1	1	1

Tabla 4.7: SJ de los valores de salida, cuando han sido colocados faults en el algoritmo 16.

Segunda Versión.

Como podemos observar, el algoritmo 16 contiene un hoyo en su seguridad debido a que el valor de C_0, cuando $d_{i,1} = d_{i,2} = 0$, se vuelve a multiplicar por si mismo, eliminando todo tipo de SJ igual a -1. Por lo que nuestro siguiente intento por proteger el algoritmo 8 estuvo enfocado en hacer que el SJ de C_0 siempre afecte a un registro diferente de C_0 y así el SJ siempre afecte el cálculo del resultado final. Con esta idea se realizó el algoritmo 17.

Algoritmo 17 Algoritmo de Da-zhi con atomicidad, segunda versión.

Entrada m, d, N con $d_1 = d_{\lceil n/2 \rceil,1} \cdots d_{1,1}$ y $d_2 = d_{\lceil n/2 \rceil,2} \cdots d_{1,2}$
Salida $C = m^d \ mod \ N$
$C_0 = m$
$C_1 = C_2 = C_3 = C_4 = 1$;
$i = k = b = 0$;
while $i \leq \lceil i/2 \rceil$ **do**
    $k = \bar{k} \ AND \ (b \oplus [d_{i,1} \ OR \ d_{i,2}])$;
    $b = k \ OR \ \bar{b}$;
    $C_{3k+b-2kd_{i,2}-kd_{i,1}} = C_0 \cdot C_{3k+b-2kd_{i,2}-kd_{i,1}}$;
    $i = i + k$;
end while
$C_1 = C_1 \cdot C_4^{-1}$;
$C_3 = C_3 \cdot C_1$
$C_2 = C_2 \cdot C_1$
$C = sq^{(\lceil n/2 \rceil)}(C_3) \cdot C_2$

El algoritmo 17, cuenta con un mayor número de registros, incluyendo el registro C_4. Cuando un fault es colocado en C_0, el SJ de este registro siempre alterara el cálculo del resultado final, para lograr esta alteración, C_0 multiplica a C_1 y a C_4 cuando $d_{i,1} = d_{i,2} = 0$,

por lo tanto, C_0 no se multiplica más por si mismo como en el algoritmo 16. Estas multiplicaciones por C_1 y C_4 permiten que un determinado SJ en C_0 siempre afecten toda la ejecución del algoritmo. Como observación, se tiene que el registro C_4 es utilizado para almacenar un valor inverso de C_0, lo que eliminara el valor C_0 de C_1 en la línea 12 del algoritmo 17, también podemos observar que solamente un registro es protegido con el valor de C_0 y no fue necesario proteger todos los registros restantes.

Una situación importante, es que la selección de los registros es diferente en comparación con los algoritmos 8 y 16. En este algoritmo, cuando $d_{i,1} = d_{i,2} = 0$, se usan los registros C_1 y C_4; cuando $d_{i,1} = 1$ y $d_{i,2} = 0$, se utiliza el registro C_3; cuando $d_{i,1} = 0$ y $d_{i,2} = 1$, se utiliza el registro $C2$; y por último, cuando $d_{i,1} = 1$ y $d_{i,2} = 1$, se utiliza el registro C_1. C_0 es utilizado en todas las combinaciones de bits, primero se usa C_0 y después se usa el registro correspondiente a la combinación.

Este algoritmo es un poco más complejo que el anterior y el original, el primer paso para su diseño fue el determinar dos expresiones para calcular b y k, con la finalidad de que el algoritmo realice tres multiplicaciones cuando $d_{i,1} = d_{i,2} = 0$ y solo dos multiplicaciones para cualquier otro caso. Ambas formulas fueron obtenidas con la ayuda de *mapas de Karnaugh*.

En la tabla 4.8, podemos observar el comportamiento del algoritmo 17 cuando este es atacado por medio del SJ. Podemos observar que cuando $d_{i,1} = 1$ y $d_{i,2} = 0$, el resultado siempre es un JS=1. Este comportamiento se debe a que cuando el registro C_3 es atacado, este siempre se elevara a un valor cuadrado en el paso 15 de dicho algoritmo, por lo tanto todo $SJ = -1$ siempre es eliminado, permaneciendo ese registro indefenso a ataques de este tipo.

01	**00**	**00**	**11**	**00**	**10**	**11**	**00**	**11**	**00**	**10**
-1	-1	1	-1	1	1	1	-1	-1	-1	1
1	-1	-1	-1	1	1	-1	1	1	1	1
1	1	-1	1	-1	1	-1	1	1	-1	1
1	-1	-1	-1	-1	1	-1	1	-1	1	1
-1	1	-1	1	1	1	1	-1	-1	1	1
-1	1	1	1	-1	1	1	-1	-1	-1	1
1	1	-1	-1	-1	1	-1	1	1	-1	1
-1	-1	1	-1	1	1	1	1	1	1	1
-1	1	1	-1	- 1	1	1	-1	1	-1	1

Tabla 4.8: SJ de los valores de salida, cuando han sido colocados faults en el algoritmo 17.

Características de las modificaciones al algoritmo de Sun Da-Zhi.

Las versiones aquí presentadas del algoritmo de Da-Zhi, tienen diferentes características a pesar de que están basados en un mismo esquema original. La tabla 4.9 muestra un resumen de las características de dichos algoritmos.

Algoritmo	**No. de multiplicaciones**	**No. de inversiones**	**No. de variables**
Algoritmo 8	$1{,}5n + 2$	0	5
Algoritmo 16	$1n + (3/8)n + 3 = 1{,}375n + 3$	0	4
Algoritmo 17	$1{,}5n + (1/8)n + 4 = 1{,}625n + 4$	1	6

Tabla 4.9: Características de los algoritmos modificados.

En la tabla 4.9, se consideran multiplicaciones tanto las multiplicaciones como las exponenciaciones cuadradas, en el número de variables no se cuentan d_1, d_2, k, i, m, d y N ya que estas variables son inherentes a la exponenciación modular y no a la implementación de esta.

Como podemos ver en la tabla 4.9, el mejor algoritmo en cuestiones de tiempo y almacenamiento es el algoritmo 16, sin embargo, este algoritmo presenta una vulnerabilidad que debe ser evitada.

Adicionalmente a las mejorías de tiempo y espacio del algoritmo 16 con respecto a los demás algoritmos (incluyendo el algoritmo original), se puede observar que este algoritmo puede mejorar su eficiencia temporal si utilizamos una representación binaria del exponente con signo (SD), llamada *non adjacent form* (NAF) [30], ya que esta forma tiene en promedio $(1/3)n$ bits con valores diferentes de cero, y en este algoritmo entre más valores iguales a cero existan, mejor es su tiempo de ejecución. Por otro lado, el algoritmo 17 es incompatible con este tipo de representación binaria, ya que entre mas valores igual a cero existan, más multiplicaciones serán realizadas por este algoritmo.

Capítulo 5

Contramedida que puede ser implementadas en sistemas criptográficos reales.

En este capítulo, se presenta la técnica llamada *exponenciaciones intermedias pares* (EIP), que sirve para proteger el cálculo de la exponenciación modular contra diferentes ataques físicos, adicionalmente, también se explican las características que hacen de esta contramedida, una opción práctica y eficiente en su implementación.

A grandes rasgos, el método de EIP consiste en eliminar todos los exponentes impares que son calculados a través de las iteraciones de un algoritmo de exponenciación modular, logrando que el algoritmo solo calcule exponenciaciones pares a través de su ejecución. Decimos que son exponenciaciones intermedias, debido a que son las exponenciaciones que el algoritmo realiza para llegar al cálculo final de la exponenciación principal m^d.

5.1. Protección contra el ataque $N-1$.

Si observamos la tabla 4.1, podemos notar que para que se pueda obtener la cadena binaria del exponente, es necesario que existan dos patrones de comportamiento, uno correspondiente al valor $1 \cdot r$ y el otro correspondiente al valor $(N-1) \cdot r$, sin embargo, si todos los patrones fueran iguales, no existiría forma de diferenciar cuando un bit es igual a 0 o cuando es igual a 1.

Los algoritmos de exponenciación, realizan exponenciaciones intermedias pares e impares a través de su ejecución, ambos tipos de exponenciaciones, como se explicó en el capítulo 4, permiten dos patrones observables en una señal de consumo de potencia. Para evitar que ambos patrones existan a través del cálculo de la exponenciación, se han eliminado los exponentes intermedios impares de la ejecución de los algoritmos atacados por el esquema $N-1$, dando como resultado los algoritmos 18 y 19 [31].

Ambos algoritmos, son seguros contra el ataque $N-1$, ya que solo muestran un patrón

Algoritmo 18 Square and multiply always, left-to-right modificado.

Entrada m, d, N, con $d = (d_{n-1} \cdots d_0)_2$
Salida $S = m^d \ mod \ N$
$R[0] = 1$
$M = m \cdot m$
for $n - 1$ to 1 **do**
    $R[0] = R[0]^2 \ mod \ N$
    $R[1] = R[0] \cdot M \ mod \ N$
    $R[0] = R[d_i] \ mod \ N$
end for
if $d_0 = 0$ **then**
    Return $R[0]$
else
    Return $R[0] \cdot m$
end if

Algoritmo 19 BRIP modificado.

Entrada m, d, N, con $d = (d_{n-1} \cdots d_0)_2$
Salida $S = m^d \ mod \ N$
$R[0] = r$
$R[1] = r^{-1}$
$R[2] = m \cdot m \cdot r^{-1}$
for $n - 1$ to 1 **do**
    $R[0] = R[0]^2$
    if $d_i = 0$ **then**
        $R[0] = R[0] \cdot R[1]$
    else
        $R[0] = R[0] \cdot R[2]$
    end if
end for
if $d_0 = 1$ **then**
    $R[0] = R[0] \cdot R[1] \cdot m$
else
    $R[0] = R[0] \cdot R[1]$
end if
Return $R[0]$

observable durante toda su ejecución, obviamente un solo patrón no puede ser utilizado por un atacante, ya que no es posible diferenciar entre un bit con valor igual a 1 y uno con valor igual a 0. Para ejemplificar este funcionamiento, la tabla 5.1 presenta el comportamiento del algoritmo 19, cuando un mensaje igual a $N-1$ es ingresado.

i	d_i	**Pasos intermedios**	Resultado (Patron)
6	1	$R[0] = r^2R[0] = m^2 \cdot r$	$(1) \cdot r$
5	0	$R[0] = m^4 \cdot r^2R[0] = m^4 \cdot r$	$(1) \cdot r$
4	1	$R[0] = m^8 \cdot r^2R[0] = m^{10} \cdot r$	$(1) \cdot r$
3	1	$R[0] = m^{20} \cdot r^2R[0] = m^{22} \cdot r$	$(1) \cdot r$
2	0	$R[0] = m^{44} \cdot r^2R[0] = m^{44} \cdot r$	$(1) \cdot r$
1	0	$R[0] = m^{88} \cdot r^2R[0] = m^{88} \cdot r$	$(1) \cdot r$
0	1	If $d_0 = 1$, $R[0] = R[0] \cdot R[1] \cdot m = m^{88} \cdot r \cdot r^{-1} \cdot m = m^{89}$.	

Tabla 5.1: Algoritmo 19 ejecutado con un mensaje de entrada igual a $N-1$ y un exponente $d = 89 = 1011001$.

Podemos hacer la comparación entre las tablas 4.1 y 5.1, para observar la diferencia entre el comportamiento de un algoritmo que no esta protegido con la técnica de las EIP y uno que si esta protegido con esta técnica.

5.1.1. Contramedidas existentes contra el ataque $N-1$.

La medida propuesta para evitar el ataque $N-1$, es más eficiente en tiempos de ejecución e implementación práctica que las existentes antes de nuestra propuesta. Las medidas propuestas anteriormente son enumeradas a continuación:

1. *Proteger el mensaje*. El mensaje m es ocultado con un valor aleatorio r, esta idea parece una forma correcta de protección, sin embargo, el algoritmo 11 protege el mensaje con r y aún así permanece vulnerable al ataque $N-1$. La mejor técnica de ocultación de mensaje, bajo este esquema, es la propuesta por *Fumaroli* y *Vigilant* [20], pero esta técnica requiere un registro adicional, y ejecuta n operaciones cuadradas más, donde n es el número de bits del exponente.

2. *Proteger el modulo*, mencionada en [32] sección 3.2. El modulo N es protegido con un valor aleatorio r, es decir, se trabaja en modulo $N \cdot r$. Esta idea tiene la desventaja de que al aumentar el tamaño del modulo, los cálculos se realizarán con enteros más grandes, y por lo tanto, el consumo de potencia y el tiempo de ejecución serán mayores a aquellos del algoritmo no protegido, adicionalmente, el tamaño del modulo puede impedir que sea implementado en dispositivos reales, debido a que dichos dispositivos se fabrican basados en estándares ya definidos, por lo que un modulo mayor puede no estar considerado en tales estándares.

3. *Bloquear el mensaje* $N-1$, propuesto en [22]. Esta puede ser la reacción más lógica contra este tipo de ataques, sin embargo, de acuerdo con [22] es posible utilizar

mensajes con un valor $N-1$ modificado, por ejemplo, un atacante puede usar los valores de entrada al algoritmo m_1 y $m_2 = (N-1) \cdot m_1$, el puede obtener las mediciones de consumo de potencia de los cálculos de $m_1^d \ mod \ N$ y $m_2^d \ mod \ N$, entonces el atacante puede detectar las colisiones cuando $m_1^i \equiv m_2^i \ mod \ N$, y con estas observaciones puede deducir el exponente d. Una posibilidad de evitar este ataque modificado, es mediante la comparación de los valores de entrada $m_2 \equiv m_1 \cdot (N-1) \ mod \ N$ para cada par 1 y 2, pero realizar esta comparación entre todos los mensajes de entrada es extremadamente difícil e impráctico, debido a que m_1 y m_2 pueden no ser mensajes consecutivos, adicionalmente, no es posible almacenar todos los mensajes de entrada para poder realizar la comparación.

5.1.2. Ventajas de la contramedida propuesta en comparación con las contramedidas existentes.

Ya teniendo las posibles contramedidas contra el ataque $N-1$ y los motivos por los cuales son imprácticas, ahora enumeramos las razones por las que la contramedida propuesta es mejor:

1. Los algoritmos propuestos tienen el mismo tiempo de ejecución con respecto a las versiones originales, solamente necesitan una condicional *if* extra.

2. Con la contramedida propuesta no es necesario ocultar el mensaje, debido a que en cada paso intermedio del algoritmo, el valor resultante siempre será igual a 1 (si el valor de entrada es $N-1$) sin importar el valor del bit que se este procesando, y por lo tanto, el atacante observará un solo patrón en las mediciones de potencia obtenidas y no será capaz de diferenciar las posiciones donde los bits sean 1 o 0.

3. Los algoritmos propuestos no incrementan el tamaño del modulo ni de cualquier otro valor, por lo tanto, están en concordancia con los estándares de fabricación de los criptodispositivos.

4. Dentro de un dispositivo electrónico, los algoritmos protegidos con la técnica IEP no requieren un mayor consumo de potencia que los algoritmos no protegidos.

5. La contramedida propuesta es útil para evitar ataques $N-1$ simples y para evitar ataques con un valor $N-1$ modificado, esto debido a que si existen ataques con valores $N-1$ modificados, la igualdad $m_1^i \cdot 1 = m_2^i$ siempre será cierta en cada paso del algoritmo y no habrá colisiones que puedan ser utilizadas por un atacante, por lo que no es necesario revisar la relación que existe entre dos diferentes mensajes de entrada, lo cual es muy costoso e impráctico.

5.2. Explicación de la contramedida propuesta.

Los algoritmos 5 y 11 se encuentran en la forma left-to-right. Esta forma de ejecución esta basada en que, dado el resultado de m^a, es posible obtener $m^{a'}$, donde a' es calculada por medio de la adición del bit b a a; se puede notar que $a' = 2a + b$, y por consiguiente

$$m^{a'} = m^{2a+b} = (m^a)^2 \cdot m^b \tag{5.1}$$

Si la ecuación (5.1) es usada para calcular m^d en un algoritmo de exponenciación, donde $d = \sum_{i=0}^{n-1} d_i \cdot 2^i$, la ecuación puede ser representada de la siguiente manera

$$(\cdots((m^{d_{n-1}})^2 \cdot m^{d_{n-2}})^2 \cdots m^{d_1})^2 \cdot m^{d_0} \tag{5.2}$$

donde n es el número de bits del exponente siendo procesado.

Como se mencionó anteriormente, la idea principal se basa en que las exponenciaciones intermedias dentro del algoritmo sean siempre pares. Ahora bien, la ecuación (5.3), podría ser la representación de los algoritmos que solo tienen exponentes intermedios pares, sin embargo, es fácil observar que la ecuación (5.3) no calcula el valor correcto de m^d, mientras que la ecuación (5.2) si calcula el valor correcto de m^d.

$$(\cdots((m^{2d_{n-1}})^2 \cdot m^{2d_{n-2}})^2 \cdots m^{2d_1})^2 \cdot m^{2d_0} \tag{5.3}$$

Para los fines deseados, se busco la manera de obtener una ecuación que definiera el funcionamiento de un algoritmo de exponenciación que solo utiliza exponentes intermedios pares en su ejecución, por lo que la idea principal fue que pudiera dar el resultado correcto de m^d, pero a partir del comportamiento de la ecuación 5.3.

Con la finalidad de obtener la representación de los algoritmos que solo usan exponentes intermedios pares, vamos a desglosar el comportamiento de las ecuaciones (5.2) y (5.3) a través de las iteraciones de un algoritmo de exponenciación. El comportamiento de ambas ecuaciones, será presentado asumiendo que ambas son ejecutadas desde el bit d_{n-1} al bit d_0, $k = n - 1 - i$ en ambas representaciones y cada *step* representa una iteración del algoritmo.

El comportamiento de la ecuación (5.2) a través de un algoritmo, esta dado por las ecuaciones (5.4) a (5.7):

$$Step\ 1 \qquad (\ldots((m^{2^1(d_{n-1})+2^0(d_{n-2})})^2 \cdot m^{(d_{n-3})})^2 \cdots m^{(d_1)})^2 \cdot m^{(d_0)} \tag{5.4}$$

$$Step\ i \qquad (\ldots((m^{2^i(d_{n-1})+2^{i-1}(d_{n-2})+\cdots+2^0(d_k)})^2 \cdot m^{(d_{k-1})})^2 \cdots m^{(d_1)})^2 \cdot m^{(d_0)} \tag{5.5}$$

$$Step\ n-2 \qquad (m^{2^{n-2}(d_{n-1})+2^{n-3}(d_{n-2})+\cdots+2^0(d_1)})^2 \cdot m^{(d_0)} \tag{5.6}$$

$$Step\ n-1 \qquad m^{2^{n-1}(d_{n-1})+2^{n-2}(d_{n-2})+\cdots+2^1(d_1)} \cdot m^{(d_0)} \tag{5.7}$$

Ahora, el comportamiento de la ecuación (5.3) esta dado por las ecuaciones (5.8) a (5.11):

$$Step\ 1 \qquad (\ldots((m^{2^{1+1}(d_{n-1})+2^{0+1}(d_{n-2})})^2 \cdot m^{2(d_{n-3})})^2 \cdots m^{2(d_1)})^2 \cdot m^{2(d_0)} \tag{5.8}$$

$$\textit{Step } i \qquad (\ldots((m^{2^{i+1}(d_{n-1})+2^{i-1+1}(d_{n-2})+\cdots+2^{0+1}(d_k)})^2 \cdot m^{2(d_{k-1})})^2 \cdots m^{2(d_1)})^2 \cdot m^{2(d_0)} \tag{5.9}$$

$$\textit{Step } n-2 \qquad (\underline{m^{2^{n-2+1}(d_{n-1})+2^{n-3+1}(d_{n-2})+\cdots+2^{0+1}(d_1)}})^2 \cdot m^{2(d_0)} \tag{5.10}$$

$$\textit{Step } n-1 \qquad m^{2^{n-1+1}(d_{n-1})+2^{n-2+1}(d_{n-2})+\cdots+2^{1+1}(d_1)} \cdot m^{2(d_0)} \tag{5.11}$$

Es fácil ver que la ecuación (5.7) es el valor correcto de m^d. Ahora, podemos observar que la parte subrayada de la ecuación (5.10) ($\underline{m^{2^{n-1}(d_{n-1})+2^{n-2}(d_{n-2})+\cdots+2^1(d_1)}}$) es muy similar a la ecuación (5.7).

Para obtener igualdad entre las ecuaciones (5.7) y (5.10), primero que nada, es necesario eliminar la última exponenciación cuadrada y la última multiplicación por $m^{2(d_0)}$ en la ecuación (5.10), lo que nos da la siguiente expresión:

$$m^{2^{n-1}(d_{n-1})+2^{n-2}(d_{n-2})+\cdots+2^1(d_1)} \tag{5.12}$$

Ahora, la ecuación (5.12) debe ser multiplicada por $m^{(d_0)}$, y de esta forma el valor correcto de m^d se obtiene a partir del comportamiento de la ecuación (5.3).

Tomando en cuenta la explicación dada, los algoritmos 18 y 19 calculan el valor correcto de m^d debido a que se ejecutan desde $n-1$ a 1, así eliminando la última exponenciación cuadrada y la última multiplicación por $m^{2(d_0)}$, y aparte usa la condicional *if*, la cual es la multiplicación por $m^{(d_0)}$, donde $d_0 \in \{0, 1\}$.

Este comportamiento se puede reducir a la ecuación (5.13), la cual representa el comportamiento de algoritmos de exponenciación modular que solo tienen exponentes intermedios pares a través de su ejecución.

$$(\cdots((m^{2d_{n-1}})^2 \cdot m^{2d_{n-2}})^2 \cdots m^{2d_2})^2 \cdot m^{2d_1} \cdot m^{d_0} \tag{5.13}$$

5.3. Algoritmo 10 protegido contra el ataque propuesto por Schmidt.

El ataque propuesto por Schmidt, también utiliza las exponenciaciones intermedias para lograr su finalidad, en este ataque es necesario que existan tanto exponentes intermedios pares como impares, ya que solo de esta forma se puede hacer una diferenciación en cuanto a los valores de los bits procesados, esto se puede observar en la tabla 4.2.

Ahora bien, si como ya se menciono, es necesario que existan tanto exponentes pares como impares para que el ataque funcione, la solución más lógica para evitar estos ataques, es eliminar la existencia de uno de los dos tipos de exponentes intermedios a través de la ejecución del algoritmo. Y esa es la idea básica para proteger al esquema FV contra este

ataque.

Como se puede observar, la formulación de este ataque es análoga al ataque $N-1$, por lo que, la solución para este problema es una solución análoga a la del ataque $N-1$. En base a esta característica, se modificó el algoritmo 10 en concordancia con la ecuación (5.13), de tal manera que a través de cada una de las iteraciones del ciclo *for*, solo existan exponentes intermedios pares, el algoritmo resultante esta dado como algoritmo 20 [33].

Algoritmo 20 Algoritmo FV modificado.

Input $m \in \mathbb{G},\ d = (d_{n-1}, \ldots, d_0)_2$
Output $s = m^d \in \mathbb{G}$
$R[0] \leftarrow r$
$R[1] \leftarrow m^2 \cdot r$
$R[2] \leftarrow r^{-1}$
for $n-1$ to 1 **do**
    $R[\bar{d}_i] \leftarrow R[\bar{d}_i] \cdot R[d_i]\ mod\ N$
    $R[d_i] \leftarrow R[d_i] \cdot R[d_i]\ mod\ N$
    $R[2] \leftarrow R[2] \cdot R[2]$
end for
if $d_0 = 1$ **then**
    $R[0] = R[0] \cdot m$
end if
Return $R[0] \cdot R[2]$

Como podemos ver, el algoritmo 20 realiza, en promedio, la misma cantidad de multiplicaciones que el algoritmo 10, lo que significa que no sacrifica tiempo de ejecución ni utiliza espacio de almacenamiento adicional, el único requerimiento extra, en comparación con el algoritmo original, es una condicional *if*.

Implementando la técnica de las EIP sobre el algoritmo FV, podemos evitar el FA propuesto por Schmidt, de una forma sencilla y no costosa en su implementación real. El algoritmo 20, calcula todas sus operaciones con un SJ igual a 1, por lo que no importa si una o más operaciones son saltadas, el SJ permanecerá siendo siempre igual a 1 en todos y cada uno de los registros, evitando así, que un atacante pueda utilizar este concepto para poder romper la seguridad del criptosistema.

Con el algoritmo 20, el atacante no va a obtener información que le pueda indicar donde $d_i = d_{i+1}$ y donde $d_i \neq d_{i+1}$, ya que el valor de salida siempre tendrá un SJ igual a 1 si $(m/N) = 1$ y un SJ igual a -1 si $(m/N) = -1$, estos valores serán dados sin importar que iteración fue atacada.

5.4. Comentarios acerca de las exponenciaciones intermedias pares.

Como se ha mencionado, esta técnica permite realizar el cálculo de exponenciación modular, logrando que el número de multiplicaciones y el tiempo de ejecución permanezcan siendo iguales con respecto a los valores que las versiones originales de los algoritmos implementados con este método presentan, adicionalmente, el consumo de potencia también se ve inalterado con respecto a los algoritmos originales. Estas características, permiten deducir que esta técnica puede ser implementada en sistemas criptográficos reales, ya que, la eficiencia de los algoritmos no se ve alterada y se tiene un grado de seguridad mayor.

Un punto interesante de mencionar acerca de esta técnica, es el hecho de que sirve para evitar tanto un SCA como un FA, esto es interesante debido a que, generalmente, se buscan contramedidas contra SCA por un lado y contra FA por otro, de forma independiente, y en este caso la misma técnica sirve para casos específicos de ambos tipos de ataques.

La tabla 5.2, sirve para dar una idea general de algunas de las características de los algoritmos presentados en este capítulo con respecto a otros algoritmos de exponenciación, en esta tabla pueden observarse aspectos, tales como el número de registros y número de multiplicaciones, que pueden ser considerados para su implementación en sistemas reales con capacidades pequeñas de almacenamiento y procesamiento. En esta tabla, se consideran como multiplicaciones tanto las exponenciaciones cuadradas como las multiplicaciones, para determinar el número de registros no se toman en cuenta las variables correspondientes a m, d y N.

Algoritmo	**No. de registros**	**No. de multiplicaciones**	**Inverso**
SaM LtR	1	$1{,}5n$	No
SaM RtL	2	$1{,}5n$	No
SaMA	2	$2n$	No
MPL	2	$2n$	No
KQ	3+1	$2n+2$	Si
FV	3	$3n+2$	Si
BRIP	3	$2n+2$	Si
Da-Zhi	5	$1{,}5n+4$	No
SaMA propuesto.	3	$2n$	No
BRIP propuesto.	3	$2n+2$	Si
FV propuesto.	3	$3n+1$	Si.

Tabla 5.2: Caracteísticas de los algoritmos propuestos en comparación con otros algoritmos de interes.

En el caso del algoritmo KQ, en la tabla 5.2, se cuenta un registro adicional correspondiente al valor inverso, este registro se cuenta debido a que al final del algoritmo, el valor a es requerido explícitamente, por otro lado, los otros algoritmos que también requieren un valor inverso, guardan desde el principio dicho valor en uno de los registros principales

y no se vuelve a requerir el valor inverso de forma especifica a través de la ejecución del algoritmo, por eso no se enumeran en la tabla.

Capítulo 6

Comentarios y conclusiones.

Son varias y variadas, las conclusiones a las cuales se llego después de haber realizado este trabajo. Estas conclusiones, están enfocadas en gran medida a la forma de diseñar y desarrollar un algoritmo utilizado en el área de seguridad. En el presente escrito, se ha llegado a conclusiones muy particulares referentes a la exponenciación modular, sin embargo, estas conclusiones también pueden ser generalizadas a sistemas criptográficos completos, es decir aquel sistema donde se involucran una gran cantidad de procesos y entornos para su diseño y ejecución.

Retomando un poco lo dicho en el párrafo anterior, los sistemas de seguridad pueden ser vistos como entes grandes y complejos, y tal vez lo sean. Estos entes se rigen bajo ciertas normas de "conducta", por lo que el diseño de estos sistemas debe estar sujeto a dichas normas, tales como la funcionalidad, la garantía de la seguridad y la sencillez en su uso e implementación. Un aspecto importante e interesante de estos sistemas completos de seguridad, es que están divididos en varios procesos individuales y cada uno de ellos debe a su vez, cumplir las mismas normas conductuales que el sistema completo, para que el ente funcione de la forma requerida.

La técnica de las exponenciaciones intermedias pares, es una técnica, que como ya explicamos, puede evitar tanto un tipo de fault attack como un tipo de side channel attack. Lo interesante de esta propuesta, es que se puede implementar de una forma sencilla y práctica en dispositivos reales, debido a que no requiere recursos extra, temporales ni espaciales. Es decir, un dispositivo electrónico que ejecute un criptosistema con un algoritmo de exponenciación modular no protegido contra ataques $N - 1$ o el ataque de Schmidt, puede fácilmente ejecutar un criptosistema con algoritmos que utilicen la técnica de las exponenciaciones intermedias pares, ya que no es necesario aumentar el número de registros al diseño electrónico, ni es necesario realizar cálculos en un mayor tiempo. Desde este punto de vista, también podemos agregar que este método no genera una mayor eficiencia en cuanto al tiempo de ejecución con respecto a los algoritmos no protegidos, pero tampoco genera una disminución de la eficiencia temporal.

Un punto importante de mencionar, es la cuestión práctica de los algoritmos diseñados. En ocasiones, se dan soluciones que, aunque teóricamente son correctas, no son correctas

de una forma práctica, como ya se hizo notar en el capítulo 5. El problema de la situación práctica, proviene del hecho de que estamos trabajando con dispositivos que no cuentan con grandes recursos de almacenamiento ni de procesamiento, es decir, si estos dispositivos son diseñados y fabricados para trabajar con valores de una determinada cantidad de bits en su forma binaria, estos aparatos electrónicos no podrán realizar cálculos que impliquen valores más grandes.

Debido al hecho mencionado en el párrafo anterior, es necesario diseñar algoritmos que cumplan con los estándares existentes en la fabricación de dispositivos. Bajo este esquema, la técnica de las EIP ha sido diseñada, es decir, cualquier algoritmo que sea protegido utilizando esta técnica, va a estar en concordancia con dichos estándares, ya que no es necesario realizar ni cálculos más grandes ni más movimientos que los que los algoritmos no protegidos realizan.

Un aspecto ya anteriormente mencionado en [25], y retomado en esta trabajo en forma particular, es el hecho de que "todo algoritmo de seguridad si no es debidamente abordado, puede provocar una nueva vulnerabilidad", esta afirmación la pudimos comprobar en el ataque propuesto contra el algoritmo de Sun Da-zhi en el capítulo 4. Dicho algoritmo, fue un algoritmo innovador e interesante, también fue presentado como un algoritmo seguro, sin embargo, las mismas características que lo hacían un algoritmo "seguro" y eficiente, dieron pauta para poder obtener información relacionada a la clave privada del criptosistema, es decir, para asegurar la regularidad del algoritmo fue necesario recurrir a una operación dummy, que fue explotada por nosotros, y también, la selección de los registros era condicionada por los valores de las dos cadenas binarias que utilizaba, lo cual también fue utilizado por nosotros para obtener información.

Como ya se ha hecho mención a lo largo del presente trabajo, es necesario que todo criptosistema (lo cual incluye hardware y software) este protegido contra diversos tipos de ataques, ya que es en estos criptosistemas donde se deposita la confianza para realizar todo tipo de transacciones. Hoy en día, estas transacciones incluyen datos de toda índole, incluyendo personales y financieros, que si no son debidamente protegidos pueden generar consecuencias graves en las formas de comunicación tal como ahora las conocemos.

Apéndice A

Otros ataques y contramedidas.

A lo largo de este trabajo, se ha hablado de SCA que se basan en la medición del consumo de potencia de un dispositivo electrónico cuando ejecuta un algoritmo criptográfico, sin embargo, no es el único ataque del tipo SCA, ya que estos ataques existen en grandes cantidades y formas. No solamente el consumo de potencia de un dispositivo nos puede dar información de los datos protegidos, sino que, cualquier señal emitida por tales dispositivos puede mostrarnos información relevante. Muchos ataques se han llevado a cabo por medio del uso de señales electromagnéticas, térmicas y sonoras generadas por los aparatos electrónicos. Aunado a esto, también existen otro tipo de ataques más agresivos que requieren la destrucción de los pequeños dispositivos para tener éxito, estos últimos ataques no están contemplados en el cuerpo principal de este trabajo, pero como una cuestión de conocimiento, es bueno tener información relacionada a ellos.

También, en el cuerpo principal de este trabajo, se han explicado métodos algorítmicos para proteger dispositivos reales, tales como smart cards, de diferentes ataques físicos, sin embargo, la protección de dichos dispositivos, es un trabajo que debe realizarse en conjunción con técnicas ingenieriles y de diseño de equipo electrónico. Un buen sistema, como lo ha mencionado Ferguson y Schneier, debe de ser seguro en cada una de sus partes, y en este caso, en cada una de las áreas de diseño, tanto algorítmicas como electrónicas.

Existen técnicas de ingeniería que son útiles en la protección de dispositivos electrónicos pequeños, debido a que nuestro trabajo de tesis se centra en la cuestión algorítmica, este apéndice muestra un pequeño resumen de algunas técnicas de diseño electrónico que se han propuesto para evitar SCA y FA, así como también muestra un pequeño compendio de ataques físicos existentes y no mencionados en los capítulos principales.

Esta sección, es presentada para mostrar a un futuro lector, el como se han realizado tanto ataques como protecciones a nivel práctico. Esto es interesante, debido a que es un punto adicional en el diseño de sistemas completos de seguridad.

A.1. Ataques.

A.1.1. Side channel attacks.

Se ha demostrado que toda señal emitida por un dispositivo electrónico, tiene relación con los cálculos realizados dentro de él, de esta forma, la medición, observación y estudio de estas distintas señales puede dar como resultado la ruptura de la seguridad de un sistema de protección de datos. Estos ataques se han realizado en gran medida, sin embargo, solo vamos a mencionar algunos pocos.

En 2001, *Quisquater* y *Samyde* [34] realizaron lo que ellos llamaron "la continuación de las ideas de Kocher", ellos se enfocaron en la medición de campos electromagnéticos que generan los dispositivos electrónicos, a lo cual llamaron *análisis electro magnético* (EMA, por sus siglas en ingles), determinando que podían obtener la misma información a partir de un dispositivo electrónico, tal como un procesador de computadora, que la obtenida por medio de la medición del consumo de potencia, en base a esto, también mencionan que análisis del tipo DPA pueden llevarse a cabo de esta forma, ya que las firmas electromagnéticas contienen la misma información que las señales de consumo de potencia.

Otro tipo de ataque del tipo SCA, es el ataque acústico, esto es, en base a un estudio de los sonidos provocados por equipos de computo cuando realizan operaciones, un atacante puede obtener información relacionada a las operaciones que el sistema esta ejecutando. *Tromer* y *Shamir* [35], han realizado experimentos de este tipo, en donde observaron que era posible distinguir varios patrones de las operaciones ejecutadas por un CPU y operaciones de acceso a memoria. Ellos mencionan, que este comportamiento puede ser observado en casos artificiales, tal como loops de las instrucciones del CPU; y en casos de la vida real, tal como la desencriptación por medio de RSA.

Tromer y *Shamir*, también comunican que para todas las operaciones de un CPU, existe una *firma acústica*. Estas operaciones pueden ser criptográficas o de cualquier otro tipo, y obviamente son señales que pueden ser analizadas, observadas y descifradas.

Por otro lado, *Asonov* y *Agrawal* en 2004 [36] mostraron que los sonidos de teclas de computadora, teléfonos y cajeros automáticos pueden ser reconocidos e identificados, de esta forma, un atacante puede obtener información referente a claves privadas por medio de la simple monitorización de sonidos de un dispositivo electrónico que cuente con un sistema de teclas. Este ataque, estuvo principalmente basado en la suposición de que cada tecla tiene un sonido diferente aunque a oídos de humano suene exactamente igual, este ataque no es invasivo y es barato, ya que solo se requiere un micrófono parabólico para su ejecución. Este experimento se llevo a cabo mediante el uso y entrenamiento de redes neuronales, para que ellas pudieran reconocer los sonidos de las teclas sin importar la velocidad o estilo que cada persona pueda dar al momento de teclear un código alfanumérico. La detección de los sonidos también se pudo lograr colocando el micrófono a metros de distancia del dispositivo atacado.

Un SCA por demás interesante pero que no esta dirigido hacia pequeños criptodispositivos, es el ataque presentado por *Joe Loughry* y *David A. Umphress* [37]. Estos autores, presentaron un análisis relacionado a la información que puede ser obtenida a partir de los LEDs parpadeantes de equipos electrónicos, tales como modems, routers, etc. Ellos se dieron cuenta de que estos LED's parpadean de acuerdo a la información que se esta transmitiendo a través de una conexión de ese tipo, por lo que un atacante puede observar a distancia el parpadeo y deducir los datos que se estén transmitiendo. Es importante hacer notar, que este estudio lo realizaron sobre equipos comerciales, demostrando su utilidad en la práctica.

A.1.2. Fault attacks.

Schmidt y *Hutter* [38] demostraron que varios bits de una memoria SRAM de un microcontrolador, pueden ser alterados por medio de un FA de tipo óptico, es decir, ellos colocaron la luz de un diodo láser, por medio de una guía de luz de fibra óptica, sobre el dispositivo atacado, alterando bits del dispositivo. Dentro de este microcontrolador se ejecuto el algoritmo 12, y con los errores inducidos por medio del haz láser, se generaron valores erróneos al momento de realizar los cálculos correspondientes. En [39] también se llevo a cabo un FA por medio de elementos ópticos, ataque realizado a un bajo costo económico, los autores mencionan que tecnologías como EPROM y EEPROM pueden ser manipuladas por este tipo de ataques.

En [40], se mencionan un FA de tipo térmico, es decir, el circuito funciona correctamente bajo niveles mínimos y máximos de temperatura. El objetivo aquí, es variar la temperatura hasta un punto que está fuera de los límites del dispositivo, para que las celdas de la memoria RAM sean modificadas de forma aleatoria.

También en [40], se menciona la variación en el reloj externo, este ataque puede causar una mala lectura de los datos, ya que el circuito trata de leer un valor del bus de datos antes de que la memoria tenga tiempo de guardar el valor requerido. Esta situación, también puede provocar una perdida de datos, ya que el circuito comienza ejecutando la instrucción $n-1$ antes de que el procesador termine de ejecutar la instrucción n.

Para un buen resumen acerca de FA y algunas contramedidas útiles para evitarlos, es recomendable leer el trabajo [40].

A.1.3. Ataques invasivos.

Este tipo de ataques, tal como su nombre lo indica, son ataques agresivos y destructivos. Estos ataques, pueden destruir completamente el dispositivo que se quiere espiar o vulnerar. En [41], se describe la forma en como poder realizar este tipo de ataques:

1. Primero se separa el chip de la tarjeta plástica. Esto se realiza calentando la tarjeta, hasta que esta sea flexible, y así el chip puede ser retirado.

2. Ahora, se cubre el chip con acido nítrico fumante calentado a 60°C, y esperamos a que la resina negra que encapsula al silicio este completamente disuelta.

3. El chip es lavado con acetona en un baño ultrasónico, después de esto, son removidos los remanentes con pinzas.

4. Ahora, se va a crear un mapa del chip, esto mediante el uso de un microscopio con una cámara CCD, que va a tomar fotografías de alta resolución de la superficie del chip.

5. Estructuras básicas como líneas de dirección y datos, pueden ser identificados por medio del estudio de patrones de conductividad y por el trazo de ciertas líneas que cruzan por módulos conocidos como ROM, RAM, ALU, etc.

6. Las fotografías tomadas, van a mostrar la capa del chip que se encuentra más arriba, pero capas más profundas se pueden reconocer con otra serie de fotografías, después de que las capas han sido removidas, lo cual se hace sumergiendo el chip en acido hidrofluórico en un baño ultrasónico.

7. Ya teniendo todo el mapa de las capas del chip, es posible reconstruir la circuitería y así manipular los accesos a todos los valores de memoria.

Para una descripción más completa de todo este proceso, es necesario revisar la referencia [41], de donde se saco la metodología descrita en la parte superior de este párrafo, y donde es explicada de forma más especifica. También puede ser consultado [42], donde también se da una explicación de como se llevan a cabo este tipo de ataques.

Este tipo de ataques, como se puede suponer, requiere un conocimiento mayor acerca de equipos especializados y diseño electrónico.

A.2. Contramedidas.

En este punto, vamos a resumir algunas contramedidas en la parte del hardware, implementadas sobre dispositivos electrónicos para evitar ataques de tipo físico, tales como SCA y FA.

Al igual que las contramedidas algorítmicas, las contramedidas de diseño electrónico también son ingeniosas y, generalmente, muestran una bella sencillez, ya que en base a soluciones fáciles de implementar y baratas en su construcción, es posible evitar distintos tipos de ataques físicos.

Una de estas técnicas es la propuesta por *Shamir* [43], esta técnica utiliza dos capacitores en carga y descarga para alimentar al procesador, el consumo de potencia de los capacitores es un consumo regular e independiente de los cálculos que se estén realizando dentro del chip. De esta forma, un atacante no podrá ver el consumo de potencia relacionado a

la ejecución del algoritmo, ya que, el consumo de potencia de los capacitores va a estar obstruyendo esas señales. Esta técnica se diseño para su uso en contra de SCA, es una técnica sencilla y solo requiere un circuito con dos capacitores y cuatro diodos para su funcionamiento.

Otro tipo de técnicas utilizadas para evitar ataques del tipo SCA, es el uso de operaciones aleatorias durante la ejecución del algoritmo, estas operaciones serán realizadas por el procesador, de tal manera que generen señales también aleatorias que no puedan ser utilizadas por un atacante. Otra técnica para evitar SCA, es el uso de tecnologías donde los elementos electrónicos usen un consumo de energía fijo por evento, de esta forma, el consumo de potencia va a ser independiente de los datos, por lo que no revelara información confidencial del sistema [44].

Obviamente, un método a nivel hardware para detectar FA, es por medio del uso de sensores, sensores que puedan determinar si los valores nominales del dispositivo han sido rebasados. En [40], son mencionados, como contramedidas a los FA, los detectores de luz, detectores de frecuencia y un *escudo activo*, este último, es una maya metálica que cubre el chip completamente y tiene datos pasando continuamente sobre ella, si hay una desconexión o interrupción de la maya, el chip dejará de funcionar. Para una explicación un poco más detallada de este tipo de escudos, puede revisarse, adicionalmente a la cita ya mencionada, el trabajo [41].

Complementariamente en [40], se dan algunas otras contramedidas de hardware, basadas en la comparación de datos calculados, es decir, en uno de los esquemas propuestos se tienen dos bloques dentro del dispositivo, ambos bloques realizan la misma operación, los resultados de ambos bloques son ingresados a un comparador, si los valores no son iguales, una alerta se envía a un bloque de decisión, para que este bloque tome la reacción a seguir. En ese mismo trabajo, se presentan más propuestas de este tipo, todas ellas fundamentadas en este primer esquema, para más información acerca de estas propuestas, dirigirse al trabajo citado.

Apéndice B

Técnicas para mejorar el tiempo de ejecución.

La velocidad de los algoritmos de exponenciación modular, es de suma importancia en sistemas criptográficos completos, ya que esta es una de las primeras características evaluadas al momento de implementar un sistema de seguridad en un entorno real. La idea central de esta tesis, y por lo tanto los capítulos centrales, está enfocada a la cuestión de seguridad de los algoritmos mencionados, por lo que no se ha dado una mayor mención a las diferentes técnicas empleadas para acelerar la ejecución de los sistemas.

Este apéndice tiene la finalidad de resumir y dar a conocer algunas de las técnicas empleadas en los algoritmos de exponenciación modular para acelerar su tiempo de ejecución, así como también se hace referencia a algunos algoritmos de exponenciación que han utilizado estas técnicas en su diseño. El conocimiento de estas técnicas, ayudara al lector a visualizar, en una forma más general, los distintos procesos de diseño de este tipo de algoritmos, para que estos puedan ser realmente funcionales en su implementación real.

Existen varias formas en que un algoritmo puede ser acelerado: por medio del uso de diferentes representaciones binarias del exponente, por medio de la disminución del *hamming weight* del exponente o por medio del uso de multiplicaciones compartidas. Algunas de esas técnicas serán resumidas en esta sección, haciendo la aclaración que nada de lo expuesto en este apéndice es idea original, todo ha sido retomado y resumido de las referencias y autores que se irán mencionando a lo largo de este apartado.

B.1. Diversas representaciones binarias.

Como ya se ha explicado a lo largo del presente trabajo, en general, los algoritmos de exponenciación trabajan con las representaciones binarias del exponente, y en varios de ellos, mientras mayor sea el hamming weight (HW, es el número de 1's existentes dentro de la cadena binaria), mayor número de multiplicaciones realiza el algoritmo. En promedio, una representación binaria común tiene un HW de $n/2$, por lo que se han buscado

representaciones que puedan reducir el HW de la cadena binaria. Estas representaciones se han enfocado en valores con signos, es decir la cadena va a contener bits con valores -1, 0, 1. Estas representaciones signadas, tienen la desventaja de que no son representaciones únicas de la cadena, por lo que, puede existir más de una forma binaria para representar el mismo valor decimal.

Para evitar esta desventaja de las representaciones signadas, se han buscado representaciones que sean consistentes, tal como la propuesta en [30], denominada *non adjacent form* (NAF). Esta representación, tiene la característica de que solo presenta una forma binaria para cada valor decimal existente. Adicionalmente, esta representación tiene, en promedio, un HW de $n/3$, lo que significa que reduce considerablemente el número de multiplicaciones realizadas por un algoritmo. El único inconveniente que puede tener, es que requiere 1 bit más en su representación, por lo tanto, si la cadena original tenía n bits, esta cadena NAF tendrá $n+1$ bits.

Algoritmo 21 Convierte una cadena binaria a su representación NAF.

1: **Entrada** Cadena binaria de longitud n.
2: **Salida** Cadena binaria NAF de longitud $n+1$
3: n es el número de bits de la cadena binaria
4: $i = 0$
5: **while** $n > 0$ **do**
6: **if** $n \; mod \; 2 = 1$ **then**
7: $cadena\_NAF[i] = 2 - (n \; mod \; 4)$
8: $cadena\_NAF[i] = 0$
9: **end if**
10: $n = (n - cadena\_NAF[i])/2$
11: $i = i + 1$
12: **end while**

Otro algoritmo para representar una cadena binaria en forma canónica (donde dos dígitos consecutivos son diferentes de 0), es el presentado por *Joye* y *Yen* en [45], representado como algoritmo 22.

Una forma distinta de representación binaria fue presentada por *Katsuyuki Okeya* y otros autores en [46]. Esta representación fue nombrada *mutual opposite form* (MOF), y también da como resultado una representación única de cada entero. En esta técnica, los signos de bits adyacentes, diferentes de cero, son opuestos, por ejemplo $\underline{1-}1010\underline{-11}0-1$. En esta representación, el bit más significativo diferente de cero, y el bit menos significativo diferente de cero, tienen valores 1 y -1, respectivamente. Esta técnica esta mostrada como el algoritmo 23.
La tabla B.1, muestra algunos ejemplos de las representaciones binarias vistas en este apartado. En dicha tabla, se puede comparar el HW de las representaciones binarias con signo y sin signo.

Algoritmo 22 Representación canónica de una representación binaria.

Entrada $(d_{n-1}, \cdots, d_0)$.
Salida $(d'_n, d_{n-1}, \cdots, d_0)$.
$j \leftarrow n$
$b \leftarrow 0$
$d_n \leftarrow 0$
for $(n-1$ to $0)$ **do**
  if $(d_{i+1} = d_i)$ **then**
    $d'_j \leftarrow d_i - b$
    while $(j > i+1)$ **do**
      $j \leftarrow j - 1$
      $d'_j \leftarrow 1 - d_i - b$
      $b \leftarrow 1 - b$
    end while
    $b \leftarrow d_i$
    $j = j - 1$
  end if
end for
$d'_j \leftarrow -b$
while $(j > 0)$ **do**
  $j \leftarrow j - 1$
  $d'_j \leftarrow 1 - b$
  $b \leftarrow 1 - b$
end while

Algoritmo 23 Convierte una cadena binaria a su representación MOF.

Entrada $d = d_{n-1}, \cdots, d_0$.
Salida Cadena binaria MOF: $d' = d'_n, \cdots, d'_0$
$d'_n \leftarrow d_{n-1}$
for $(n-1$ to $1)$ **do**
  $d'_i \leftarrow d_{i-1} - d_i$
end for
$d'_0 \leftarrow -d_0$
return $(d'_n, \cdots, d'_0)$

Decimal	Binario	NAF	MOF
35831	1000101111110111	010010-1000000-100-1	1-1001-1100000-1100-1
45671	1011001001100111	10-10-1001010-10100-1	1-110-101-1010-10100-1
56991	1101111010011111	100-1000-101010000-1	10-11000-11-1010000-1
55155	1101011101110011	100-10-1000-100-1010-1	10-11-1100-1100-1010-1
69122	10000111000000010	01000100-1000000010	1-1000100-10000001-10
83451	10100010111111011	01010010-1000000-10-1	1-11-1001-1100000-110-1
98274	10111111111100010	10-1000000000-100010	1-11000000000-1001-10
131071	11111111111111111	100000000000000000-1	10000000000000000-1

Tabla B.1: Ejemplos de representaciones binarias signadas de un valor decimal.

B.2. Técnicas para acelerar los algoritmos de exponenciación.

Existen varias técnicas para acelerar el cálculo de la exponenciación modular, sin embargo, en este apartado solo vamos a mencionar unas cuantas. Estas técnicas, se mencionan con la intención de que el lector conozca varias partes del diseño de algoritmos de exponenciación modular. A continuación se da una pequeña introducción de cada una de estas técnicas de optimización:

1. *Common-multiplicand multiplication*, propuesta en [14]. Consideramos el calculo de $\{X \cdot Y_i | i = 1, 2 \ldots t; t \geq 2\}$. En esta técnica, buscamos calcular una sola vez los valores que son comunes en t distintos cálculos, en vez de calcularlos t veces, para esto se definen las variables: $Y_{common} = AND_{i=1}^{t}\ Y_i$ y $Y_{i,c} = Y_i\ XOR\ Y_{common}$, donde $AND_{i=1}^{t} Y_i = Y_1\ AND\ Y_2\ AND \cdots AND\ Y_t$. De esta forma, Y_i puede ser representada por
$$Y_i = Y_{i,c} + Y_{common}.$$
Por lo que la multiplicación $\{X \cdot Y_i | i = 1, 2 \ldots t; t \geq 2\}$ puede ser calculada de la siguiente manera
$$X \cdot Y_i = X \cdot Y_{i,c} + X \cdot Y_{common}.$$
Esta técnica ha sido la fuente fundamental para desarrollar la técnica $t - folding$.

   Antes de iniciar con la siguiente técnica, vamos a dar un pequeño ejemplo: si se quieren multiplicar los valores $Y_1 = 1453$ y $Y_2 = 1186$ por $X = 1622$, vamos a utilizar esta técnica para acelerar el calculo. Tenemos que: $X = 11001010110$, $Y_1 = 10110101101$ y por último $Y_2 = 10010100010$. Lo primero que realizamos es obtener los bits comunes en todas las cadenas Y_i por medio de la operación AND, en este caso solo tenemos dos cadenas. Es importante mencionar, que las operaciones AND se hacen bit a bit, así obtenemos

$$Y_{common} = Y_1\ AND\ Y_2 = 10110101101 AND 10010100010 = 10010100000$$

el valor decimal de Y_{common} es igual a 1184. Ahora se calcula las cadenas $Y_{1,c}$ y $Y_{2,c}$ de la siguiente manera:

$$Y_{1,c} = Y_1 \oplus Y_{common} = 10110101101 \oplus 10010100000 = 00100001101$$

$$Y_{2,c} = Y_2 \oplus Y_{common} = 10010100010 \oplus 10010100000 = 00000000010$$

Con valores decimales $Y_{1,c} = 269$ y $Y_{2,c} = 2$. Ya teniendo todas las variables calculadas, realizamos la operación

$$X{\cdot}Y_1 = X{\cdot}Y_{1,c}{+}X{\cdot}y_{common} = 11001010110{\cdot}00100001101{+}11001010110{\cdot}10010100000$$
$$X \cdot Y_1 = X \cdot Y_{1,c} + X \cdot y_{common} = 1101010100001011110 + 11101010110111000000$$

$$X \cdot Y_1 = 1000111111011000011110$$

Lo que nos da un resultado decimal de $X \cdot Y_1 = 2356766$. Ahora procedemos a calcular $X \cdot Y_1$:

$$X{\cdot}Y_2 = X{\cdot}Y_{2,c}{+}X{\cdot}y_{common} = 11001010110{\cdot}00000000010{+}11001010110{\cdot}10010100000$$
$$X \cdot Y_2 = X \cdot Y_{2,c} + X \cdot y_{common} = 110010101100 + 11101010110111000000$$

$$X \cdot Y_2 = 11101010110101001101100$$

Lo que nos da como resultado decimal $X \cdot Y_2 = 1923692$, y de esta forma se realizan varias multiplicaciones con un multiplicador común.

2. *T-folding*, propuesta en [47]. Esta técnica busca extraer *Subcadenas* de la cadena binaria principal del exponente d. Para implementar esta técnica, primero se parte d en t mitades, por lo tanto d es dividido en 2^t subcadenas del mismo tamaño, cada subcadena es denotada como d_k para $k = 1, \cdots, 2^t$, y $d = d_{2^t}||d_{2^{t-1}}|| \cdots d_1$ donde $||$ es el operador de concatenación. Por lo tanto

$$x^d = \prod_{i=1}^{2^n} sq^{((i-1)(k/2^n))}(x^{d_i}).$$

De esta forma, son definidas las variables $d_{comm-j} = d_{comm-j+1} = d_j \ AND \ d_{j+1}$ para $j = 1, 3, \ldots, 2^n - 3, 2^n - 1$, y $d_{excl-i} = d_{comm-i} \ XOR \ d_i$ para $i = 1, 2, \ldots, 2^n$. Cada d_i puede ser representada como

$$d_i = d_{comm-i} + d_{excl-i}$$

y por lo tanto $x^{d_j} = x^{d_{comm-j}+d_{excl-j}}$. Esta técnica ha sido implementada en [48].

3. *Performing Complements*, propuesta en [49]. Esta técnica lo que busca es reducir el HW de un entero, y por lo tanto realizar menos operaciones binarias al calcular una exponenciación o una multiplicación. La rapidez de esta técnica esta basada en el hecho de que cuando el HW(Y) es mayor que n, donde n es el número de bits, el HW del complemento será menor a $n/2$. El complemento de Y esta definido por

$$Y = (100\cdots0)_{(n+1)bits} - \overline{Y} - 1$$

donde $\overline{Y} = \overline{y_{n-1}y_{n-2}}\cdots\overline{y_0}$, tenemos que $\overline{y_i} = 0$ si $y_i = 1$, y $\overline{y_1} = 1$ si $y_i = 0$. Esta técnica ha sido implementada en [50, 51].

Para ejemplificar esta técnica, si queremos multiplicar $X = 11$ por $Y = 1454$, primero que nada obtenemos la representación binaria de ambos elementos $X = 1011$ y $Y = 10110101110$. La longitud del elemento más grande es de $n = 11$ bits, pero vemos que su $HW = 7$, es decir, es mayor que $\lfloor 11/2 \rfloor$. Debido a que el HW es mayor que la mitad de n, convertimos Y a su complemento, de tal manera que

$Y = 100000000000 - 01001010001 - 1.$

Con lo que hemos reducido el HW de Y, por lo que solo nos resta multiplicar por X:

$X \cdot Y = (1011)(100000000000 - 01001010001 - 1)$

$X \cdot Y = 101100000000000 - 1100101111011 - 1011 = 11111001111010.$

O lo podemos ve en su representación decimal:

$$11 \cdot 1454 = (11)(2048 - 593 - 1) = 22528 - 6523 - 11 = 15994.$$

Como se puede observar, el resultado es el mismo que si se realizara la multiplicación común pero se ha reducido el numero de operaciones al reducir el HW del operador Y.

Bibliografía

[1] A. Boscher, H. Handschuh, and E. Trichina. Blinded fault resistant exponentiation revisited. pages 3–9, 2009.

[2] J.M. Schmidt and M. Medwed. Fault attacks on the montgomery powering ladder. *Information Security and Cryptology-ICISC 2010*, pages 396–406, 2011.

[3] N. Ferguson and B. Schneier. Practical cryptography. 2003.

[4] Eli Biham, Orr Dunkelman, Sebastiaan Indesteege, Nathan Keller, and Bart Preneel. How to steal cars–a practical attack on keeloq. In *EUROCRYPT*, pages 1–18, 2008.

[5] Peter Wright and Paul Greengrass. *Spycatcher: The candid autobiography of a senior intelligence officer*. Dell, 1988.

[6] Wim Van Eck. Electromagnetic radiation from video display units: an eavesdropping risk? *Computers & Security*, 4(4):269–286, 1985.

[7] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T Manzuri Shalmani. On the power of power analysis in the real world: A complete break of the keeloq code hopping scheme. pages 203–220, 2008.

[8] Flavio D Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter Van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs. Dismantling mifare classic. pages 97–114, 2008.

[9] Omar Choudary. The smart card detective: a hand-held emv interceptor. *University of Cambridge, Computer Laboratory, Darwin College, Cambridge, MPhil Thesis*, 2010.

[10] W. Diffie and M. Hellman. New directions in cryptography. *Transactions on Information Theory*, 22(6):644–654, IEEE, 1976.

[11] RL Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[12] J.S. Coron. Resistance against differential power analysis for elliptic curve cryptosystems. pages 725–725, 1999.

[13] M. Joye and S.M. Yen. The montgomery powering ladder. *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 1–11, 2002.

[14] S.M. Yen and C.S. Laih. Common-multiplicand multiplication and its applications to public key cryptography. *Electronics letters*, 29(17):1583–1584, 1993.

[15] B. Chevallier-Mames, M. Ciet, and M. Joye. Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. *Computers, IEEE Transactions on*, 53(6):760–768, 2004.

[16] D.Z. Sun, J.P. Huai, J.Z. Sun, and Z.F. Cao. An efficient modular exponentiation algorithm against simple power analysis attacks. *Consumer Electronics, IEEE Transactions on*, 53(4):1718–1723, 2007.

[17] P. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. pages 104–113, 1996.

[18] P.C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. *In Wiener, M., Ed.: Advances in Cryptology-CRYPTO 99. Volume 1666 of Lecture Notes in Computer Science*, pages 388–397, Springer, 1999.

[19] C.H. Kim and J.J. Quisquater. How can we overcome both side channel analysis and fault attacks on rsa-crt? pages 21–29, 2007.

[20] G. Fumaroli and D. Vigilant. Blinded fault resistant exponentiation. *Fault Diagnosis and Tolerance in Cryptography*, pages 62–70, 2006.

[21] H. Mamiya, A. Miyaji, and H. Morimoto. Efficient countermeasures against rpa, dpa, and spa. *Cryptographic Hardware and Embedded Systems-CHES 2004*, 3156 of Lecture Notes in Computer Science:343–356, 2004.

[22] S.M. Yen, W.C. Lien, S.J. Moon, and J.C. Ha. Power analysis by exploiting chosen message and internal collisions–vulnerability of checking mechanism for rsa-decryption. *Progress in Cryptology–Mycrypt 2005*, 3715 of Lecture Notes in Computer Science:183–195, 2005.

[23] A. Miyamoto, N. Homma, T. Aoki, and A. Satoh. Enhanced power analysis attack using chosen message against rsa hardware implementations. pages 3282–3285, 2008.

[24] D. Boneh, R. DeMillo, and R. Lipton. On the importance of checking cryptographic protocols for faults. *In Fumy, W., Ed.: Advances in Cryptology-EUROCRYPT 97. Volume 1233 of Lecture Notes in Computer Science*, pages 37–51, Springer, 1997.

[25] S.M. Yen, S. Kim, S. Lim, and S.J. Moon. A countermeasure against one physical cryptanalysis may benefit another attack. *Lecture Notes in Computer Science*, 2288:414–427, 2002.

[26] A. Boscher, R. Naciri, and E. Prouff. Crt rsa algorithm protected against fault attacks. *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems*, pages 229–243, 2007.

[27] C. Kim and J.J. Quisquater. Fault attacks for crt based rsa: New attacks, new results, and new countermeasures. *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems*, pages 215–228, 2007.

[28] M. Boreale. Attacking right-to-left modular exponentiation with timely random faults. *Fault Diagnosis and Tolerance in Cryptography*, 4236 of LNCS:24–35, 2006.

[29] D. Tinoco Varela. Attack against an efficient exponentiation algorithm used in cryptosystems. *Proceedings of the 5th International Conference MSAST 2011 of IMBIC*, pages 196–204, 2012.

[30] G.W. Reitwiesner. Binary arithmetic. *Advances in computers*, 1:231–308, 1960.

[31] D. Tinoco Varela. How to avoid the n-1 attack without costly implementations. *International Journal of Network Security and Its Applications*, 4(4):109–122, 2012.

[32] C. Giraud. An rsa implementation resistant to fault attacks and to simple power analysis. *Computers, IEEE Transactions on*, 55(9):1116–1120, 2006.

[33] D. Tinoco Varela. Blinded montgomery powering ladder protected against the jacobi symbol attack. *International Journal of Security*, 6(3):15–27, 2012.

[34] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. *Smart Card Programming and Security*, pages 200–210, 2001.

[35] E. Tromer A. Shamir. Acoustic cryptanalysis on nosy people and noisy machines. *http://tau.ac.il/ tromer/acoustic/*.

[36] Dmitri Asonov and Rakesh Agrawal. Keyboard acoustic emanations. *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*, pages 3–11, 2004.

[37] Joe Loughry and David A Umphress. Information leakage from optical emanations. *ACM Transactions on Information and System Security (TISSEC)*, 5(3):262–289, 2002.

[38] Jörn-Marc Schmidt and Michael Hutter. Optical and em fault-attacks on crt-based rsa: Concrete results. *Proceedings of the Austrochip*, pages 61–67, 2007.

[39] Sergei P Skorobogatov and Ross J Anderson. Optical fault induction attacks. *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 2–12, 2003.

[40] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006.

[41] Oliver Kömmerling and Markus G Kuhn. Design principles for tamper-resistant smartcard processors. *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, pages 2–2, 1999.

[42] Ross Anderson and Markus Kuhn. Tamper resistance-a cautionary note. *Proceedings of the second Usenix workshop on electronic commerce*, 2:1–11, 1996.

[43] Adi Shamir. Protecting smart cards from passive power analysis with detached power supplies. *Cryptographic Hardware and Embedded Systems-CHES 2000*, pages 71–77, 2000.

[44] Kris Tiri, Moonmoon Akmal, and Ingrid Verbauwhede. A dynamic and differential cmos logic with signal independent power consumption to withstand differential power analysis on smart cards. *Solid-State Circuits Conference, 2002. ESSCIRC 2002. Proceedings of the 28th European*, pages 403–406, 2002.

[45] Marc Joye and Sung-Ming Yen. Optimal left-to-right binary signed-digit recoding. *Computers, IEEE Transactions on*, 49(7):740–748, 2000.

[46] Katsuyuki Okeya, Katja Schmidt-Samoa, Christian Spahn, and Tsuyoshi Takagi. Signed binary representations revisited. *Advances in Cryptology–CRYPTO 2004*, pages 123–139, 2004.

[47] D.C. Lou and C.C. Chang. Fast exponentiation method obtained by folding the exponent in half. *Electronics Letters*, 32(11):984–985, 1996.

[48] D.Z. Sun, Z.F. Cao, and Y. Sun. How to compute modular exponentiation with large operators based on the right-to-left binary algorithm. *Applied mathematics and computation*, 176(1):280–292, 2006.

[49] C.C. Chang, Y.T. Kuo, and C.H. Lin. Fast algorithms for common-multiplicand multiplication and exponentiation by performing complements. pages 807–811, 2003.

[50] C.L. Wu. An efficient common-multiplicand-multiplication method to the montgomery algorithm for speeding up exponentiation. *Information Sciences*, 179(4):410–421, 2009.

[51] C.L. Wu, D.C. Lou, J.C. Lai, and T.J. Chang. Fast modular multi-exponentiation using modified complex arithmetic. *Applied mathematics and computation*, 186(2):1065–1074, 2007.