UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

A SIMPLE POWER ANALYSIS ATTACK ON THE TWOFISH KEY SCHEDULE

Autor: José Javier González Ortiz

Director: Kevin J. Compton

Madrid

Julio 2016

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

A Simple Power Analysis Attack on the TwoFish Key Schedule

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico (2015/16) es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: José Javier González Ortiz Fecha: ……/ ……/ ……

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Kevin J. Compton Fecha: ……/ ……/ ……

Vº Bº del Coordinador de Proyectos

Fdo.: David Contreras Bárcena Fecha: ……/ ……/ ……

pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.
La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- ➢ La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- ➢ La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- ➢ La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- ➢ La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a ……….. de ………………………….. de ………

ACEPTA

Fdo……………………………………………

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

A SIMPLE POWER ANALYSIS ATTACK ON THE TWOFISH KEY SCHEDULE

Autor: José Javier González Ortiz

Director: Kevin J. Compton

Madrid

Julio 2016

A SIMPLE POWER ANALYSIS ATTACK ON THE TWOFISH KEY SCHEDULE

Autor: Gonzalez Ortiz, José Javier
Director: Compton, Kevin J.
Entidad Colaboradora: University of Michigan, EECS Department

RESUMEN

Se describe un ataque de Potencia Simple para la implementación del cifrado *TwoFish*. El ataque es capaz de recuperar de forma inequívoca la clave secreta con altos niveles de ruido superpuesto. Se emplean varias etapas de preprocesado seguidas por una combinación del algoritmo de mínimos cuadrados y una búsqueda óptima de máscara de Hamming.

Palabras clave: *TwoFish*, SPA, Power Attack, Block Cipher, Noise Resistance

1. Introducción

Dada la creciente presencia de los dispositivos embebidos y los sistemas de pago electrónico en la sociedad, resulta importante analizar los protocolos de seguridad empleados por este tipo de tecnologías. La inmensa mayoría de los protocolos criptográficos actuales se apoyan en una serie de desarrollos matemáticos que garantizan la confidencialidad del sistema. Sin embargo, si bien las bases teóricas son importantes, dichos protocolos deben ser implementados de forma adecuada. De lo contrario, existe la posibilidad de que ciertos parámetros físicos se filtren, revelando suficiente información para volver al sistema inseguro.

Este tipo de ataques se denominan Ataques de Canal Lateral (*Side Channel Attacks*) y fueron descritos por primera vez por *Kocher* [Koc96]. *Kocher* presentó un ataque que medía los tiempos de respuesta del sistema para obtener información del mensaje encriptado. Desde entonces, otros parámetros han sido considerados a la hora de debilitar algoritmos criptográficos. Entre estos destacan las mediciones de potencia de los sistemas, las cuales han obtenido multitud de resultados en los últimos años.

El primer ataque de potencia fue presentado por *Kocher et al.* [KJJ99]. En este trabajo se mostró la capacidad de obtener información de los operandos del microprocesador a partir del consumo de potencia de un chip CMOS. Entre los ataques de potencia encontramos los denominados Ataques de Potencia Simple (*Simple Power Attacks (SPA)*), que se centran en las vulnerabilidades específicas del diseño del algoritmo. Estos ataques han tenido especial éxito al aplicarse a dispositivos que carecen fuente de alimentación propia tales como las *smartcards*

A la hora de analizar protocolos criptograficos robustos, modernos y con aplicaciones en plataformas embebidas, es natural analizar los finalistas del concurso AES (Advanced Encryption Standard). Esta competición fue convocada por el Instituto Nacional de Normas y Tecnología (NIST) para reemplazar el ya obsoleto DES (Digital Encryption Standard). El esquema de cifrado ganador, Rijndael, fue establecido como el estándar AES y desde entonces ha sido implementado en millones de dispositivos a nivel mundial. Sin embargo, tanto Rijndael como Serpent (el algoritmo en segundo puesto) han mostrado ser vulnerables a Ataques de Potencia Simple. [VBC05; CTV09]. Este proyecto explora la existencia de un Ataque de Potencia Simple en el protocolo TwoFish, el tercer puesto del concurso AES.

2. Definición del Proyecto

TwoFish fue uno de los cinco finalistas del concurso AES y con su propuesta incluyeron una implementación de 8 bits para *smart cards* y dispositivos embebidos [Tsc]. Análisis previos del protocolo TwoFish, tales como el de *Mirza* y *Murphy* [MM99], muestran que uno de los puntos débiles del esquema de cifrado (al igual que Rijndael y Serpent), es el algoritmo de generación de subclaves. Por ello, el objetivo de este proyecto es desarrollar un Ataque Simple de Potencia para la implementación de 8 bits del algoritmo TwoFish.

Dadas las condiciones en las que se realizan los ataques de potencia es necesario considerar la presencia de un nivel de ruido superimpuesto a las mediciones que vayamos a realizar. Para que el ataque sea aplicable al mundo real, éste deberá ser capaz de filtrar el ruido procedente de los sistemas de medicíon así como otras posibles interferencias.

Inicialmente se explora la capacidad de recuperar la clave secreta dada solamente una lectura de potencia de la ejecución del algoritmo. Obtener la clave secreta es lo más difícil pero lo más útil, ya que no sólo permite recuperar el mensaje actual sino también mensajes pasados y futuros que emplean la misma clave. Una vez desarrollado este sistema se considera la capacidad de emplear varias lecturas de cara a obtener una mejor aproximación de la clave.

3. Descripción del modelo

La investigación realizada por *Messerges et al.* [MDS02] y *Mayer-Sommer* [MS00] ha demostrado que es posible extraer los pesos de Hamming de los operandos intermedios a partir del consumo de potencia del dispositivo que realiza la encriptación. Apoyándonos en esta investigación, el proyecto empleará simulaciones software para extraer las lecturas de potencia de la implementación TwoFish de 8 bits. Se superpone ruido gaussiano para a recrear fielmente las condiciones en las que se realizan las mediciones.

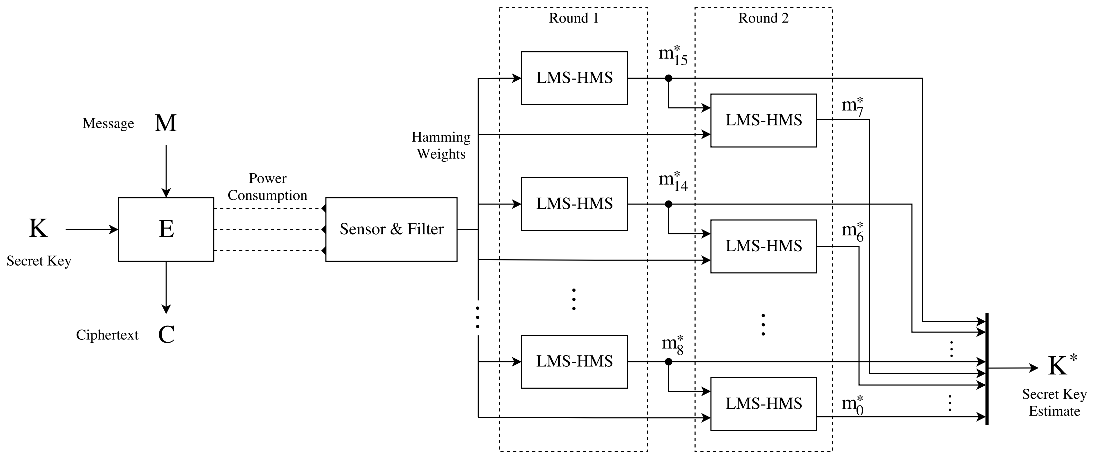


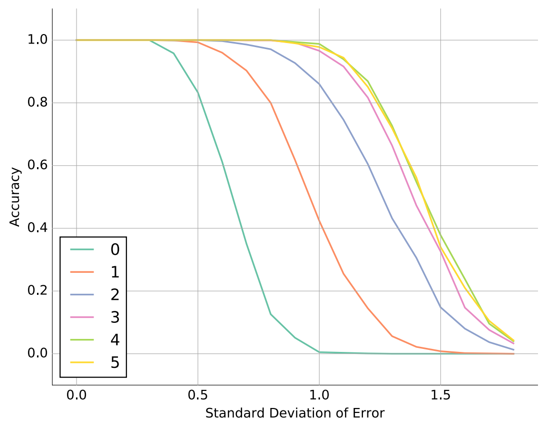
Figura 1: Diagrama de bloques del Ataque de Potencia a TwoFish (Clave 128 bits)

El ataque desarrollado emplea una serie de subsistemas para obtener una aproximación de la clave secreta dada una sola lectura de potencia. Debido a la estructura del algoritmo de generación de subclaves, los bytes de la clave se estiman en varias rondas y empezando por los bytes más significativos. Esto se puede comprobar en la Figura 1. Para estimar cada byte individual se aplica un algoritmo de mínimos cuadrados (LMS) combinado con una búsqueda de Máscara de Hamming (HMS) para poder corregir los errores de medida.

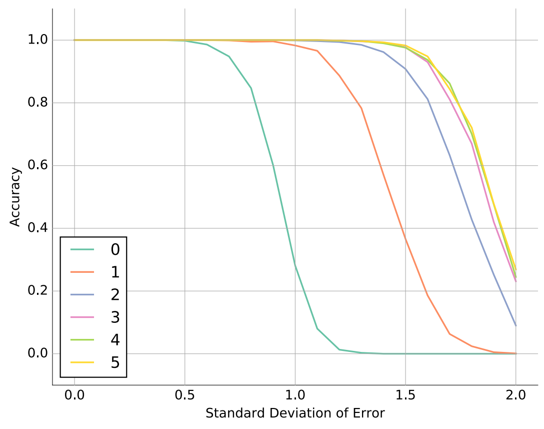
4. Resultados

Se han cumplido los objetivos originalmente establecidos superándose las expectativas. El ataque propuesto es capaz de recuperar la clave secreta cerca al 100 % de las veces con altos niveles de ruido presente. El sistema funciona indiferentemente para los tres tamaños de clave (128, 192 y 256 bits) de la especificación. Adicionalmente, un cuidadoso análisis del resto de implementaciones existentes para el protocolo TwoFish ha revelado que todas ellas son susceptibles al ataque diseñado, mostrando una gran debilidad del esquema de cifrado frente a los ataques de potencia.

Adicionalmente, se ideó un mecanismo para identificar lecturas pertenecientes a la misma clave. Esto permitió combinar múltiples lecturas de potencia incrementando la precisión tal y como queda reflejado en la Figura 2. Las gráficas muestran que para el máximo nivel de corrección se alcanzan tolerancias de 1.0 y 1.5 desviaciones típicas para una y cinco lecturas de potencia respectivamente.



(a) Una Lectura (10^3 claves)



(b) Cinco Lecturas (10^3 claves)

Figura 2: Precisiones medias para varios umbrales de máscara de Hamming (128 bit)

5. Conclusiones

Se ha presentado evidencia de que el protocolo criptográfico TwoFish es susceptible a un Ataque de Canal Lateral basado en los pesos de Hamming de las lecturas de potencia, rechazando la hipótesis de *Biham* y *Shamir* [BS99] de que TwoFish no era vulnerable a los ataques de potencia. TwoFish sube así a tres la cifra de finalistas del concurso AES con SPA conocidos. Esto sienta un precedente importante de cara a la elaboración de nuevos protocolos criptográficos. Éstos no sólo deberán satisfacer garantías teóricas sino que deberán cubrir los posibles puntos débiles de la implementación hardware y software. Con la cada vez más omnipresente figura del *Internet of Things*, la confidencialidad e integridad de los datos recogidos y transmitidos compondrá una de las mayores prioridades.

6. Referencias

[BS99] Eli Biham y Adi Shamir. “Power analysis of the key scheduling of the AES candidates”. En: *Proceedings of the second AES Candidate Conference*. 1999, págs. 115-121.

[CTV09] Kevin J. Compton, Brian Timm y Joel VanLaven. “A Simple Power Analysis Attack on the Serpent Key Schedule”. En: *IACR Cryptology ePrint Archive* 2009 (2009), pág. 473.

[KJJ99] Paul Kocher, Joshua Jaffe y Benjamin Jun. “Differential power analysis”. En: *Advances in Cryptology—CRYPTO’99*. Springer. 1999, págs. 388-397.

[Koc96] Paul C Kocher. “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems”. En: *Advances in Cryptology—CRYPTO’96*. Springer. 1996, págs. 104-113.

[MDS02] Thomas S. Messerges, Ezzat A. Dabbish y Robert H. Sloan. “Examining Smart-Card Security Under the Threat of Power Analysis Attacks”. En: *IEEE Trans. Comput.* 51.5 (mayo de 2002), págs. 541-552.

[MM99] Fauzan Mirza y Sean Murphy. “An observation on the Key Schedule of Twofish”. En: *In the second AES candidate conference, printed by the national institute of standards and technology*. 1999, págs. 22-23.

[MS00] Rita Mayer-Sommer. “Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards”. En: *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*. CHES ’00. London, UK, UK: Springer-Verlag, 2000, págs. 78-92.

[Tsc] *TwoFish Source Code*. `https://www.schneier.com/cryptography/twofish/download.html`. Accessed: 2016-03-01.

[VBC05] Joel VanLaven, Mark Brehob y Kevin J. Compton. “A Computationally Feasible SPA Attack on AES via Optimized Search”. En: *Security and Privacy in the Age of Ubiquitous Computing: 20th International Information Security Conference (SEC 2005)*. 2005, págs. 577-588.

A SIMPLE POWER ANALYSIS ATTACK ON THE TWOFISH KEY SCHEDULE

Author: Gonzalez Ortiz, José Javier
Supervisor: Compton, Kevin J.
Collaborating Entity: University of Michigan, EECS Department

ABSTRACT

This paper introduces an SPA power attack on the implementation of the *TwoFish* block cipher. The attack is able to unequivocally recover the secret key even under substantial amounts of error. The algorithm employs several thresholding preprocessing stages followed by a combination of least mean squares and optimized Hamming mask search.

Keywords: *TwoFish*, SPA, Power Attack, Block Cipher, Noise Resistance

1. Introduction

With the ever increasing relevance of embedded systems and mobile electronic payment it is important to analyze the security of the protocols that can be used in this type of systems. Most cryptographic protocols have quite complex designs in order to maintain secrecy of the information. To accomplish that, they use a number of theoretical constructs that prove security or at least reduce it to solving a computationally hard problem. However, the implementation of the ciphers needs to be appropriate, otherwise, physical parameters of the functioning of such cryptosystem can leak enough information to make it insecure.

Attacks that rely on information gained from the physical implementation of a cryptosystem are called Side Channel Attacks, and were first described by *Kocher* [Koc96]. What *Kocher* introduced was a timing attack that could exploit the response execution times of the system to gain information of the encrypted message. However, since the inception of Side Channel Attacks, other physical measurements have been used to compromise encryption algorithms. Among these, the analysis of the Power Consumption has been one of most successful in terms of compromised ciphers.

The first research power attack is credited to *Kocher et al.* [KJJ99], in which they were able to obtain information of the data manipulated by the processor by carefully measuring the power consumption of a CMOS chip. Within Power Attacks we find what are called Simple Power Attacks, which target particular vulnerabilities of the algorithm's design. These attacks have been proven to be effective against remotely powered devices such as smartcards [MDS02].

When analyzing available, robust, peer reviewed encryption algorithms we can look at the finalists of the Advanced Encryption Standard process. The competition was started by the National Institute of Standards and Technology (NIST) in 1997 in order to designate a successor of the aged Digital Encryption Standard (DES). The winning algorithm, Rijndael, was designated as the AES standard and has been widely used ever since. Nevertheless, recent research efforts have shown that both Rijndael [VBC05] and Serpent [CTV09] (the second finalist) algorithms were susceptible to Simple Power Attacks. This project explores the existence of a Simple Power Attack of another one of the AES finalists, the TwoFish cryptosystem.

2. Project Definition

TwoFish was one of the finalists for the Advanced Encryption Standard process and with its proposal it provided a 8-bit implementation for smart cards [Tsc]. Due to previous analysis showcased by *Mirza* and *Murphy* [MM99], the analysis focuses on the Key schedule of the encryption algorithm. Therefore, the main objective of this investigation project is to construct a Simple Power Attack on the TwoFish Key Schedule for the 8-bit smart card implementation.

Due to the inherent working restrictions of Power Attacks, a significant amount of noise is expected to be superimposed to the signal. For the attack to be relevant and effective it is necessary that it can take into account significant amounts of superimposed noise.

The initial setting explores the possibility of recovering the secret key given a single reading of the power trace of the execution. Obtaining the secret key is the most ambitious goal that a SPA can aim for, since not only uncovers the current message but also compromises past or future messages as well as the integrity of said key.

3. Model Description

The works of *Messerges et al.* [MDS02] and *Mayer-Sommer* [MS00] have proved that it is possible to accurately correlate the hamming weights of the intermediate variables and the power utilization of the encryption system. Thus, the attack will be developed using software simulations of the TwoFish 8-bit power trace. Gaussian variables will be then used to recreate the superimposed noise expected from the power readings.

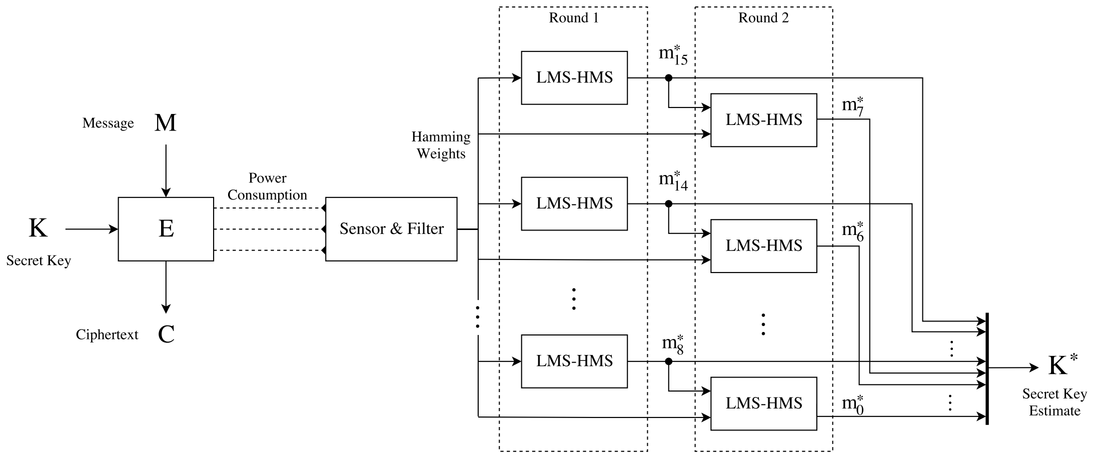


Figure 1: Block diagram of the error noise TwoFish SPA attack for 128-bit key

The TwoFish SPA attack developed uses a series of techniques to get an estimate of the key given the power trace of a single execution of the algorithm. Due to the sequential procedures performed in the key schedule, the bytes of the key are estimated in several rounds as depicted in Figure 1. To estimate each individual byte we first perform a modified Least Mean Square algorithm (LMS). After discretizing the LMS solution, a Hamming

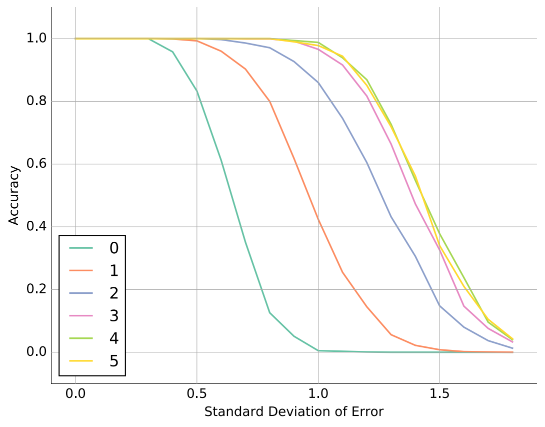
Mask Search (HMS) is applied to correct for possible errors in the solution.

The given design is then used as a building block in order to cluster secret keys given multiple power readings. Readings that are identified to come from the same key can then be combined in order to obtain a more accurate secret key estimate.

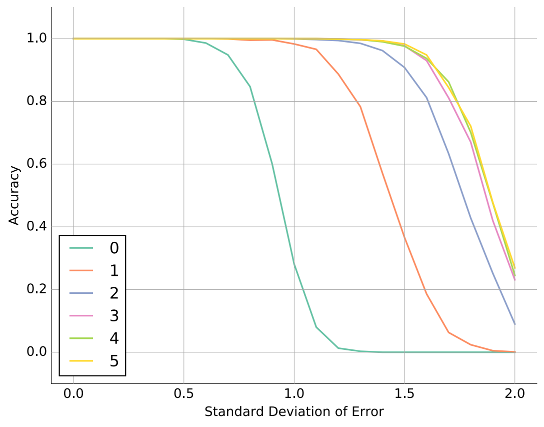
4. Results

The goal of the project was achieved since it was possible to design an noise-resistant algorithm that recovers the secret key with close to 100% accuracy. The algorithm is able to recover all the key sizes given in the TwoFish specification (128, 192 and 256 bit). Careful analysis of the operations performed in the key schedule also revealed that all available TwoFish implementations are susceptible to the described attack, making the cryptosystem highly vulnerable against power attacks.

We were able to identify power traces that were produced by the same key. This produced better accuracies of the estimated keys as we can observe in Figure 2. Here we can notice that if we allow the maximum level of correction, we are able to correct for noise of standard deviation 1.0 and 1.5 for one and five readings respectively.



(a) One reading (10^3 keys)



(b) Five readings (10^3 keys)

Figure 2: Average accuracies 128-bit for different hamming mask sizes

5. Conclusions

We have shown that the TwoFish Encryption Standard is susceptible to a Simple Power Attack that is based in the Hamming weights of the Key Schedule computation, refuting *Biham* and *Shamir* [BS99] assertion that TwoFish was not susceptible to power attacks. With the inclusion of TwoFish, three out of the five AES finalists have been found to be susceptible to SPAs. This sets an relevant precedent for the next generation of cryptographic standards to consider vulnerabilities not only from the theoretical formulation but from the specific implementation of the algorithm. Embedded systems are becoming more and more widespread and with the arrival of the *Internet of Things*, confidentiality and integrity of the information acquired and transmitted by these devices will be an crucial concern.

6. References

[BS99] Eli Biham and Adi Shamir. “Power analysis of the key scheduling of the AES candidates”. In: *Proceedings of the second AES Candidate Conference*. 1999, pp. 115–121.

[CTV09] Kevin J. Compton, Brian Timm, and Joel VanLaven. “A Simple Power Analysis Attack on the Serpent Key Schedule”. In: *IACR Cryptology ePrint Archive* 2009 (2009), p. 473.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. “Differential power analysis”. In: *Advances in Cryptology—CRYPTO’99*. Springer. 1999, pp. 388–397.

[Koc96] Paul C Kocher. “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems”. In: *Advances in Cryptology—CRYPTO’96*. Springer. 1996, pp. 104–113.

[MDS02] Thomas S. Messerges, Ezzat A. Dabbish, and Robert H. Sloan. “Examining Smart-Card Security Under the Threat of Power Analysis Attacks”. In: *IEEE Trans. Comput.* 51.5 (May 2002), pp. 541–552.

[MM99] Fauzan Mirza and Sean Murphy. “An observation on the Key Schedule of Twofish”. In: *In the second AES candidate conference, printed by the national institute of standards and technology*. 1999, pp. 22–23.

[MS00] Rita Mayer-Sommer. “Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards”. In: *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*. CHES ’00. London, UK, UK: Springer-Verlag, 2000, pp. 78–92.

[Tsc] *TwoFish Source Code*. `https://www.schneier.com/cryptography/twofish/download.html`. Accessed: 2016-03-01.

[VBC05] Joel VanLaven, Mark Brehob, and Kevin J. Compton. “A Computationally Feasible SPA Attack on AES via Optimized Search”. In: *Security and Privacy in the Age of Ubiquitous Computing: 20th International Information Security Conference (SEC 2005)*. 2005, pp. 577–588.

"The brick walls are there for a reason. The brick walls are not there to keep us out; the brick walls are there to give us a chance to show how badly we want something. The brick walls are there to stop the people who don't want it badly enough. They are there to stop the other people!"

— RANDY PAUSCH

"It's the questions we can't answer that teach us the most. They teach us how to think. If you give a man an answer, all he gains is a little fact. But give him a question and he'll look for his own answers."

— PATRICK ROTHFUSS

A Helia y Pablo,
por los pilises y las trócolas

Contents

List of Figures

List of Tables

Abbreviations

AES Advanced Encryption Standard. 7, 8, 15–17, 21, 25, 27, 54, 55

DES Digital Encryption Standard. 7, 8, 11, 14, 25

DPA Differential Power Attack. 10, 11, 13–15

HMS Hamming Mask Search. 42, 44, 53, 55, 73

LFSR Linear Feedback Shift Register. 16, 20, 54

LMS Least Mean Squares. 22, 37, 39, 42–44, 53, 73

MDS Maximum Distance Separable. 27

NIST National Institute of Standards and Technology. 8, 21, 25

PHT Pseudo-Hadamard Transform. 11, 27

PRF Pseudo Random Function. 12

PRP Pseudo Random Permutation. 11, 12

RSA Rivest-Shamir-Adleman. 8, 10, 13

SK Secret Key. 7

SPA Simple Power Attack. 10, 11, 13–15, 20–22, 25, 51, 53–55

Nomenclature

C Ciphertext. 6

D Decryption algorithm. 6

E Encryption algorithm. 6

K_i i^{th} subkey in the TwoFish Key Schedule. 27

M Plaintext. 6

N Key size in bits. 25

PK^+ Public Key. 8

PK^- Private Key. 8

R Number of rounds in a TwoFish Key Schedule. $(N/64)$. 27, 29

$\mathrm{H}[x]$ Hamming weight of a binary variable. 31

τ The threshold for the maximum Hamming weight of the applied mask. 38, 39, 44, 47, 48, 51, 73

σ Standard deviation. 35, 73

i Subkey index in the TwoFish key schedule. 27, 29–32

j Byte index of the 32 byte word taken by TwoFish's h function. 29, 30

k Round index in the TwoFish's h transformation. 29, 30

l Composed index to address the key bytes m_l. 30

m_l lth byte of the secret used in the TwoFish Key schedule. XI, 31, 32, 37, 38

v_{ijk} Intermediate byte computation before S-box substitution in TwoFish key schedule. 29

w_{ijk} Intermediate byte computation after S-box substitution in TwoFish key schedule. 29

Agradecimientos

En primer lugar me gustaría agradecer inmensamente a mis padres por todo su apoyo y consejo durante estos cuatro años. Esta travesía habría sido imposible sin vosotros, muchas gracias por enseñarme a ser constante a la par que curioso, jamás podré agradeceros lo suficiente todo lo que habéis hecho por mí.

Me gustaría dar las gracias al Dr. Compton por por su guía y consejo durante todo este proyecto. Ha sido un verdadero placer tenerte como profesor y director este año, muchas gracias por todo el tiempo y la dedicación que han hecho posible este proyecto.

De igual forma doy las gracias a los excelentes profesores que me han acompañado durante el grado. Especial mención a Lucía y Sonja. No sólo por vuestras excelentes explicaciones y por vuestra dedicación constante, sino también por vuestra franqueza e involucración, que es lo que han hecho de estos años una experiencia inolvidable.

Cómo no, a la *study room crew*, que me vieron devanarme los sesos mientras hacía este proyecto y me reconfortaban cuando las cosas dejaban de funcionar. Martin, Jordan, Marc, Emma y Rayaan, muchas gracias por todo el apoyo y la fe que depositabais en mí mientras escribía este pequeño gran monstruo.

Por último, pero no por ello menos importante, muchas gracias a mi compañera de laboratorio favorita. No podría imaginarme estos últimos años sin ti, gracias por tu paciencia y tu constancia; por todos esos momentos en los que perdía en ánimo y me ayudabas a continuar.

Chapter 1

Introduction

"Experience is what you get when you didn't get what you wanted. And experience is often the most valuable thing you have to offer."

– Randy Pausch

COMPUTER security permeates the modern world we live in. From bank transactions to the passcode in our smartphone, our society employs hundreds of information security techniques everyday to keep secure both information and access to resources. Major aspects of our lives rely on modern information technologies being secure, but the tireless efforts to exploit these systems by malicious agents make information security an ever changing field, racing to stay one step ahead of the so called black hats.

Before the modern era, cryptography solely focused in message confidentiality, i.e. transforming comprehensible pieces of information into completely unreadable message in a process called encryption. If correctly performed, eavesdroppers would not be able to extract any information from the encrypted message without some sort of secret knowledge. Thus, cryptography was only about safely communicating messages. After the start of the information age, not only encryption techniques were developed. At the present moment, we employ constantly techniques such as integrity checking, authentication and digital signatures; expanding significantly the role of cryptography in the modern era.

Many of the widely used cryptographic techniques that we use nowadays rely on what are believed to be computationally hard problems. Nevertheless, computational complexity theory is one of the hardest challenges humanity has ever embraced. Even when using what are widely known facts, we sometimes lack the necessary proof, which means that no absolute guarantees can be made. For instance, if an extremely smart mathematician were to discover a fast and efficient way to factor integers, many of the techniques used in public key cryptography would become instantly obsolete.

This lack of complete certainty is coupled with the fact that creating practical security protocols based on theoretical results is not as straightforward as one would expect. Systems with strong security measures have been weakened by details provoked by the physical implementation. Take for instance the Enigma machines used by the German army in World War II. The Enigma never substituted a letter for itself, due to how the electrical wiring was performed in its reflector. This small imperfection later rendered to be a major weakness in the system.

1.1 Motivation

As we can see, modern secure protocols and systems usually rely in what we could call "experience". Well studied protocols that have been around for long enough, and have resisted the all kinds of attacks are believed to be more more secure. Many malicious agents try to design attacks for a personal gain. At the same time, the academic community also tries to expose the weakness and faults of the design and implementation of cryptographic protocols.

This task has a twofold benefit. First, when attacks are discovered on widely used protocols it allows society to change the protocol or tweak the implementation preventing any further attacks of that kind. Secondly, it allows cryptographers to learn about ciphers, what type of patterns or procedures make cipher insecure. This experience is invaluable, since it allows for the construction and design of better, more secure ciphers.

The goal of this work, carried out at the department of Electrical Engineering and Computer Science at the University of Michigan is to look for implementation weaknesses in widely known ciphers employing modern approaches in order to show how current protocols can be compromised, as well as contributing insights for the designers of the next generation of ciphers to take into account.

Chapter 2

Background

"If you think cryptography is the answer to your problem,
then you don't know what your problem is."

– Peter G. Neumann

IN this chapter a number of basic concepts in Cryptography in general and Side Channel Attacks in particular will be introduced in order to lay the foundations from which this thesis will build upon. First, we will cover basic concepts in cryptography such as private key vs. public key encryption, semantic security, etc. Therefore, if you are familiar with these concepts you should be able to feel comfortable skipping Section 2.1.

Later in the Chapter we present an overview of a more specific concept in Cryptanalysis, Side Channel Attacks. These encompass techniques such as Timing attacks, Acoustic cryptanalysis, Differential fault analysis and Power-monitoring attacks among others. The rest of the chapter will focus mainly on the latter of these four since is the technique that will be presented later in the document. Finally, a number of common cryptographic constructs are introduced since they will later be utilized in Chapter 5.

2.1 Introduction to Cryptography

Cryptography is commonly defined as the practice and study of techniques and mechanisms for secure communication of information in the possible intrusion of third parties called adversaries. More generally, cryptography is focused in constructing and analyzing protocols that are able to prevent third parties from reading private messages; various aspects in information security such as data confidentiality, data integrity, authentication, and non-repudiation are central to modern cryptography. Modern cryptography employs techniques that come from various fields both theoretical and practical, since ciphers not only have to be designed but also need an accompanying implementation. [KL14; MVOV96]

In the modern age, cryptography heavily relies both in mathematical theory and computer science practice; cryptographic algorithms are designed around computational hardness assumptions. Namely, it is theoretically possible to break such a system, but it is practically infeasible given the computational power and mathematical techniques given at the time, rendering the scheme *computationally secure*. These secure schemes can be made insecure due to faster computing technology or newly discovered cryptanalysis techniques, which makes cryptography an dynamic and ever moving field. Some schemes such as the *One-Time pad* are denominated perfectly secure, since it is possible to prove that they cannot be broken even with unlimited computing time and space [Sha49]. Nevertheless, perfect secrecy schemes are more difficult to implement that computationally secure algorithms which explains the modern popularity of the latter.

2.1.1 Terminology

Here it is presented a short summary of the common terms and definitions that one usually finds when talking about cryptography or cryptanalysis. **Encryption** is process of transforming ordinary information (called **plaintext**) into unintelligible representation (called **ciphertext**). Ciphertext does not necessarily need to be text and in fact, it is usually represented using integer numbers, bytestreams or even abstract mathematical constructs. **Decryption** is the reverse process in which we take the unintelligible ciphertext back to plaintext.

A **cipher** is usually referred to be a pair of algorithms that describe the encryption and the reversing decryption. The detailed operation of a cipher is controlled both by the **algorithm** and in each instance by a **key**. The key is a secret, which means that is should only be known to the communicants; and is required in order to decrypt the ciphertext. **Cryptanalysis** refers to the study of methods to obtain the meaning of encrypted information without access to the key normally required to do so [Kah74].

These concepts can be easily visualized in Figure 2.1. Here, two agents Alice and Bob exchange a plaintext M using a cipher (E,D) and a pair of keys (K_a, K_b). Alice uses the Encyption algorithm E and the her key K_a to map the plaintext M to the ciphertext C and communicates it to Bob. This communication need not be private, and we can have eavesdropper (depicted with Eve) that can have access the ciphertext but they lack access to the secret keys. Once Bob receives the ciphertext, he uses the decryption algorithm D and his key K_b to recover the original plaintext M transmitted by Alice. [1]

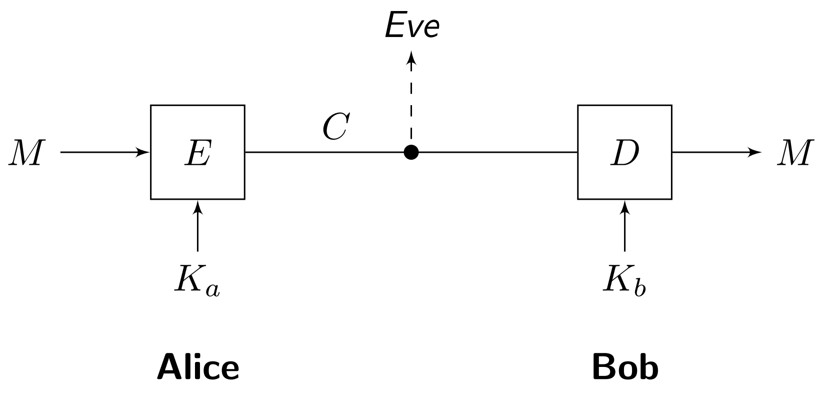


Figure 2.1: Illustration of how secure communication is performed

[1]Note that this is a simplified diagram that ignores a number of key concepts common in modern secure communications, such as integrity or authentication. For a more comprehensive reference please refer to [Sch07]

2.1.2 Confusion and Diffusion

Introduced by *Claude Shannon* in *"A mathematical theory of cryptography"* [Sha45] these two are key concepts in cryptography that used when designing and evaluating ciphers.

Confusion means that every single digit of the ciphertext needs to depend on several parts of the key. Confusion is often achieved by employing nonlinear constructs such as S-boxes. S-box is short for substitution box, lookup tables that cannot be summarized with any sort of linear or pseudo linear expressions.

On the other hand, diffusion measures the amount of change that the ciphertext experiences when we perform a change in the plaintext as well as the change in the plaintext when making a change in the ciphertext. Ideally, for a system to have good diffusion, when we change any single bit of the ciphertext statistically half of the bits in the ciphertext should change and viceversa.

2.1.3 Symmetric-Key Cryptography

Symmetric-key cryptography encompasses the encryption where the same [2] Secret Key (SK) is used to encrypt and decrypt a message. It was the only encryption mechanism until 1974[3] and as a general trend, symmetric ciphers are faster than asymmetric ones. This is caused by symmetric ciphers employing smaller key spaces (which usually implies shorter key lengths).

Famous examples of symmetric cryptosystems include the widely extended Advanced Encryption Standard (AES) as well as its predecessor the Digital Encryption Standard (DES). Commonly, symmetric-key ciphers can be implemented as either block or stream ciphers.

- **Stream Ciphers** create a key stream as long as the original message and then operate on the plaintext in an elementwise fashion (bit-by-bit,character-by-character,etc). The one-time pad is an example of a stream cipher.

- **Block Ciphers** operate on fixed length groups of bits or characters denominated *blocks*. Block Ciphers split the plaintext in blocks of a predefined length. Different modes of operation can be used in order to encrypt arbitrarily long messages using a Block Cipher. Both AES and DES are block ciphers.

  Some block ciphers (including AES, DES and TwoFish) perform the encryption (and decryption) operation in a iterated fashion, applying the same transformation a fixed number of times. These iterated ciphers, also known product ciphers, need a source of randomness to guarantee cryptographic security. Since the key should not be used directly that many times, these ciphers usually employ subkeys that are derived from the key in some fashion. The **Key Schedule** is the algorithm, that given the key, computes the subkeys for these rounds.

[2]Technically they may be different, as with some uncommon schemes; but in these cases they are related in a easily computable manner

[3]DH76.

2.1.4 Public-Key Cryptography

Also known as Asymmetric-Key cryptography, public-key cryptography uses both a public key to encrypt a message as well as a private key to decrypt it. The main caveats of symmetric-key cryptography are both the task of key distribution and the fact that the amount of keys in a network with n members grows as $\mathcal{O}(n^2)$. In their groundbreaking paper [DH76] *Diffie* and *Hellman* introduce public-key cryptography in which two different keys are used. The public key PK^+ and the private key PK^- share an intrinsic mathematical relation, however it is computationally infeasible to get one given the other. Later appeared the first public-key encryption system known as the Rivest-Shamir-Adleman (RSA) algorithm. Named after its designers, it is used everyday in hundreds of millions of systems in order to perform authenticated communications.

2.2 Advanced Encryption Standard Process

From 1997 to 2000, the National Institute of Standards and Technology (NIST) held the AES process in order to choose a successor to the aging DES. Whilst Triple-DES was a possible alternative, its slow performance in hardware motivated the design of a new, more secure system; the Advanced Encryption Standard. In order to provide a more open and transparent process than its predecessor, the selection of AES was open to the public, eliminating suspicious of possible backdoors as DES received.

The process went for several rounds, evaluating not only the security of the candidates but also aspects like performance in settings varying from personal computers to more computationally limited devices such as smart cards. Five finalists, made it to the final round. All of them had respected and well-known members of the cryptographic community as designers. The final votes for each of the five candidates are displayed Table 2.1

Cipher	Positive	Negative
Rijndael	86	10
Serpent	59	7
Twofish	31	21
RC6	23	37
MARS	13	84

Table 2.1: AES2 Conference votes for the AES contest finalists

After the votes were casted, another conference was held in which finalists argued for their designs. On October 2000 NIST declared Rijndael as the winner and has been used ever since as a standard in industry.

2.3 Side-Channel Attacks

As we have previously described, the goal of cryptanalysis is to exploit some weakness in a cryptographic scheme that can allow its evasion or subversion. In cryptography, a so called Side-channel attack is defined as any attack that uses information obtained from the specific physical implementation of a cryptosystem. Whereas classic techniques in Cryptanalysis employ weaknesses in the mathematical construction of the algorithm, side-channel attacks make use of physical variables like timing information, power consumption, error messages produced by faulty inputs or even electromagnetic radiation that is emitted from the device. In Figure 2.2 we can see a more detailed representation of how real world communication occurs.

It is important to understand that even though all Side-Channel attacks rely on information obtained from the technical implementation of the cryptosystem, not all of them require knowledge of the internal workings of the cipher. They can effectively use Differential Cryptanalysis techniques to treat the algorithm as a black box, generalizing the technique to all the ciphers that leak enough information of that physical parameter in question.

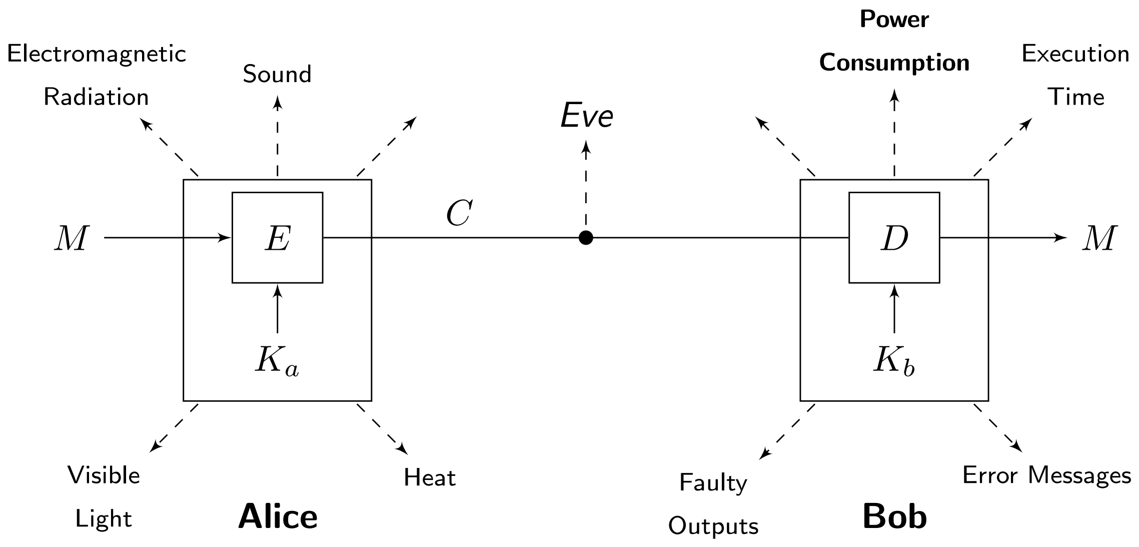


Figure 2.2: Physical parameters that systems can leak when operating[4]

Since side-channel attacks can effectively compromise the security of a cryptosystem, countermeasures are usually taken in order to throttle this kind of exploits . A first straightforward approach to remove the possibility of side channel attacks is to effectively eliminate or decimate the amount of information that is being emitted from the system. While this can be quite effective in cases like timing attacks, it can be extremely hard to implement for other physical variables such as power consumption [Spa06].

A second approach involves making the leaked information useless, i.e. to make the emitted information to be uncorrelated with the variables linked to the execution of the algorithm. Nevertheless, this is sometimes a costly process and usually built on top of the algorithm rather than inside of it; rendering some ciphers susceptible to this type of attacks.

[4]Note that there is no relation between the type of physical parameter and the encryption and decryption procedure

2.4 Power Analysis Attacks

Power analysis attacks are a specific kind of side-channel attacks in which the power consumption trace of an embedded device or integrated circuit such as an smart card is carefully studied in order to gain information from the encryption or decryption process. Depending on whether we treat the system as a glass box or black box we differentiate two types of Power analysis attacks:

2.4.1 Simple Power Attacks

A Simple Power Attack (SPA) involves measuring variations in the power consumption of a device as it is computing the operations, in order to discover information about secret key material or data that is being manipulated. This is achieved by mapping certain operation types to consumption patterns. For example, due to the inherent architecture of the microprocessor, a series of exclusive-OR operations will exhibit a different trace on an oscilloscope to a series of multiplication operations.

A good example of this behavior is the RSA algorithm, which has to perform large multiplications, and therefore leaks information about the internal state of a large integer multiplication via the pattern of operations it performs. This can be visualized in Figure 2.3, where we can see the oscilloscope measurements of the RSA algorithm execution; and we can see the broader peaks showing that the processor is performing long multiplications.

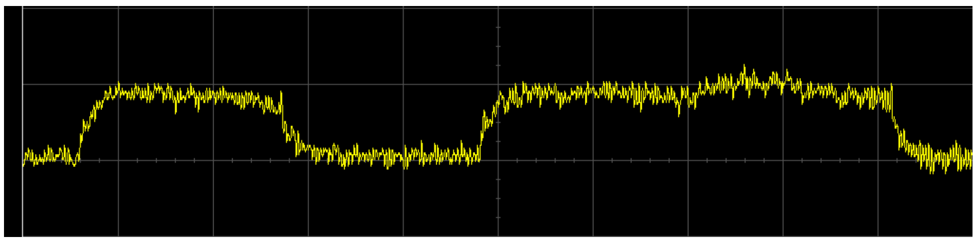


Figure 2.3: Power variations during work of the embedded processor computing RSA signatures[5]

A SPA not only employs techniques to distinguish the types of operations being performed but also makes use of the fact that digital signals are commonly represented physically by high and low voltages. Thus, a 1 being signaled uses more power than a 0 would and if accurate enough analysis can be done, we can perform statistical analysis to model the contents of the bits being processed within a margin of error.

2.4.2 Differential Power Attacks

Differential Power Attacks (DPAs) involve statistically analyzing power consumption measurements from the execution of a cryptographic system. The main caveat of SPAs is that once the processor system starts getting more complex, it becomes significantly harder to monitor and map all the operations being performed due to the amount of noise that other subsystems introduce into the power trace.

[5]Wikimedia Commons. *Power Attack.* 2010. URL: `https://upload.wikimedia.org/wikipedia/commons/6/6c/Power_attack.png` (visited on 07/10/2016).

To overcome this limitation, DPAs analyze both the consumption of the algorithm when cryptographic operations are not being performed and then compares that to the power consumption obtained when the system is actively performing cryptographic operations. In a analogous way to modern noise canceling procedures, DPAs are usually able to remove the "background noise" of the power consumption.

This mechanism enables DPAs to effectively extract secrets from measurements with high levels of noise, where traditional SPAs would fail. Moreover, DPAs are also better at attacking inherent parallel systems such as FPGAs. These systems cannot be easily attacked by SPAs since the concurrency makes the power consumption to be a factor of numerous signals.

A major disadvantage of DPAs is the amount of information they require to obtain confident results. Whilst some SPAs need barely a single power trace of the execution of the algorithm, DPAs usually need samples several orders of magnitude larger to extract information that is statistically significant.

2.5 Common Cryptographic Constructs

Here we introduce a number of commonly used cryptographic constructs since they will be later mentioned to in Chapter 5.

2.5.1 Key Whitening

Key whitening is a method employed to increase the security of a block cipher that goes through several rounds or iterations. The most common procedure involves doing what is called an XOR-Encrypt-XOR which performs input and output whitening by just XORing parts of the key or subkeys at the beginning and end of the encryption.

2.5.2 Pseudo Hadamard Transform

The Pseudo-Hadamard Transform (PHT) transformation is a revertible operation on a bit string that produces cryptographic diffusion. For a more detailed description please refer to [STM10]

$$\begin{aligned} a' &= a + b \mod 2^n \\ b' &= a + 2b \mod 2^n \end{aligned} \tag{2.1}$$

2.5.3 Feistel Network

Named after cryptographer *Hostel Feister*, this is symmetric construction is used by many block ciphers such as DES. The structure features an arbitrary number of rounds as we can see depicted in Figure 2.4. The mathematical behavior is described in Equation 2.2 where L_i and R_i are the left and right halves of the input, and L_{i+1} and R_{i+1} are the corresponding halves of the output. Feistel networks have been proven to be a cryptographically secure Pseudo Random Permutation

(PRP)[6] as long as the function F is a cryptographically secure Pseudo Random Function (PRF)[7] and the number of rounds is as least three. [LR88]

$$\begin{cases} R_{i+1} = L_i \oplus F_{K_i}(R_i) \\ L_{i+1} = R_i \end{cases} \tag{2.2}$$

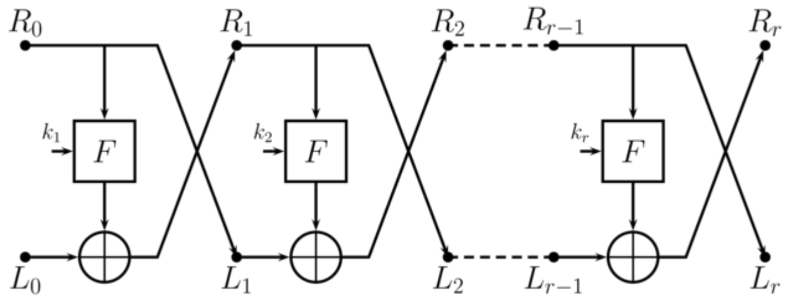


Figure 2.4: Feistel Network[8]

[6]PRP refers to a function that cannot be distinguished from a truly random permutation

[7]PRF are generalizations of PRPs where the domain set and the image need not be the same

[8]DI Management Home. *Feistel Network*. 2013. URL: `http://www.di-mgt.com.au/images/feistel.png` (visited on 07/10/2016).

Chapter 3

State of the Art

"Few false ideas have more firmly gripped the minds of so many intelligent men than the one that, if they just tried, they could invent a cipher that no one could break."

– David Kahn, in The Codebreakers

AFTER having laid the background knowledge needed to understand the problem at hand, we need to do a detailed literature review regarding both the topics of Side-Channel Attacks as well as the publications which analyze the properties the TwoFish block cipher. The first part introduces the historical evolution of Side-Channel Attacks describing with special emphasis the research efforts in Simple Power Attacks. The latter part of the chapter compiles works that describe and study the security properties of the TwoFish block cipher, outlining the few cryptanalysis findings known to date.

3.1 Side-Channel Attacks

Side Channel Attacks were first described in 1996 by Kocher with the introduction of timing attacks, in *"Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems"* [Koc96]. In this pioneering work Kocher showed how to compromise public-key cryptosystems such as Diffie-Helman, RSA key generation or the Digital Signature Standard (DSS). The timing attack was based on the fact that cryptosystems often take slighly different amounts of time to process different inputs. These differences are provoked by the numerous layers of complexity and optimization present in modern hardware. He presented attacks which can exploit timing measurements from vulnerable systems to find the entire secret key.

Nonetheless, timing attacks can be countermeasured by creating constant runtime code, even when that means sacrificing the performance gained from optimization. Since access to the hardware was one of the assumptions, attacks that used the power consumption of the cryptosystem started developing. The first research power attack is credited to Kocher, Jaffe, and Jun in *"Differential power analysis"* [KJJ99]. They were able to obtain information of the data manipulated by the processor by carefully measuring the power consumption of a CMOS chip. In this work, they created the original classification for the two different types of power attacks, discriminating between Simple Power Attack (SPA) and Differential Power Attack (DPA).

They developed a DPA with significant results, showcasing the potential problems of such a technology. The attack was not merely theoretical or limited to smart cards; they were able to employ power analysis techniques to extract keys from almost 50 different products in a variety of physical form factors. As they describe, DPAs are easy to implement, have a very low cost per device, and are non-invasive, making them difficult to detect.

Because a DPA automatically locates correlated regions in a device's power consumption, the attack can be automated and little to no information about the target implementation is required.

3.2 Simple Power Attacks

In a analogous fashion to classical differential cryptanalysis, DPAs require a considerable amount of ciphertexts in order to obtain results of statistical significance. The tradeoff between how much we analyze the algorithm and how much information we can extract from the physical parameters is exemplified by SPAs. They usually require a more in depth analysis of the protocol and device we want to compromise but in compensation statistically significant results may be drawn with as little as a single reading.

This fact made researchers interested in investigating which types of information could SPAs extract so that later they could be used to get information for the plaintext and/or the secret key. An important realization is that since microprocessors perform discrete operations on blocks of data in a sequential fashion, physical imperfections of the system make possible to correlate the power consumed to the binary variables operated upon. More concisely, we may be able to extract some function of the binary words that the microprocessor is handling.

3.2.1 Hamming Weights

Among these possible functions, one that is both easy to obtain and informative is the Hamming weight of the word. When talking about binary variables, the Hamming weight refers to how many non zero symbols the word contains. Namely, the number of ones the binary string contains. As we commented previously, in modern microprocessor systems, digital 1 is generally represented with a high voltage level. The more 1s are required to represent a number the larger we expect the power consumption.

Research has proved that the correlation in power consumption is indeed significant enough to find Hamming weights of numerous intermediate variables from the power utilization. In *"Examining Smart-Card Security Under the Threat of Power Analysis Attacks"* [MDS02], Messerges, Dabbish, and Sloan demonstrate how to extract Hamming weights of the smart card DES implementation. One of the most relevant results of this paper is that if the attacker knows the Hamming weight of each of the k words that comprise the secret key, where each word has n bits of key data, then the average brute-force key search space size is reduced from 2^{nk} to the expression in Eq. (3.1).

$$\left[\sum_{m=0}^{n} \binom{n}{m}^2 \Big/ 2^n \right]^k \tag{3.1}$$

In a similar note in *“Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards”* [MS00], Mayer-Sommer performed an important analysis of how we can correlate the Hamming weights of the processed data to the power dissipation occurring in the device. Mayer-Sommer proved that it is possible to determine Hamming weights in “an unequivocal manner” even under significant amounts of noise. Previous investigations had rendered SPA little threatening and easy to prevent, posing DPA as the only serious menace to smartcard integrity.

Nevertheless, they were able to show through careful experimental methods that a SPA allows for extracting sensitive information, requiring only a single power-consumption graph. The results suggested that SPAs were an effective and easily implementable attack and, due to its simplicity, they can potentially be a more serious threat than DPA in many real applications.

3.2.2 Countermeasures

Concern of these types of attacks led to a number of research efforts to look for ways to thwart them by masquerading the power consumption values. The idea is by masquerading the consumption with noise one could break the correlation that Power Attack uses as a basis to acquire information. Among these works we can highlight the work of Popp, Mangard, and Oswald in *“Power analysis attacks and countermeasures”* [PMO07]. In order to decorrelate this measures, operands are applied random masks before most computations are performed. These techniques based on hiding logic styles effectively randomize the intermediate values of the computation, directly preventing both SPAs and DPAs. Nevertheless, they are still far from being industry standards and most modern smartcards and ASICs do not yet include mechanisms like those ones by default.

3.2.3 SPAs on AES Finalists

Recent research efforts have demonstrated the usefulness of Hamming weights for SPAs, showing that both Rijndael and Serpent smartcard implementations are vulnerable to this type of attacks. In both cases the weakness arose from the key schedule computation; patterns in the hamming weights of the key schedule algorithm revealed enough information to compromise the secret key.

The first attack was shown for Rijndael, the winner of the AES constest by VanLaven, Brehob, and Compton in *“A Computationally Feasible SPA Attack on AES via Optimized Search”* [VBC05]. The attack employed a recursive optimized search on the key schedule hypergraph proving that the AES standard is in fact susceptible to very efficient attacks which are based solely on the Hamming weights of the expanded key. They were able to unequivocally recover the key from a single power reading of the execution of the key schedule. Rather that relying on the guarantees of perfect Hamming weights from the results of [MS00], the authors introduced a noise correction technique which increased the computational complexity of the algorithm.

Later research by Compton, Timm, and VanLaven proved that Serpent, the second finalist of the process (Table 2.1) is also susceptible to a Hamming weight based SPA. In a similar fashion to the previous work on AES, *“A Simple Power Analysis Attack on the Serpent Key Schedule”* [CTV09] used Hamming weights from the key schedule to recover the secret key. More precisely, Hamming weight measurements from 8-bit smart card imple- mentations of

Serpent's key schedule revealed enough side channel information to uniquely determine a 256-bit initial key in a few milliseconds.

More generally, they showed that Linear Feedback Shift Registers (LFSRs)[1] are a design pattern susceptible to SPA attacks.

3.3 TwoFish Cryptosystem

3.3.1 Original Definition

The original description of the TwoFish algorithm was performed in *"Twofish: A 128-Bit Block Cipher"* [Sch+98]. Schneier et al. submitted the design to the AES process and in the original paper apart from describing the behavior and structure for the block cipher they elaborated in the performance metrics and how the design choices were made. Moreover, the creators performed a survey of attacks to the cipher covering a broad number of attacks, including Linear cryptanalysis, Differential cryptanalysis and Related-Key attacks among others.

Efficient smart-card performance was one of the design choices in the AES process and subsequently TwoFish provided an implementation for a couple of 8-bit assembly languages[2] (Motorola 6800 and Zilog Z80). In the submitted paper the authors did cover the topic of Side-Channel Cryptanalysis, however the description is brief and more focused towards timing attacks and fault analysis than power analysis attacks.

Still during the AES contest, the authors published *"On the Twofish Key Schedule"* [Sch+99]; a paper in which they described and carefully analyzed the subkey generation algorithm needed to introduce randomization into the consecutive encryption rounds. Similarly as in [Sch+98], the authors described the properties of the key schedule to withstand Related-key differential attacks. The discussion was predominantly theoretical and no mention was done to how the key schedule could prevent power analysis attacks.

A year later, after the announcement of Rijndael as the winner of the AES process the TwoFish creator released *"The Twofish team's final comments on AES Selection"*, where they reiterated the stength of TwoFish as a cipher giving specific reasonings and proofs for the suitability of the cryptosystem as a widely used standard. They compared TwoFish to the other finalists and suggested modifications, to some of them including the winner Rijndael. The main critique to Rijndael was the small amount of rounds performed and they recommended extending it. One of the important contributions of the paper was the remark that *"[...] Two of the finalists, MARS and RC6, simply do not work well in certain applications. Any one of the other three algorithms — Rijndael (with the extra rounds), Serpent, or TwoFish would make an excellent standard"*.

[1]Linear-Feedback Shift Register are a type of digital systems, in which the output from a standard shift register is operated and fed back into its own input in such a way as to cause the function to endlessly cycle through a sequence of patterns.

[2]Publicly available implementations can be found at `https://www.schneier.com/cryptography/twofish/download.html`[Tsc]

3.3.2 Security of TwoFish

One the main reasons for Schneier et al. to advocate for the standardized use of TwoFish is how unsuccessful cryptanalysis performed on the cipher has been. Even when considering later literature, we can only find a handful of so called attacks to the algorithm.

In 1999 Biham and Shamir published *"Power analysis of the key scheduling of the AES candidates"* [BS99] where they carried out a preliminary analysis of power attack susceptibility of various AES cadidates. They assessed that TwoFish Key Schedule had a complex structure and seemed not to reveal direct information on the key bits from Hamming measurements. The results presented later in this work refute this assertion since an efficient and robust SPA attack that retrieves the secret key was found.

The same year Ferguson showed that impossible differentials could be performed to break at least 6 rounds of the cipher [Fer99]. Due to the key sizes employed for AES finalists, unfortunately this is not a practical attack. A year later, Kelsey pointed the Key Separation of TwoFish [Kel00]. The fact was later addressed in [Sch+00], but no practical application of this property was ever used for any attack.

In 2001 Lucks published *"The saturation attack—a bait for Twofish"* [Luc01], in which the best known attack to TwoFish is described,reducing by a factor of between 2 and 4 the complexity that a brute force approach would require. Nonetheless as with the work by Ferguson, due to the key sizes being $N = \{128, 192, 256\}$, using even a halved or quartered search space is not practical.

It is also important to mention the analysis performed by Mirza and Murphy in *"An observation on the Key Schedule of Twofish"* [MM99]. The paper describes two significant properties of the TwoFish key schedule: Firstly, there is a non uniform distribution of 16-byte whitening subkeys. Secondly, in a reduced (fixed Feistel round function) TwoFish with an 8-byte key there is a non-uniform distribution of any 8-byte round subkeys. This proves that the key scheduling of TwoFish has some weaknesses and as we shall see later, the weaknesses found in the key schedule is what compromises the cipher as a whole.

As a final note, when looking for TwoFish power analysis investigation, we can see that even though TwoFish smart-card implementation performance and versatility have been throughly analyzed in works such as *"Performance analysis of AES candidates on the 6805 CPU core"* [Kea99] and *"Performance Analysis of AES and TwoFish Encryption Schemes"* [RHW11], to the best of our knowledge, no power attacks have been found for the encryption algorithm or the key schedule.

Chapter 4

Project Definition

The point of academic attacks is not exhibiting practical breaks; the point is that only a trained cryptographer can tell whether a given algorithm is secure or not. The author of an algorithm says: "My cipher is secure, and trust me, I am an expert at this. And to prove that I am a real good expert, I challenge other experts to find even the most impractical, academic flaw in my cipher".

– Thomas Pornin

BEFORE proceeding further with this project's eponymous attack, there is a need to considerate aspects that motivate the work carried out in the project. Accordingly, this chapter will cover in more detail the objectives and expected outcomes of the project, outlining the relevance and impact of the research findings. In addition, we will describe the technologies and tools that were needed to carry out the project better contextualizing its practicality in real world scenarios.

4.1 Objectives

This investigation focuses on finding a Simple Power Analysis Attack on the TwoFish Key schedule. TwoFish was one of the finalists for the Advanced Encryption Standard and with its proposal it provided a 8-bit implementation for smart cards. The attack will look at this implementation and try to break the key from power readings of the smart card execution.

The main objective of this project is to design and implement a Simple Power Analysis Attack on the block cipher TwoFish. TwoFish was one of the finalists for the Advanced Encryption Standard and with its proposal it provided a 8-bit implementation for smart cards. The attack will look at this implementation and try to break the key from power readings of the smart card execution.

4.1.1 Main requirements

The attack needs to fulfill a series of requirements in order to be a significant piece of research. These are outlined below, describing why they are relevant to the behavior of the attack.

- **Feasible** – The attacks needs to rely in techniques that can be performed in the real world. This requirements means that either the specific techniques used are proved empirically to be possible or they are drawn out from peer reviewed research findings.
- **Accurate** – For the system to truly be effective it needs to extract precise secret information from the execution. Accuracy will be strictly enforced by considering all partial answers wrong; a key that is 1 bit wrong will be completely wrong since the diffusion and confusion methods that are built into the cryptosystem will make that difference avalanche through the encryption and decryption procedures.
- **Noise Resistance** – Due to the inherent restrictions of SPAs, the working environments where they can be performed have multiple sources of electromagnetic radiation that overlap with the measured signal. The algorithm will have to account for it in the key search or with correction factors if the key is found via estimation methods.
- **Efficiency** – Whereas some power attacks are effective with thousands or even millions of power readings, real world instances do not provide usually access to that many power traces. Therefore, the attack should ideally run in a polynomial time that translate to reasonable times given computational technologies. Previous research suggest that this type of attacks can range from milliseconds to a few days, thus we impose ourselves the restriction of the algorithm running within this time frame. Longer runtimes could result in good cryptographical insights, but would render the attack impractical.

4.1.2 Secondary Requirements

Whilst also important the requirements outlined below should only be prioritized once the main ones have been already met or if they are acquired gratuitously through the design process.

- **Complete** – An attack that is able to uncover the entirety of the secret information is complete. For instance, if an attack is able to compromise parts or the entirety of the plaintext it would be effective but incomplete, since it would not have access to the key. If we assume public availability of ciphertexts (which is an extremely common assumption in the cryptography domain), retrieving the secret key would make the attack complete since one could recover the plaintext. What is more, access to the key would allow the attacker to read past or future plaintexts and to forge the identity of the key owner.
- **Dynamic** – Ideally, the attack should be able not only to learn from the attacks in a static, independent way; but could employ numerous readings to learn useful patterns of the keys or plaintexts.
- **Influential** – This can be sometimes difficult to measure, specially since it depends on the works and outcomes of future research projects. The idea is to uncover weakness that are not particular to a implementation, i.e. to uncover theoretical or practical shortcomings of more general patterns or structures widely used in research.

  For example the work presented in [CTV09] not only revealed an SPA for the Serpent Cipher but also analyzed the inherent flaw of using modified LFSRs in key schedule algorithms. From these kind of insights the cryptographic community learns as a whole improving the design and implementation of future ciphers.

4.2 Impact

As we have explained previously, the TwoFish crpytosystem was designed as a submission to the AES contest organized by NIST. Despite not winning the contest, TwoFish came as one of the strongest contenders along with Serpent. Since then, even though the major public uses AES (Rijndael), both Serpent and TwoFish are used when the security concerns or the hardware performance is more suitable than the one obtained by AES.

Unlike AES, the successful cryptanalysis approaches to TwoFish are up to date non existent. As we saw in the previous chapter, the main research efforts only reveal aspects of the cipher which given the correct context of an attack could help reduce the security. Nevertheless, none of those shortcomings have proved useful enough to compromise the system. Even when considering the non-uniformity of research efforts when compared to the popularity of AES, after more than 18 years of the design being publicly available no efficient attacks have been found.

However, among all the analysis performed by the cipher authors and independent researchers, almost none of them considered the possibility of a Power Analysis Attack to the cipher. The only researchers that did look into it, Biham and Shamir, did not perform a through enough analysis.

Among the AES finalists, both AES and Serpent have been proven to be susceptible to power attacks in the past decade, well after their publication. Among all the finalists, just Rijndael (AES), Serpent and TwoFish were considered good enough to be the AES standard by the cryptographic community at the time. Thus, if two of the three widely peer reviewed symmetric block ciphers available have a specific type of attack and no analysis of SPAs has been performed in the remaining one, it deems urgent to study the viability of such an attack.

The impact of the attack can be greatly appreciated when we contextualize the places where SPAs can be performed. Even tough the original description back in 1999 used devices such as smart cards, the same principles could in the present time be extended to other remotely powered systems. With the arrival of new ubiquitious computing technologies such as the electronic payment or the so called *Internet of Things*, the amount of embedded devices and circuits has increased immensely.

Industry figures are able to quantify this trend. In surveys such as *"Embedded computation meets the world wide web"* [BW15], we can find in 2015 out of all the computing devices 98% of them were embedded devices. When looking among those at connected devices (which at the same time are extremely more susceptible to be attacked), we find that in 2015 there were 15 billion connected devices as pointed out in *Rise of the Embedded Internet* [Int15].

Moreover, future predictions only suggest that this figure is going to keep growing and at even higher rates. For instance, Gartner's analysis in [Wor15] the expected number of connected units due to the rise of the *Internet of Things* will be over 30 billion connected devices in 2020. Out of this 30 billion, more than 25 billion will be embedded devices. Embedded systems are becoming more and more widespread and with the arrival of the *Internet of Things*, confidentiality and integrity of the information acquired and transmitted by these devices will be an crucial concern.

Finally, as the quote that introduced this chapter eloquently said; sometimes research efforts to develop attacks can be considered oddly specific or impractical. Even if that is not the case of the work presented here, there is a reason for that behavior. When an attack is designed, patterns and properties of the cipher or the mere interaction between some of them is what makes the attack feasible. From that kind of insights the scientific community as well as the industry benefits, since it gains experience of what makes a secure system secure and what does not.

4.3 Technologies

As we will be able to see in the next Chapter, the attack description along with the discovered weak points of the ciphers are purely theoretical and thus agnostic to any platform or language. Regarding software for the attack description, only the TwoFish source code[1] implementation for smart card was required in order to corroborate how the operations are performed in the microprocessor.

For instance, to implement the S-Boxes one could refer to the original procedural definition introduced in [Sch+98] or precompute the lookup tables. Depending on which option is the one that the hardware is performing, more or less information will be revealed. Therefore the only required information to properly describe the attack is the formulation introduced in *"Twofish: A 128-Bit Block Cipher"* [Sch+98] as well as the source code for the smart cards.

Once we achieve a feasible SPA, due to the mathematical formulations not being able to throughly analyze the performance of the algorithm, we will need to implement the attack in some programming language to generate results. The programming language of choice was Python 2.7.11 due to its ease of use and high level abstractions that greatly simplifies the data manipulation and transformation that will be required to implement the algorithm.

In order to speed up the matrix computations, the scientific computing libraries `numpy` and `scipy` [VCV11] have been used. Since they rely on C and Fortran linear algebra primitives, this greatly enhances the performance in tasks such as Least Mean Squares. Due to the high number of experiments needed to be run for all the parameter combinations, a cluster allocation in the University of Michigan was used to run the simulations in batch in several parallel queues. Clocking at 2.6 GHz, the cores used streamlined the simulation process.

Moreover, the data generated from every attack scenario was visualized using Python's canonical plotting library `matplotlib`.

The main caveat of choosing Python as the programming language for the attack is that it can be considered as a slow interpreted language when compared with other compiled or precompiled languages such as C++ or Java. Being this a real time system attack we could be concerned with a faster framework. As we shall see in Chapter 6 the execution times for the worst case scenario are below a second, so there is no need to worry about the runtime. If it were the case that a faster implementation was needed we can mention the reader the existence of `Cython` [Beh+11], a Python language extension that allows explicit type declarations and is compiled directly to C. This would effectively overcome timing limitations whilst making use of the implementation given in this paper.

[1] *TwoFish Source Code*. 1998. URL: `https://www.schneier.com/cryptography/twofish/download.html` (visited on 03/01/2016).

4.4 Planning and Cost Estimation

This project was carried out as a 4 credits (8 ECTS) semester project in the University of Michigan. The deliverable was an scientific article of publishing quality reporting the findings and the implications. The Gantt chart used during the project is showcased in Figure 4.1. The cost estimate [2] including human and material resources is displayed in Table 4.1.

Profile	**Hours**	**Rate** [$/h]	**Total** [$]
Student research work	300	37	11100
Full professor supervision	20	175	3500
High Performance Computing	1000	0.053	53
Total			14653

Table 4.1: Estimation of costs

[2]University of Michigan sources were consulted in order to get the hourly cost ratios. These take into account both the compensation and the nominal cost of the employee.

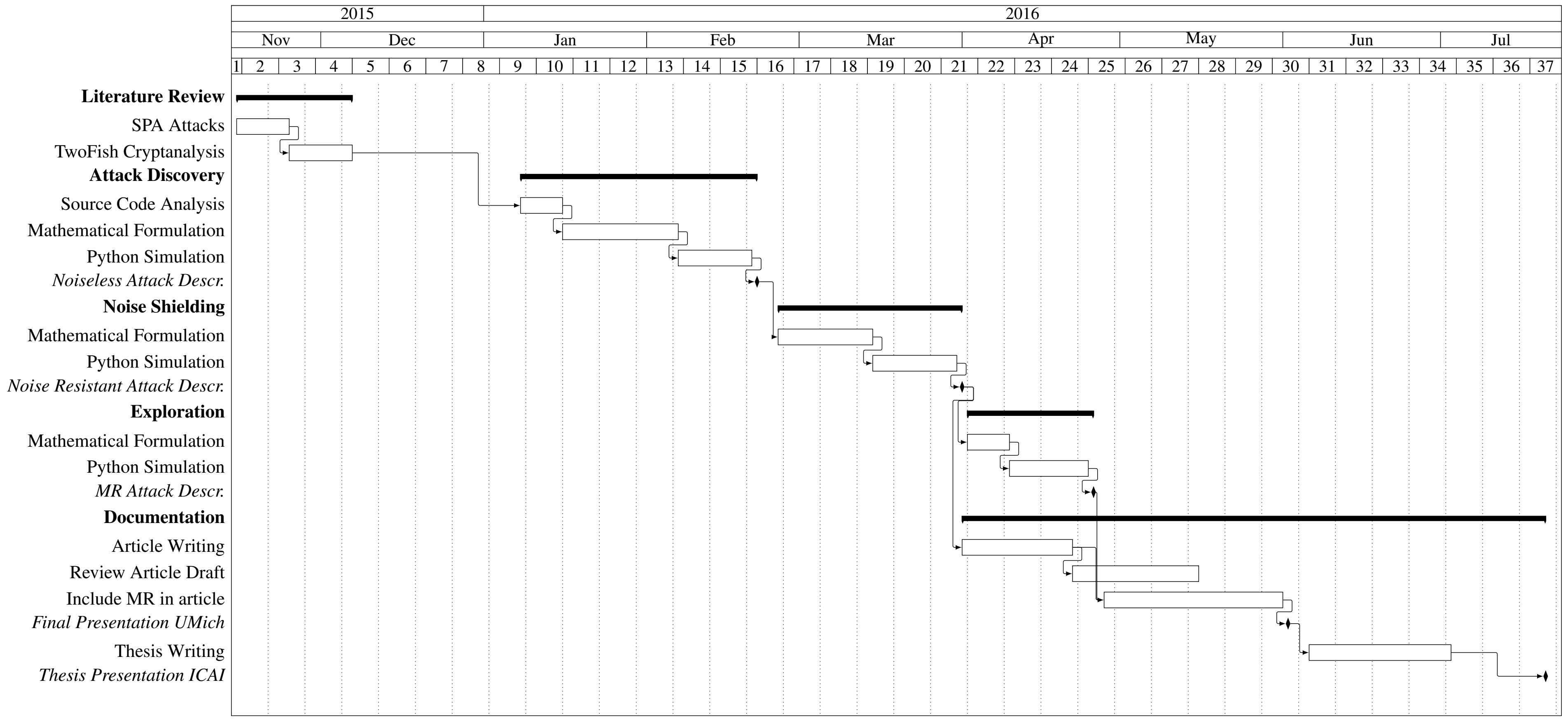


Figure 4.1: Gantt chart of the project planning

Chapter 5

Attack Description

"Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break."

– Bruce Schneier, co-creator of the TwoFish algorithm

ONCE we have laid the necessary foundations upon which this work will build upon, the current state of the art and the corresponding motivation we can dive into the analysis of the TwoFish encryption algorithm and more precisely the operations that it performs during the key schedule execution. This chapter will focus in carefully analyzing the algorithm and devising an SPA that carefully exploits the information revealed by a single power trace of the TwoFish key schedule computation.

Section 5.1 will briefly introduce the encryption scheme and key schedule in the TwoFish block cipher following the notation and terminology from [Sch+98], as well of a more extended mathematical notation of the key schedule variables. Then, Section 5.2 will present the original attack that assumes a perfect trace of hamming weights with no interference. This attack is further expanded in Section 5.3 to make it to cope with significant amounts of noise. Finally, Section 5.4 describes how to extend the algorithm to make use of multiple power readings. Numerical results of the algorithms presented here will be introduced in Chapter 6 whereas the software specification can be found in Appendix B.

5.1 TwoFish Block Cipher

As we have have previously mentioned, TwoFish is a symmetric key block cipher with a block size of 128 bits and key sizes up to 256 bits. It was one of the five finalists of the Advanced Encryption Standard contest organized by NIST and it was submitted by Schneier et al. [Sch+98].

Twofish features pre-computed key-dependent S-boxes, and a relatively complex key schedule. One half of an N-bit key is used as the actual encryption key and the other half of the N-bit key is used to modify the encryption algorithm. Twofish borrows some elements from other designs such as a Feistel structure from DES or the [STM10] from the SAFER family of ciphers [Mas94].

5.1.1 TwoFish Encryption Algorithm

The TwoFish encryption is a 16 round Feistel Network with both input and output whitening. Each round operates only in the higher 64 bits of the block and swaps both halves. A total of 40 subkeys are generated from the secret key, with each key being 32 bits[1].

Keys $K_0 \dots K_3$ are used for the input whitening, $K_4 \dots K_7$ are used for the output whitening and $K_8 \dots K_{39}$ are used as the round subkeys. Each round employs a function F which is a key-dependent permutation on 64-bit values. The function F splits the 64-bit string into two 32-bit substrings and applies the g function to each half. The function g applies a fixed number of S-box substitutions and XORs with parts of the secret key (the number of steps this is performed depends on the size of the key).

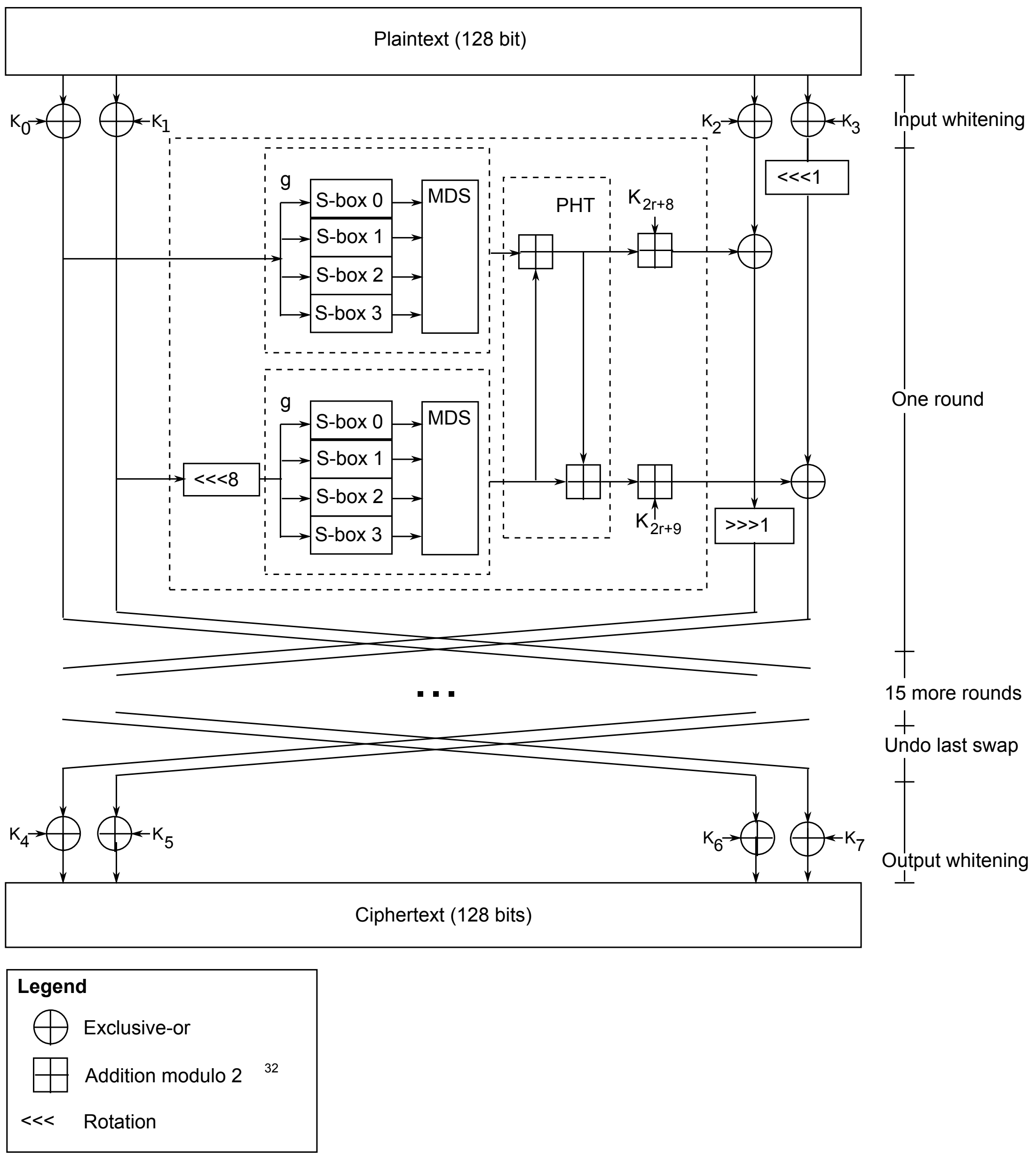


Figure 5.1: TwoFish algorithm[2]

[1]Refer back to Section 2.5 for a more detailed explanation of some of these concepts.

This is followed by a Maximum Distance Separable (MDS) Matrix transform, a Pseudo-Hadamard Transform (PHT) and round key addition modulo 2 with the two subkeys associated for that said round. Therefore each round r uses two subkeys as round key, namely K_{2r+8} and K_{2r+9}. Finally, the output is XORed with one half of the block following the Feistel Network scheme. Bitwise rotations and shifts are performed at strategic points in the encryption to maximize diffusion. They have been omitted to simplify the explanation of the encryption procedure. The algorithm can be visualized in Figure 5.1

5.1.2 TwoFish Key Schedule

The key schedule has to provide 40 32-bit words of expanded key $K_0, \ldots, K_{39}$. TwoFish is defined for sizes $N = \{128, 192, 256\}$. Shorter keys can be padded with zeros to the next larger key length. The key is split into $2R$ 32-bit words $K = (M_0, M_1, \ldots M_{2R-1})$ where $R = N/64$. To generate each subkey all the bits in the secret key are employed. Note that the number of generated subkeys is always 40 regardless of the size of the key, since the number of rounds applied is fixed. This contrasts with the designs of other systems like AES or Serpent where the size of the key determines the number of rounds performed.

The secret key is also employed to derive the vector $S = (S_{R-1}, S_{R-2}, \ldots, S_0)$ which is obtained by applying a Reed Solomon Transformation to the key. These values are used in the g function inside the F permutation as part of the encryption algorithm.

Subkeys are generated in even-odd pairs K_i,K_{i+1}, with even i. To generate this pair of keys, two 32-words are initialized, the first word has all its bytes equal to i and the second word has all its bytes equal to $i+1$. Then, both words go through R rounds, each round being composed of a specific substitution box arrangement followed by an XOR with a corresponding part of the secret key. This combination makes the S-boxes key dependent. For example, for $|K| = 128$ we have $R = 2$, so the h function will have two rounds as shown in the key schedule diagram in Figure 5.2.

Next, the words go through another S-box substitution, and through a MDS transform. All of these 32 bit S-boxes are composed of a predefined choice of two 8-bit permutations $q0$ and $q1$. This choice is fixed and only depends on the stage of the function we are in[3]. The transformation up to this point is defined to be the h function. Next, an 8 bit right rotation is applied in the odd word. Finally a Pseudo-Hadamard Transform is applied to both values resulting in the pair of subkeys K_{2n}, K_{2n+1}.

We can see that the procedure is quite similar to that of the round function, and in fact the g function can be expressed in terms of the h function. However, the round function does not use directly the bits of the secret key, but instead a Reed Solomon Matrix is applied beforehand as we previously mentioned. For a more detailed explanation please refer to the equations and diagrams in [Sch+98].

[2]Wikimedia Commons. *Twofish diagram.* 2008. URL: `https://upload.wikimedia.org/wikipedia/commons/e/ee/Twofishalgo.svg` (visited on 07/10/2016).

[3]Even though S-boxes $q0$ and $q1$ have a procedural mathematical definition, for the purposes of this work, they can be seen as non-linear byte permutations $\mathcal{P} : \{0,1\}^8 \to \{0,1\}^8$

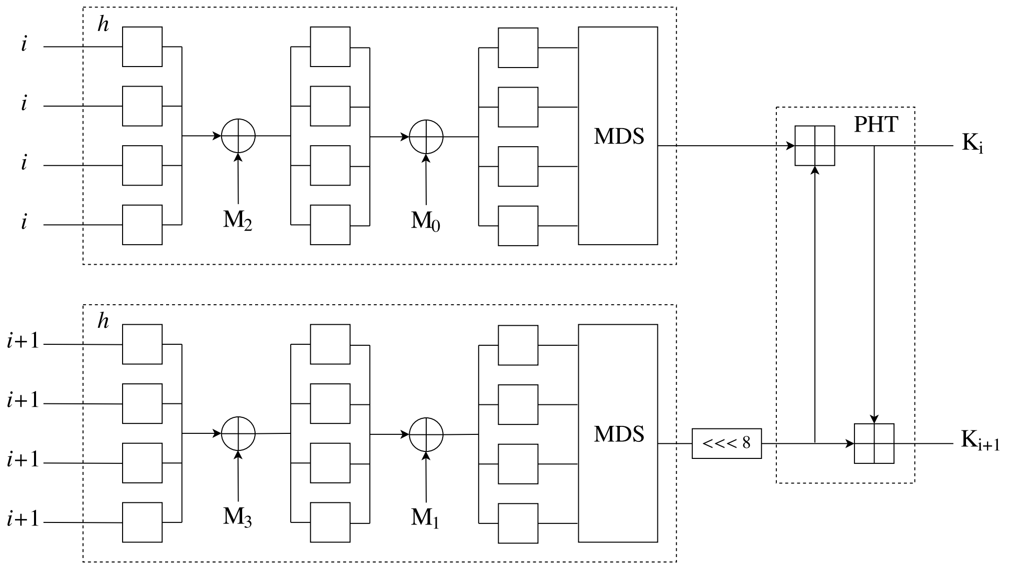


Figure 5.2: TwoFish Key Schedule (128-bit key)

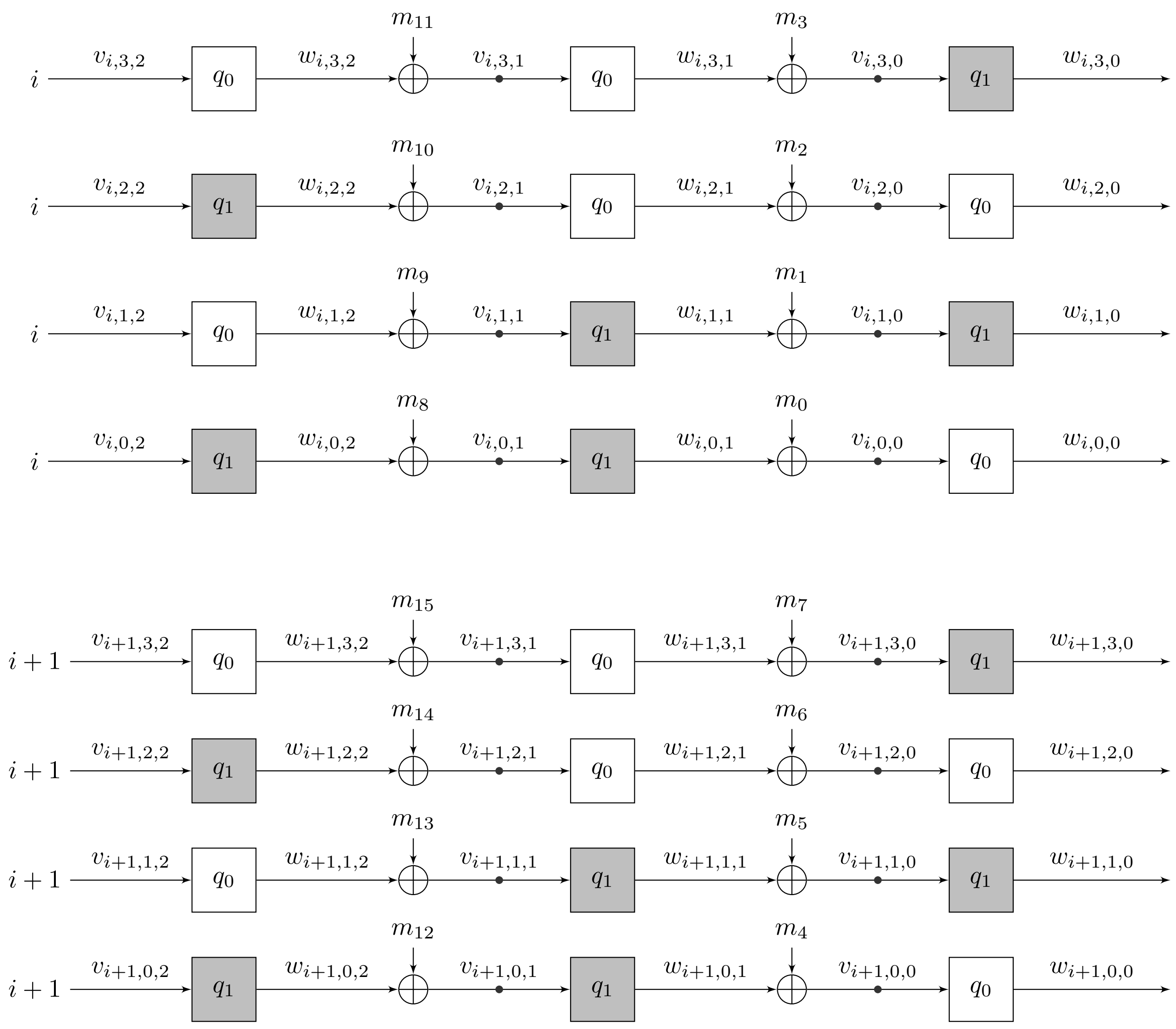


Figure 5.3: TwoFish Key Schedule bit level computations (128-bit key)

5.1.3 TwoFish Subkey generation

The main problem when using the notation in [Sch+98] is that it describes the algorithm mainly in operations of 32-bit words. Since we would like to perform an attack that will use the trace of Hamming weights from an 8-bit processor implementation, we are interested in having a mathematical formulation that operates on the byte level. Therefore, in this section we will derive the equations for the TwoFish key schedule in byte form, laying out the mathematical notation that we will use in the Simple Power Attack in the next section.

Although the secret key values are both used for the round function during the encryption procedure as well as for deriving the subkeys in the key schedule, the attack was found just by looking at Hamming weights of the key schedule algorithm, so only notation for the h function is going to be introduced.

In section 5.1.2 we described the h function involved in the key schedule procedure, which is depicted in Figure 5.2 for a 128-bit key. Expanding the diagram to take into account all the descriptions of the key schedule, we get the layout shown in Figure 5.3.

Here i is an even integer and the bytes $m_0, \dots, m_{15}$ are the 16 bytes of the key. Blocks $q0$ and $q1$ are byte substitution boxes implemented using look-up tables, and their position along the rows and rounds is part of the specification of the algorithm. Variables v_{ijk} and w_{ijk} will be later used to express the relationship between the different rounds of the algorithm. Finally, the grey dots represent the places where a Hamming weight is retrieved from the trace in the algorithm described in Section 5.2.

As we can see from the diagram we will need at least three indexes in order to identify a particular Hamming weight inside the h function structure.

i – Identifies the subkey we are generating as stated previously. Index i will range from 0 to 39, since 40 subkeys need to be generated.

j – Specifies the byte within the 32-bit word taken in the h function. j will range from 0 to 3. In the diagram this is represented as rows.

k – Will identify in which round inside the h function we are at. Since the number of rounds will depend on the size of the secret key, k will range from 0 to R, where $R = |K|/64$.

In general, the key consists of $8R$ bytes, which are all used in every execution of the h function.

$$K = \{m_0, m_1, \dots, m_{8R-1}\} \tag{5.1}$$

In order to describe the relationships between the intermediate values within the h function we can use two byte vectors $\boldsymbol{V}$ and $\boldsymbol{W}$ to represent the values before and after applying a S-box respectively.

$$\boldsymbol{V} = \left\{ v_{ijk} \in \{0,1\}^8 : \quad i = 0,1,\dots,39 \quad j = 0,1,2,3 \quad k = 0,1,\dots,R \right\} \tag{5.2}$$

$$\boldsymbol{W} = \left\{ w_{ijk} \in \{0,1\}^8 : \quad i = 0,1,\dots,39 \quad j = 0,1,2,3 \quad k = 0,1,\dots,R \right\} \tag{5.3}$$

This notation is displayed in Figure 5.3. One important observation is that the index k decreases as we apply more rounds. This comes from the fact that increasing the key size adds more rounds to the left of the h function, leaving the rest of the rounds unaltered. Since the layout of $q0$ and $q1$ boxes depends both on j and k, the simplest way to define this relationship is to increment k from right to left. Otherwise the S-box layout would need to be redefined for each one of the key sizes.

Furthermore, to generalize the notion of the S-boxes to these newly defined indexes, we can create a matrix P whose elements P_{jk} are 0 when the substitution used in row j and round k is $q0$ and 1 if it is $q1$ [4] . This allows us to succinctly store the disposition of $q0$ and $q1$ in the algorithm and define a permutation $Q_{jk}[x]$ that will evaluate to either $q0$ or $q1$.

$$P = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix} \tag{5.4}$$

$$Q_{jk}[x] = \begin{cases} q_0[x] & \text{if} \quad P_{jk} = 0 \\ q_1[x] & \text{if} \quad P_{jk} = 1 \end{cases} \tag{5.5}$$

Given all these definitions we can expresses the function h using the following relations.

$$v_{i,j,R} = i \tag{5.6}$$

$$w_{i,j,k} = Q_{jk}[v_{i,j,k}] \tag{5.7}$$

$$v_{i,j,(k-1)} = w_{i,j,k} \oplus m_l \tag{5.8}$$

As we see in equation (5.8), a index l for the specific byte of the key is required. In the diagram we could see directly where every byte of the key was used. In general, half of the executions of h will require even words of the key $M_e = \{M_0, M_2, \ldots\}$ and the other half will require odd words of the key $M_0 = \{M_1, M_3 \ldots\}$.

We can express l in terms of the rest of the indexes, so from now on when l is used, it can be thought as a function of i, j, k.

$$l(i,j,k) = 8(k-1) + j + 4(i \bmod 2) \tag{5.9}$$

It is important to notice that in order to simplify the notation, l will be sometimes used without directly specifying the associated i, j and k. In those cases, this short hand notation can be substituted and the expression reevaluated with the contextual i, j, k.

[4]The easiest way to check this notion is to compare the 0,1 values in Equation 5.4 to the white (0) and grey (1) boxes present in Figure 5.3. Note that column order needs to be reversed.

5.2 TwoFish Key Schedule Power Analysis Attack

From the formulation described in the previous section we can see that each byte of the key m_l is used 20 different times in order to generate the 40 round subkeys. Since $m_l \in \{0,1\}^8$, knowing the Hamming weights of w_{ijk} should allow us to determine the secret key with a extremely high probability.

We can define the Hamming weight $\mathrm{H}[x]$ of a N-bit variable as follows, where b_i is the ith bit of x.

$$\begin{aligned} \mathrm{H} : \{0,1\}^N &\mapsto \mathbb{N} \\ \mathrm{H}[x] &= \sum_{i=0}^{N-1} b_i \end{aligned} \tag{5.10}$$

Since we want to compute Hamming weights it is useful to express $v_{i\,j\,k}$ and $w_{i\,j\,k}$ in binary form as follows:

$$v_{i\,j\,k} = \sum_{n=0}^{7} c_{i\,j\,k\,n} \cdot 2^n \qquad c_{i\,j\,k\,n} \in \{0\,1\} \tag{5.11}$$

$$w_{i\,j\,k} = \sum_{n=0}^{7} d_{i\,j\,k\,n} \cdot 2^n \qquad d_{i\,j\,k\,n} \in \{0,1\} \tag{5.12}$$

Where $c_{i\,j\,k\,n}$ and $d_{i\,j\,k\,n}$ represent the nth bit of $v_{i\,j\,k}$ and $w_{i\,j\,k}$ respectively

In order to express the xor relation described in Equation (5.8), it will be convenient to have as well the bit representation of m_l, where $x_{l\,n}$ will be unknowns since they represent the individual bits of the secret key.

$$m_l = \sum_{n=0}^{7} x_{l\,n} \cdot 2^n \tag{5.13}$$

Now we can formulate the relationship of Equation 5.8 in a bitwise form

$$c_{i,j,(k-1),n} = d_{i,j,k,n} \oplus x_{l,n} \tag{5.14}$$

Taking Hamming weights in both sides of said equation renders the following expression.

$$\mathrm{H}[v_{i,j,(k-1)}] = \mathrm{H}[w_{i,j,k} \oplus m_l] = \sum_{n=0}^{7} d_{i,j,k,n} \oplus x_{l,n} \tag{5.15}$$

Looking into (5.15) we can see that we will have 40 equations and 16 variables, but even and odd equations will refer to different m_l and $x_{l,n}$. This is caused by the alternation in even and odd expressions. This allows us to split this system of equations into two disjoint systems of 20 equations and 8 variables since they are completely independent of one another.

Thus, for each j,k we will arrive at two systems of equations, one for the even subkeys (Equation (5.16)) and the other one for the odd subkeys (Equation (5.17)).

$$\begin{cases} \mathrm{H}\left[v_{0,j,(k-1)}\right] = d_{0,j,k,0} \oplus x_{l,0} + d_{0,j,k,1} \oplus x_{l,1} + \ldots + d_{0,j,k,7} \oplus x_{l,7} \\ \mathrm{H}\left[v_{2,j,(k-1)}\right] = d_{2,j,k,0} \oplus x_{l,0} + d_{2,j,k,1} \oplus x_{l,1} + \ldots + d_{2,j,k,7} \oplus x_{l,7} \\ \mathrm{H}\left[v_{4,j,(k-1)}\right] = d_{4,j,k,0} \oplus x_{l,0} + d_{4,j,k,1} \oplus x_{l,1} + \ldots + d_{4,j,k,7} \oplus x_{l,7} \\ \qquad \ldots \\ \mathrm{H}\left[v_{38,j,(k-1)}\right] = d_{38,j,k,0} \oplus x_{l,0} + d_{38,j,k,1} \oplus x_{l,1} + \ldots + d_{38,j,R,7} \oplus x_{l,7} \end{cases} \tag{5.16}$$

$$\begin{cases} \mathrm{H}\left[v_{1,j,(k-1)}\right] = d_{1,j,k,0} \oplus x_{l',0} + d_{1,j,k,1} \oplus x_{l',1} + \ldots + d_{1,j,k,7} \oplus x_{l',7} \\ \mathrm{H}\left[v_{3,j,(k-1)}\right] = d_{3,j,k,0} \oplus x_{l',0} + d_{3,j,k,1} \oplus x_{l',1} + \ldots + d_{3,j,k,7} \oplus x_{l',7} \\ \mathrm{H}\left[v_{5,j,(k-1)}\right] = d_{5,j,k,0} \oplus x_{l',0} + d_{5,j,k,1} \oplus x_{l',1} + \ldots + d_{5,j,k,7} \oplus x_{l',7} \\ \qquad \ldots \\ \mathrm{H}\left[v_{39,j,(k-1)}\right] = d_{39,j,k,0} \oplus x_{l',0} + d_{39,j,k,1} \oplus x_{l',1} + \ldots + d_{39,j,k,7} \oplus x_{l'7} \end{cases} \tag{5.17}$$

In general, we can solve both these systems of equations as long as we know all $d_{i,j,k,n}$ for a fixed round k, by simply using a brute force search in the value m_l. Note that since we are performing a Simple Power Attack, careful readings[5] will allow us to obtain the values of $\mathrm{H}[v_{ij(k-1)}]$ for all i,j,k. The search only takes 2^8 different tries with each taking at most 20 equation evaluations. We simply iterate m_l from 0 to 255 and evaluate by xoring it with $w_{i,j,k}$ and checking against $\mathrm{H}[v_{i,j,(k-1)}]$ for each $i \in \{0,2,\ldots 38\}$. To crack m'_l the same procedure is performed, except that this time i moves in the range $i \in \{1,3,\ldots 39\}$. As soon as one equation is not met, we try the next possible value of m_l. Solving for a fixed j,k produces two keys m_l and $m'_l = m_{l+4}$ due to the even and odd scheme shown before. The algorithm employed for the brute force search is outlined in Algorithm 5.1.

While $\mathrm{H}[v_{ij(k-1)}]$ values are easy to derive from the execution of the algorithm, bit values $d_{i,j,k,n}$ will generally be unknown due to the dependence between rounds. Nevertheless, we can compute the values of $d_{i,j,k,n}$ for $k = R$. The values of $v_{i,j,R}$ are predetermined (Equation (5.6)) and $Q_{jk}[x]$ is a known fixed permutation so we can derive the values of $w_{i,j,R}$, thus knowing all the binary variables $d_{i,j,R,n}$. We can then solve both systems of equations for $k = R$ obtaining the 8 most significant bytes of the key.

By cracking the most significant 64 bits of the key we have obtained enough information to calculate the values of v in the next round ($v_{i,j,(R-1)}$) via equation (5.8). Using the appropriate S-boxes (Equation (5.7)) we can also resolve the values of w in the next round ($w_{i,j,(R-1)}$). Since we know $\mathrm{H}[v_{ij(R-2)}]$ from the measurements and we just derived $w_{i,j,(R-1)}$ we will have all $d_{i,j,R-1,n}$. So we can solve the systems of equations for $k = R-1$ getting the following 8 bytes

[5]For now we assume perfect readings, later in the chapter the topic of noise will be covered

Algorithm 5.1 Guess Key Byte

```
Require: H[$v_{i,j,(k-1)}$]    $\forall i = 0,1,\dots 39$
Require: $w_{i,j,k}$    $\forall i = 0,1,\dots 39$
 1: procedure BreakKeyByte(H[$v_{i,j,(k-1)}$],$w_{i,j,k}$)
 2:     for $m_l = 0$ to 255 do
 3:         Valid $\leftarrow$ True
 4:         for $i = 0$ to 38 step 2 do
 5:             if H[$v_{i,j,(k-1)}$] $\neq$ H[$w_{i,j,k} \oplus m_l$] then
 6:                 Valid $\leftarrow$ False
 7:                 break
 8:         if Valid = True then
 9:             break
10:     for $m_{l+4} = 0$ to 255 do
11:         Valid $\leftarrow$ True
12:         for $i = 1$ to 39 step 2 do
13:             if H[$v_{i,j,(k-1)}$] $\neq$ H[$w_{i,j,k} \oplus m_{l+4}$] then
14:                 Valid $\leftarrow$ False
15:                 break
16:         if Valid = True then
17:             break
18: return $(m_l, m_{l+4})$
```

of the key.

By applying this scheme repeatedly we can successfully crack the whole key independently of its size with a quite narrow search space. Given $8R$ different bytes and a brute force search of at most 20×2^8, and with $R \leq 4$ the search space is upper bounded by

$$(20 \cdot 2^8) \cdot (8 \cdot 4) < 2^{18} = 262144 \tag{5.18}$$

It is important to notice that the system of equations is generally overdetermined and if the Hamming values are unequivocally measured, then a compatible solution will always exist. However, there is no guarantee of the system not being underdetermined. If only 7 or fewer of the 20 equations are linearly independent, multiple solutions may be possible and the described search will return the lowest one. Nevertheless, since S-boxes are designed for diffusion and we have 20 equations, the probability of this event happening is extremely unlikely[6].

Furthermore, the modifications that are introduced later to cope with the presence of error can easily resolve this situation, so no further elaboration is needed regarding the possibility of the system of equations to be underdetermined.

[6]It is most likely impossible given the definition of the S-boxes $q0$ and $q1$. However no proof is given

5.3 Attack in the Presence of Noise

Unfortunately, smart card attacks usually involve an amount of random noise overlapped with the signal. The work of Mayer-Sommer suggests that we can use correlation measures to determine Hamming weights [MS00]. However, the key schedule has a significant amount of redundancy, so we can directly shield the algorithm from noise without statistical measures, simplifying the attack and increasing the accuracy using the insights of the key schedule workings.

Adding noise in our measurements will mean that the values that we assumed we knew unequivocally have now added a degree of uncertainty. We can express the measured Hamming weight with added measurement error of v_{ijk} as follows.

$$\mathrm{H}\left[v_{ijk}\right] + \epsilon \qquad \epsilon \leftarrow \mathcal{N}(0, \sigma^2) \tag{5.19}$$

Where ϵ is a random Gaussian variable with zero mean and variance σ^2. Gaussian noise has been considered because even if ϵ is not a random variable, by the central limit theorem, repeated measurements over time should render a Gaussian distribution. Zero mean should be a consequence of the tuning procedure used when measuring the values, since a calibration that rendered zero mean expected error should minimize the measured error.

5.3.1 Least Mean Squares

Since the quantity measured is now real, and our algorithm worked with purely integer values, a first good step would be rounding to the nearest integer. Although not ideal, this technique will correct a significant number of our measurements. Due to the Gaussian shape of the distribution, $\epsilon < 0.5$ will hold for a significant amount of samples even for high values of σ. Empirical evidence backs this assertion since byte accuracies were significantly higher when including this preprocessing step, which seems to confirm our hypothesis.

Therefore, we can define a modified Hamming function as follows, where the term $\mathrm{H}[x] + \epsilon$ reflects the measure value as a whole with both the original value and the added error ϵ. The operator $\{\cdot\} : \mathbb{R} \to \mathbb{N}$ represents the nearest integer.

$$\mathrm{H}^*_\epsilon(x) = \{\mathrm{H}[x] + \epsilon\} \tag{5.20}$$

Rounding the variables to the closest integer makes the systems of equations incompatible with a extremely high probability, since they are 20×8 in size. A very common way to deal with this problem is just applying the Least Mean Squares closed form solution to get an approximation of the values. However, the systems of equations presented in Eq. (5.16), (5.17) are not linear, since they are using XORs.

We can find a way of expressing the previous system as a system of linear equations. We can start by considering that the terms of the form $\alpha_{ij} \oplus x_i$ can be decomposed as follows

$$\alpha_{ij} \oplus x_i = \begin{cases} x_i & \text{if} \quad \alpha_{ij} = 0 \\ \overline{x}_i = 1 - x_i & \text{if} \quad \alpha_{ij} = 1 \end{cases} \tag{5.21}$$

By substituting the displayed expression in a arbitrary equation that follows the xor pattern shown in the systems of equations we get:

$$\begin{aligned} b &= \sum_{i=0}^{7} \alpha_i \oplus x_i \\ b &= \sum_{\substack{i=0\\ \alpha_i=0}}^{7} x_i + \sum_{\substack{i=0\\ \alpha_i=1}}^{7} 1 - x_i \\ b &= \mathrm{H}[\alpha_i] + \sum_{\substack{i=0\\ \alpha_i=0}}^{7} x_i + \sum_{\substack{i=0\\ \alpha_i=1}}^{7} -x_i \\ b - \mathrm{H}[\alpha_i] &= \sum_{i=0}^{7} (-1)^{\alpha_i} x_i \end{aligned} \tag{5.22}$$

We can make use of this simplification into the previous systems of equations (Eq. (5.16) amd (5.17)) and get a linear system of equations, where $a_{ijkn} = (-1)^{d_{ijkn}}$. We use $(-1)^x$ simply as a mapping $\{0,1\} \to \{1,-1\}$. We can show the translated formulation for a fixed j,k in Equations (5.23) and (5.24).

$$\begin{cases} \mathrm{H}^*_\epsilon\Big(v_{0,j,(k-1)}\Big) - \mathrm{H}\Big[w_{0,j,k}\Big] = a_{0,j,k,0} \cdot x_{l,0} + a_{0,j,k,1} \cdot x_{l,1} + \ldots + a_{0,j,k,7} \cdot x_{l,7} \\ \mathrm{H}^*_\epsilon\Big(v_{2,j,(k-1)}\Big) - \mathrm{H}\Big[w_{2,j,k}\Big] = a_{2,j,k,0} \cdot x_{l,0} + a_{2,j,k,1} \cdot x_{l,1} + \ldots + a_{2,j,k,7} \cdot x_{l,7} \\ \mathrm{H}^*_\epsilon\Big(v_{4,j,(k-1)}\Big) - \mathrm{H}\Big[w_{4,j,k}\Big] = a_{4,j,k,0} \cdot x_{l,0} + a_{4,j,k,1} \cdot x_{l,1} + \ldots + a_{4,j,k,7} \cdot x_{l,7} \\ \ldots \\ \mathrm{H}^*_\epsilon\Big(v_{38,j,(k-1)}\Big) - \mathrm{H}\Big[w_{38,j,k}\Big] = a_{38,j,k,0} \cdot x_{l,0} + a_{38,j,k,1} \cdot x_{l,1} + \ldots + a_{38,j,R,7} \cdot x_{l,7} \end{cases} \tag{5.23}$$

$$\begin{cases} \mathrm{H}^*_\epsilon\Big(v_{1,j,(k-1)}\Big) - \mathrm{H}\Big[w_{1,j,k}\Big] = a_{1,j,k,0} \cdot x_{l',0} + a_{1,j,k,1} \cdot x_{l',1} + \ldots + a_{1,j,k,7} \cdot x_{l',7} \\ \mathrm{H}^*_\epsilon\Big(v_{3,j,(k-1)}\Big) - \mathrm{H}\Big[w_{3,j,k}\Big] = a_{3,j,k,0} \cdot x_{l',0} + a_{3,j,k,1} \cdot x_{l',1} + \ldots + a_{3,j,k,7} \cdot x_{l',7} \\ \mathrm{H}^*_\epsilon\Big(v_{5,j,(k-1)}\Big) - \mathrm{H}\Big[w_{5,j,k}\Big] = a_{5,j,k,0} \cdot x_{l',0} + a_{5,j,k,1} \cdot x_{l',1} + \ldots + a_{5,j,k,7} \cdot x_{l',7} \\ \ldots \\ \mathrm{H}^*_\epsilon\Big(v_{39,j,(k-1)}\Big) - \mathrm{H}\Big[w_{39,j,k}\Big] = a_{39,j,k,0} \cdot x_{l',0} + a_{39,j,k,1} \cdot x_{l',1} + \ldots + a_{39,j,k,7} \cdot x_{l',7} \end{cases} \tag{5.24}$$

Since we can express the system with linear equations, we can apply the least mean squares closed form solution.

$$\begin{aligned} A_{jk}\,\overline{x}_l^* &= \overline{h}_{j(k-1)} \\ \overline{x}_l^* &= (A_{jk}^\top\,A_{jk})^{-1}A_{jk}^\top\,\overline{h}_{j(k-1)} \end{aligned} \tag{5.25}$$

However, the vector $\overline{x}_l^*$ will be real valued so it needs to be mapped to $\{0,1\}$, so a simple mapping $\mathbb{R}\to\{0,1\}$ is performed.

$$x_{l,n} = \max\Big(0, \min\Big(1, \Big\{x_{l,n}^*\Big\}\Big)\Big) \tag{5.26}$$

Solving the equations and applying the transformation shown in Equation (5.26) gives a relatively good performance for small variances σ^2. The main problem arises from forward propagating errors. If we make a mistake predicting a byte of the key m_l all the bytes in that row $m_{l-8}, m_{l-16}, \ldots$ will also be affected. Thus if we have a probability p_1 of making a mistake solving a specific system, the probability of making at least one mistake in that round will be $p_2 = 1-(1-p_1)^8$ since we guess 8 bytes per round. Furthermore, the probability of not making a mistake in any round out of $R-1$ rounds will go as

$$p_3 = (1-p_2)^{R-1} = (1-p_1)^{8(R-1)} \tag{5.27}$$

Compounding these two expressions reveals the sensitivity of the algorithm to error. As the key size increases, we can expect for the accuracy to decrease significantly.

5.3.2 Hamming Mask Search

In order to circumvent the problem outlined in the previous section, we can use the high redundancy built into the system to correct the mistakes made by the Least Mean Squares (LMS) approximation.

When we make a mistake solving for m_l it is because one or more of the bits $x_{l,n}$ are incorrect. By Occam's Razor one can easily see that the most common scenario is one incorrect bit, then two incorrect bits and so on and so forth. Thus, we would like to try to correct the possible the errors in order of increasing complexity.

We can solve this issue by ordering the set of integers by Hamming weight as follows

$$\mathrm{H_M} = \{0,1,2,4,8,16,32,64,128,3,5,9,\ldots,254,255\} \tag{5.28}$$

Note that this is not the only possible order and there are multiple other identically good orderings as well. Since bit errors are assumed to be equally likely in all bits [7], the ordering of the integers with equal Hamming weight is arbitrary.

Given this order, we can xor these masks to toggle one or several bits of the estimated key

[7]This is a reasonable assumption given the confusion and diffusion techniques employed in the cipher and the non-linearity of the S-boxes

m_l producing the set of candidate keys ordered by Hamming distance to the original estimate.

$$\mathrm{M}_l = \{m_l \oplus h_\mathrm{M} : \quad \forall h_\mathrm{M} \in \mathrm{H}_\mathrm{M}\} \tag{5.29}$$

However we need a measure of how good is the fit of an arbitrary $m_l' \in \mathrm{M}_l$ with respect to other candidates in the set. A good approach is to minimize the sum of Hamming distances of the predicted values $\mathrm{H}\Big[w_{i,j,k} \oplus m_l'\Big]$ after xoring with m_l' and the rounded measured Hamming weights $\mathrm{H}_\epsilon^*\Big(v_{i,j,(k-1)}\Big)$. Ideally both this quantities should agree and since we have 20 equations per byte of the key m_l we could expect to find a good estimate.

$$m_l^* = \underset{m_l' \in \mathrm{M}_l}{\mathrm{argmin}} \left\{ \sum_{i=0}^{19} \left| \mathrm{H}\Big[w_{i,j,k} \oplus m_l'\Big] - \mathrm{H}_\epsilon^*\Big(v_{i,j,(k-1)}\Big) \right| \right\} \tag{5.30}$$

However, this approach ignores the complexity of the error making as likely to have no errors as to have eight. The results show overfitting of the key to this particular measure by getting a mask with a unnecessary large Hamming weight just because it renders the smallest value. In order to compensate one could add a normalization term with $\mathrm{H}[m_l']$ and some hyperparamter λ and carefully adjust the penalty of the key.

$$m_l^* = \underset{m_l' \in \mathrm{M}_l}{\mathrm{argmin}} \left\{ \lambda \cdot \mathrm{H}[m_l'] + \sum_{i=0}^{19} \left| \mathrm{H}\Big[w_{i,j,k} \oplus m_l'\Big] - \mathrm{H}_\epsilon^*\Big(v_{i,j,(k-1)}\Big) \right| \right\} \tag{5.31}$$

Nevertheless, since part of the algorithm involves computing the output variables of the S-boxes, we can use the non-linearity built into the S-boxes and minimize the sum of Hamming distances both in the input and the output of the substitution. In order to include the output Hamming distances, we will need to compare the predicted values $\mathrm{H}\Big[Q_{jk}\Big(w_{i,j,k} \oplus m_l'\Big)\Big]$ with the rounded measured values $\mathrm{H}_\epsilon^*\Big(w_{i,j,(k-1)}\Big)$. Until now we only needed the trace of Hamming weights of $\boldsymbol{V}$, but to implement this correction the trace of $\boldsymbol{W}$ will also be needed.

$$m_l^* = \underset{m_l' \in \mathrm{M}_l}{\mathrm{argmin}} \left\{ \sum_{i=0}^{19} \left| \mathrm{H}\Big[w_{i,j,k} \oplus m_l'\Big] - \mathrm{H}_\epsilon^*\Big(v_{i,j,(k-1)}\Big) \right| + \lambda \sum_{i=0}^{19} \left| \mathrm{H}\Big[Q_{jk}\Big(w_{i,j,k} \oplus m_l'\Big)\Big] - \mathrm{H}_\epsilon^*\Big(w_{i,j,(k-1)}\Big) \right| \right\} \tag{5.32}$$

The algorithm was implemented for a tradeoff $\lambda = 1$ and the accuracy of the algorithm was significantly improved under the presence of error, even with high variance σ^2.

The main issue with this approach is caused by how computationally expensive becomes optimizing this function. In order to simplify it, we can restrict these masks to have a maximum Hamming weight. For example, if we only consider the mask with weight smaller or equal to two we reduce the space of possible masks from 255 to 37. Therefore, the subset of masks with Hamming weights smaller than a threshold τ is defined as follows:

$$\mathrm{M}_l(\tau) = \{m_l \oplus h_\mathrm{M} : \quad \mathrm{H}[h_\mathrm{M}] < \tau \quad \forall h_\mathrm{M} \in H_\mathrm{M}\} \tag{5.33}$$

The size of this set does not grow linearly, due to the combinatorial terms. In general increasing τ means that we can resolve larger amounts of error, but the procedure becomes more computationally expensive as we show in the numerical results in the next chapter.

$$|\mathrm{M}_l(\tau)| = \sum_{n=0}^{\tau} \binom{8}{n} \tag{5.34}$$

$$\{|\mathrm{M}_l(\tau)| : \quad \forall \tau = 0,1,\ldots,8\} = \{0, 1, 9, 37, 93, 163, 219, 247, 255, 256\} \tag{5.35}$$

In summary, for each byte we first solve an overdetermined system of equations using Least Mean Squares and then we optimize the expression shown in Equation (5.32). In order to speed up the computation and discard unlikely candidate masks we can limit ourselves to masks with Hamming weight τ or less. There appears to be a tradeoff between time complexity and error correction that will be explored using empirical results in the next chapter. The overall structure of the attack is summarized in Figure 5.4.

As a final note, it is important to mention that for the attack to be successful we need to guarantee that both power traces of $\boldsymbol{V}$ and $\boldsymbol{W}$ can be obtained from the execution of the smart card implementation. Otherwise, we would lack the information to solve the systems of equations. Luckily, looking at the TwoFish Source Code[8], we can corroborate that the values before and after the S-boxes are processed. Even tough the original description of $q0$ and $q1$ originally introduced in [Tsc] used an procedural formulation with byte arithmetic, the S-boxes are effectively implemented with hard coded look up tables. Once we have checked the availability of desired information in the power readings we can proceed and implement the attack to evaluate its performance as we will see in Chapter 6.

[8] *TwoFish Source Code*. 1998. URL: `https://www.schneier.com/cryptography/twofish/download.html` (visited on 03/01/2016).

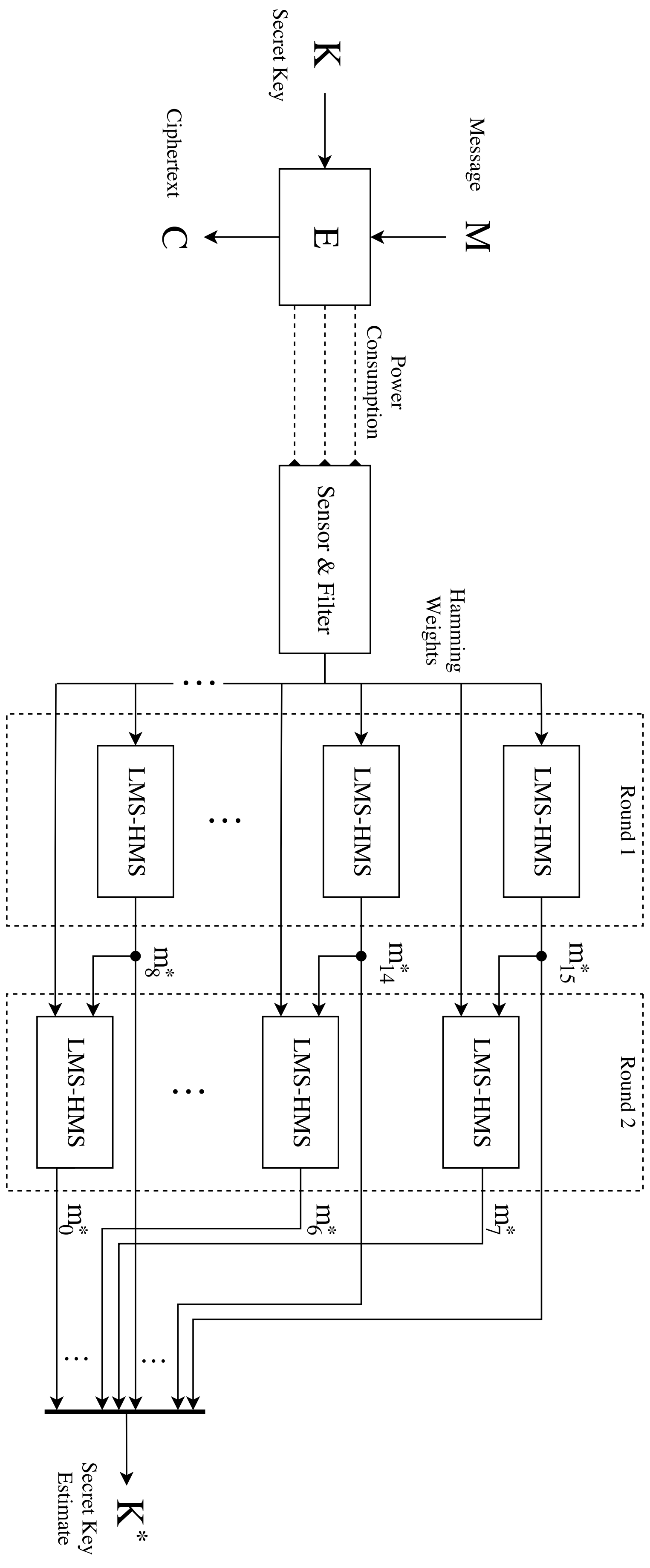


Figure 5.4: Block diagram of the error noise TwoFish SPA attack (128-bit key)

5.4 Multiple Readings

5.4.1 Clustering Keys

The analysis so far was done considering just a single reading of the execution of the algorithm. Nevertheless, in real life it is not far-fetched to obtain multiple albeit not many readings that use the same secret key. In general, the most difficult part consists on identifying which reading is associated with each secret key. However, given an algorithm as the one outlined in Section 5.3.2, we can easily cluster the readings using some kind of similarity measure between the sequences. Direct statistical analysis such as Pearson correlation would not take into account the order of the sequence which is of extreme relevance in this case. Therefore, we can represent the key estimates as elements of $\mathbb{Z}_{256}^{N/8} \quad N/8 = \{16, 24, 32\}$ and compute the euclidean distance between them. The cardinality of the set $\mathbb{Z}_{256}^{N/8}$ is the same as $\mathbb{Z}_2^N$, so we should expect that two randomly sampled different keys would have a small euclidean distance.

$$\left\|K - K'\right\|_2 = \sqrt{\sum_{i=1}^{N/8} (K_i - K_i')^2} \qquad K, K' \in \mathbb{Z}_{256}^{N/8} \tag{5.36}$$

However, by using euclidean distance we are penalizing in an non uniform way. Mistakes that appear in more significant bytes will be penalized more heavily and vice versa; bit position should not affect the distance that is obtained. Thus, in order to correct for it, we can simply aggregate the number of mistakes performed between the two keys. If we do simple aggregation we can see that the distance metric is just the Hamming distance $\mathrm{H} : \{0,1\}^n \times \{0,1\}^m \to \mathbb{N}$ of two binary variables x, y is defined as

$$\mathrm{H}(x, y) = \mathrm{H}[a \oplus y] \tag{5.37}$$

$$\left|K - K'\right|_{\mathrm{H}} = \mathrm{H}\Big(K, K'\Big) = \mathrm{H}\Big[K \oplus K'\Big] \tag{5.38}$$

Nevertheless, the Hamming distance has a linear penalty. If we had twice the amount of mistakes, the penalty would be doubled. Although we could apply a polynomial function $f(\cdot)$ to the previous expression, it would still ignore the fact that mistakes are introduced by $N/8$ independent byte estimates. To get a compromise between these two metrics we can introduce of a Hamming norm.

The q-Hamming norm is defined as a mapping $\|\cdot\|_{\mathrm{H}^q} : \{0,1\}^N \to \mathbb{N}$, that penalizes each byte mistakes independently, as (5.39) shows.

$$\left\|K - K'\right\|_{\mathrm{H}^q} = \sum_{i=1}^{N/8} \mathrm{H}\Big(K_i, K_i'\Big)^q \tag{5.39}$$

This q-Hamming norm will be a significantly better distance metric between different key guesses, effectively discriminating between different key clusters. The parameter q will control the tradeoff between individual byte differences and the overall difference of the two byte vectors. We can know reiterate the argument of the cardinality of $\mathbb{Z}_2^N$ is designed to be for a computational search to be intractable. Thus, we do not expect independently sampled keys to have small q-Hamming distances.

In a real world scenario, our eavesdropper device would get a set of power readings. For each one of them we would get a different key estimate. In order to identify those clusters, a simple technique like a threshold distance can be used or if there is a need for a more accurate detection of them based on eigenvector analysis, spectral clustering could be used. If the cluster algorithm requires an affinity $\alpha \in \{0,1\}$, we can simply kernelize the norm (5.40) and explore different kernel widths σ.

$$\alpha(K,K';q,\sigma) = \exp\left(\frac{\|K-K'\|_{\mathrm{H}^q}}{\sigma^2}\right) \tag{5.40}$$

Thus, we would get a series of clusters, one for each different key that the eavesdropper would have observed.

5.5 Combining Key Predictions

Once we are able to identify key readings accurately, there remains the task of using multiple readings from the same key to correct for any errors that the algorithm described in the previous section could not account for. As we have previously stated, the estimates for the individual bytes of the key within each round are independent of each other. Namely, a byte m_i will only depend on bytes from previous rounds $m_{i+8}, m_{i+16}, \ldots$. We can see that when making a mistake in m_i, it is going to propagate through the algorithm and affect the sequence $m_i, m_{i-8}, \ldots$. As we saw in Figure 5.3, there are 8 different sets of rows in the key schedule, so the probability of making a mistake in the same byte in two different readings is smaller than making it in any other of the seven rows. Consequently, the probability of making a mistake in the same byte using both Least Mean Squares plus Hamming Mask Search and outputting the same result for a particular byte is extremely slim.

Therefore, if we had access to multiple power traces associated with different readings we would be able to correct for these simple mistakes by just performing a majority vote in the byte level. Our approach treated the multiple readings independently and to get the final estimate of the key just took the most common value along the predicted bytes for that particular position. As long as we have at least two predicted keys that share the same byte estimate for each byte, we can be quite confident that the predicted key will be the original one.

Finally, if for a specific byte the majority vote does not output any value, namely that all byte estimates are equally likely, we could still compare which was the error for each Least Min Squares estimate, what was the weight of the hamming mask employed in the correction and which error did the objective function (5.32) achieve. Comparing these values, and assuming that the least complexity scenario is the most likely one, we would be able to tell which byte is the best candidate.

Chapter 6

Results

"The magic words are squeamish ossifrage"

– Plaintext of the message encoded in RSA-129

NUMERICAL results for accuracies and runtimes are presented in this chapter for the different algorithms outlined in Chapter 5. Results are both presented in terms of tables and figures so the reader can both refer to specific accuracies whilst visualizing the general trends for the different key sizes employing the provided figures.

Experiments are always run for the three key sizes for which TwoFish was designed to operate ($N = \{128, 192, 256\}$) since as we analyzed in Chapter 5, the more rounds we have in the h transformation the easier is to propagate errors and worse will be the performance of the attack.

In order to extract statistical significance of the experiments performed a sample size of 1000 keys was used for each algorithm and possible parameter combination. A thousand sample size was chosen empirically; smaller sample sizes resulted in variations on the accuracy of some of the combinations for the standard deviation of noise and size of the mask correction. On the other hand, higher sample sizes did increase the precision of the results, but also increased significantly the amount of time to recreate the results. Moreover, the sample size matters the most when high quantities of noise are overimposed to the signal since there algorithm has less information to work with and will be less predictable. Nevertheless, 1000 keys was chosen nonetheless for all the cases in order for the accuracies to be statistically comparable.

The specific details of the implementation of the different approaches can be found in Appendix B with corresponding descriptions of the different libraries and scripts. As it was already mentioned in a previous chapter, all the implementations were done in Python 2.7.11 (64-bit architecture) and run using a single core at 2.6 GHz. It is important to note that in order to speed up the simulations, the Least Mean Squares algorithm was included from the `numpy` library since it defines a low level implementation of the algorithm which executes significantly faster than a pure Python implementation ever could. Moreover, as we shall see later in Section 6.2, computation times grow a couple orders of magnitude as we introduce error correction techniques into the algorithm. Therefore, in order to speed up the calculation, analogous simulations were run in parallel employing close to identical processors; allowing us to still compare runtime results.

6.1 Perfect Hamming Trace

The results for the direct approach that assumes an integer Hamming trace with no errors, are shown in Table 6.1. Due to the extremely good performance of the algorithm more samples were drawn but the algorithm did not produce any faulty key. This confirms the hypotheses presented in the previous chapter, since we can use a simple Brute Force Search in order to get every single byte of the key.

Key Size	Accuracy	Avg. Runtime
128	100%	3.75 ms
192	100%	5.7 ms
256	100%	7.39 ms

Table 6.1: Accuracies of the algorithm for a thousand random keys per key size.

Regarding the runtime we can see that for every Hamming trace that apply the attack to, the elapsed times is in the order of magnitude of milliseconds rendering the attack feasible regarding the runtime and required computation resources. In order for the attack to be practical we do still need to corroborate that we can shield its prediction from a significant amount of noise.

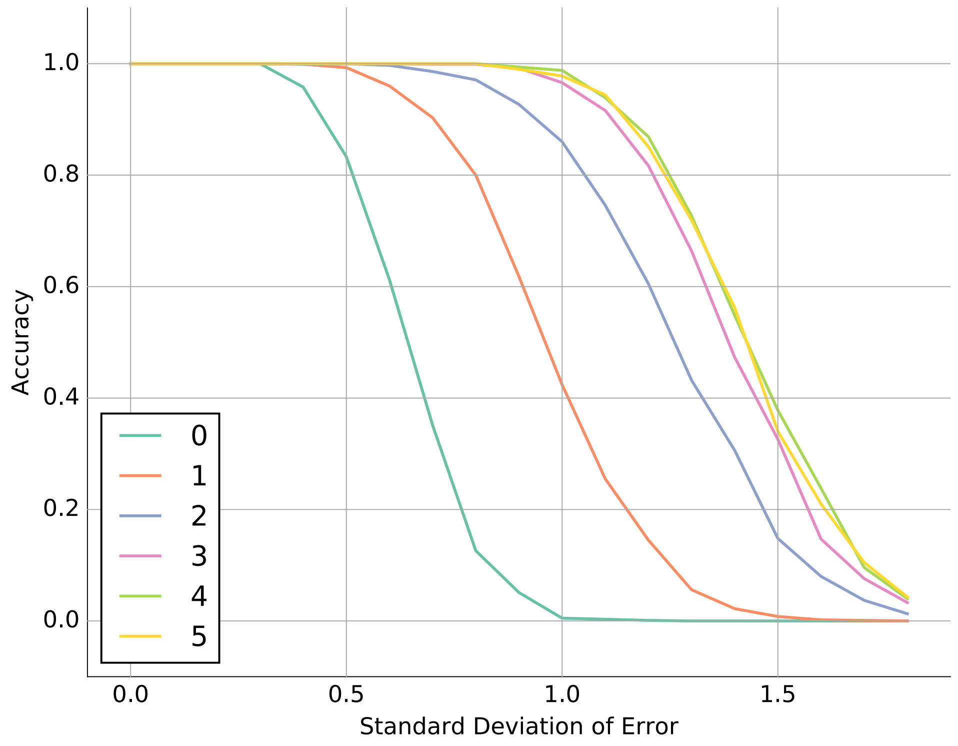
These results seem to confirm our intuition that the algorithm does perform flawlessly without error. This results makes sense when we think back of the overdetermined $\{+, \oplus\}$ system of equations, which should only have one solution. As we previously discussed, due to the sizes of $2^{128}, 2^{192}, 2^{256}$ there exists the possibility of some key producing an undetermined system for some key, but we can safely ignore that issue since it is extremely unlikely and can easily be accounted for using the noise correction techniques that we already described.

6.2 LMS-HMS Noise Correction

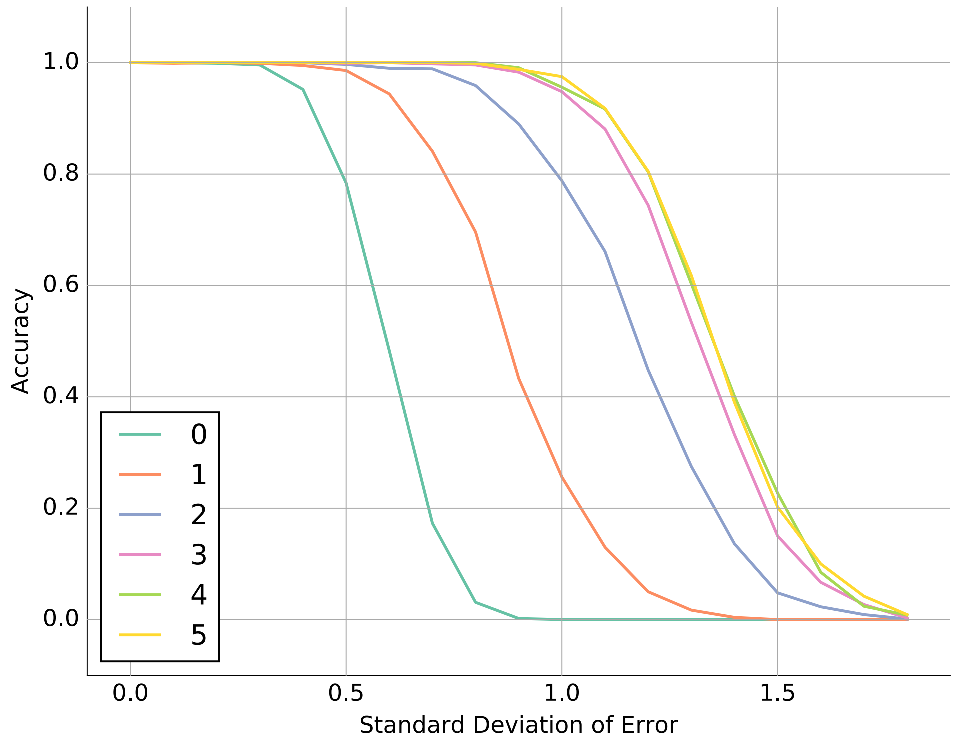
In order to accurately evaluate the performance of the algorithm that is able to correct the errors introduced by noise we need to compute the performance for several values of the standard deviation σ. Furthermore, as we stated previously, varying the maximum Hamming weight threshold τ of the masks used will result in a tradeoff between accuracy and runtime.

Therefore, a simulation of random keys was performed for values of $\sigma = 0.1, 0.2, \ldots, 2.0$ and $\tau = 0, 1, \ldots 8$. Note that for $\tau = 0$ the algorithm does not perform any sort of Hamming Mask Search and is equivalent to the one described in Section 5.3.1. This allow us to compare both the contributions of the LMS technique and the Hamming Mask Search (HMS) procedure as τ varies.

A thousand random runs of the algorithm per combination of σ and τ were performed. Table 6.2 shows average accuracies and runtimes for several σ and τ. Average runtime t does not depend on σ so it is aggregated in the last row. We can also visualize the results of the simulation for various key sizes in Figure 6.1.

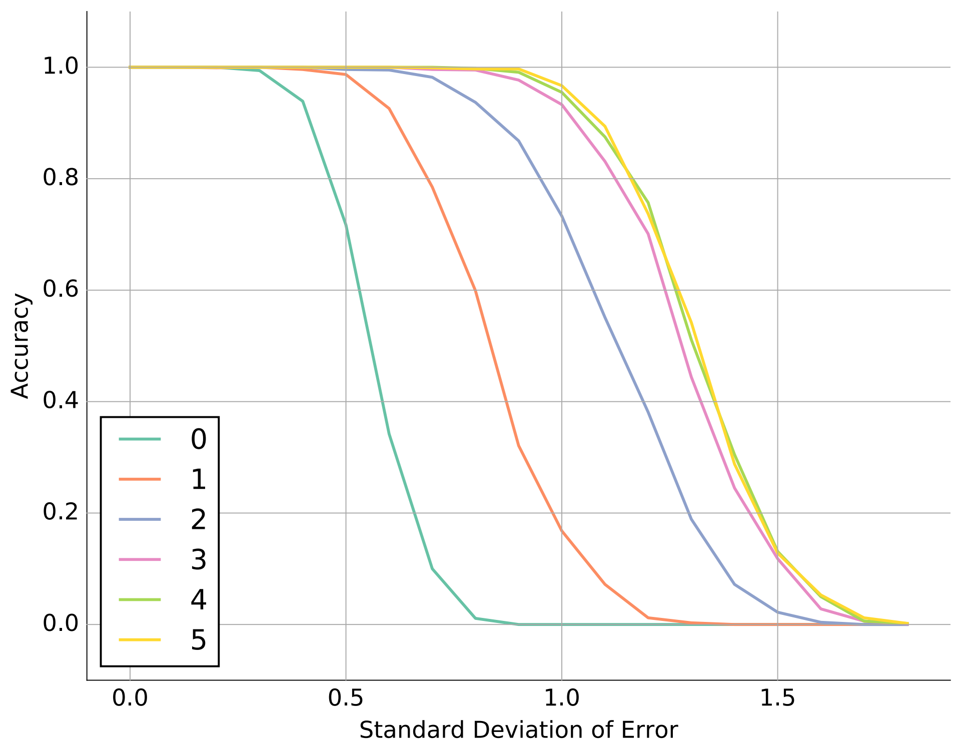


(a) Average accuracies for 128-bit key

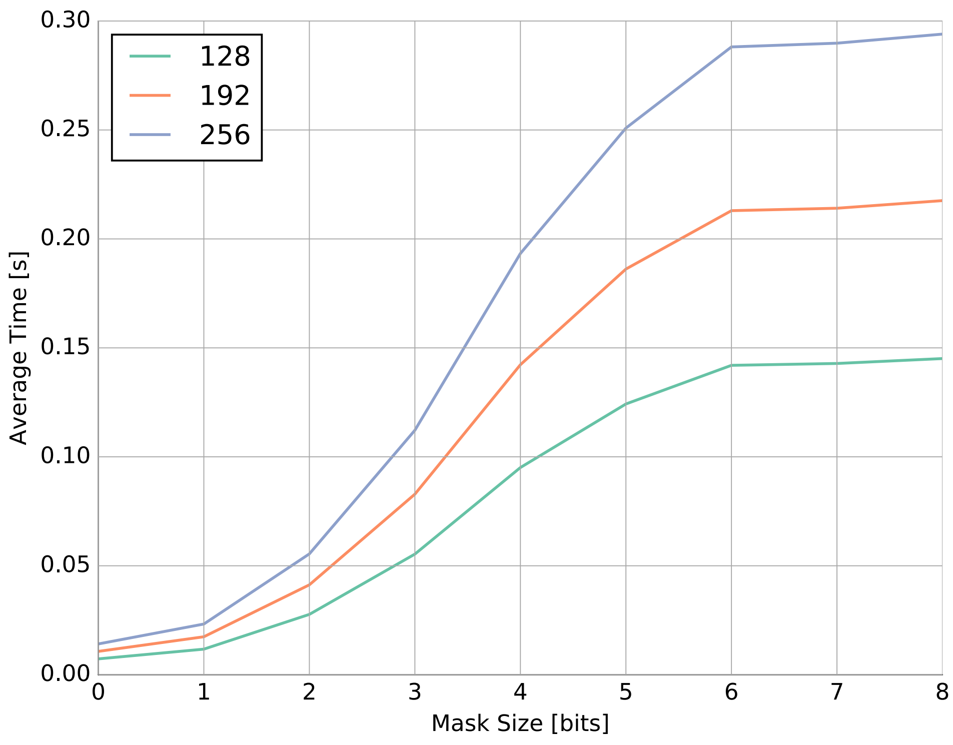


(b) Average accuracies for 192-bit key

Figure 6.1: Results of simulation of the LMS algorithm with correction



(c) Average accuracies for 256-bit key



(d) Average Runtimes for various mask sizes

Figure 6.1: Results of simulation of the LMS algorithm with correction

$\sigma \setminus \tau$	0	1	2	3	4	5	6	7	8
0.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.2	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.4	97.5	99.8	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.6	63.2	96.7	99.7	100.0	100.0	100.0	100.0	100.0	100.0
0.8	12.4	77.3	96.9	99.8	99.9	100.0	100.0	100.0	99.8
1.0	1.4	42.1	87.2	97.3	98.1	98.0	97.7	98.7	98.4
1.2	0.0	12.8	62.4	83.2	84.1	84.9	83.7	85.5	86.2
1.4	0.0	3.1	27.0	52.6	56.1	56.0	56.3	53.5	54.8
1.6	0.0	0.4	8.4	15.9	21.7	21.3	19.6	23.4	22.3
1.8	0.0	0.0	2.3	3.6	4.2	5.4	4.9	4.8	4.7
2.0	0.0	0.0	0.1	0.5	0.6	0.6	0.3	0.1	0.7
t(ms)	7.27	11.52	25.94	55.36	91.46	121.11	148.50	150.94	142.14

Table 6.2: Summarized accuracies of the algorithm for several σ and τ (128-bit key)

Note that the Table 6.2 only contains a summarized version of all the experiments that were performed and only contains accuracy values for the 128-bit key size. The rest of the results including 192 and 256-bit keys are throughly presented in Appendix A. These results were used for the plots in the Figure 6.1. As we will explain later some of the values (higher τ) have been omitted since they did not contribute any meaningful information.

From the values shown on the Table we can draw several conclusions. It is easy to see an almost diagonal pattern, since higher values of σ decrease the accuracy and higher values of τ increase it.

However, for values of $\tau > 3$ the improvement is negligible, whereas the runtime experiments a threefold increase. The times have multiplied from our original brute force search algorithm. For $\tau = 0$ (no correction performed) the runtime is doubled and the accuracy is poor for values $\sigma > 0.6$. Employing $\tau = 3$ as factor correction increases the runtime by a factor of 8. Nevertheless, increasing one order of magnitude the runtime almost completely shields the algorithm from Gaussian errors with standard deviation $\sigma < 1.0$, which is a more than considerable amount of noise.

Accompanying graphics depicting the performance for different key sizes, amounts of noise and mask sizes are included in Figure 6.1. From these Figures we can make a number of remarks. First, the results agree with the conclusions form the analysis of Table 6.2. Furthermore, since here can compare the performance for different key sizes we can observe that the higher the key size, the smaller the error tolerance of the algorithm. This was expected since as we commented previously, there exists a forward propagation of errors, so bigger key sizes will decrease the probability of cracking the key. In the figures we can also see that for $\tau \geq 3$ the improvement is almost negligible, so large values of τ have been omitted for clarity.

Regarding the average runtime as a function of the mask size we can see that this closely resembles the shape of the cumulative distribution function of a Gaussian distribution. This is produced from the combinatorial expression in Equation 5.34, confirming the intuition that the complexity is related to the size of $\mathrm{M}_l(\tau)$.

6.3 Multiple Readings

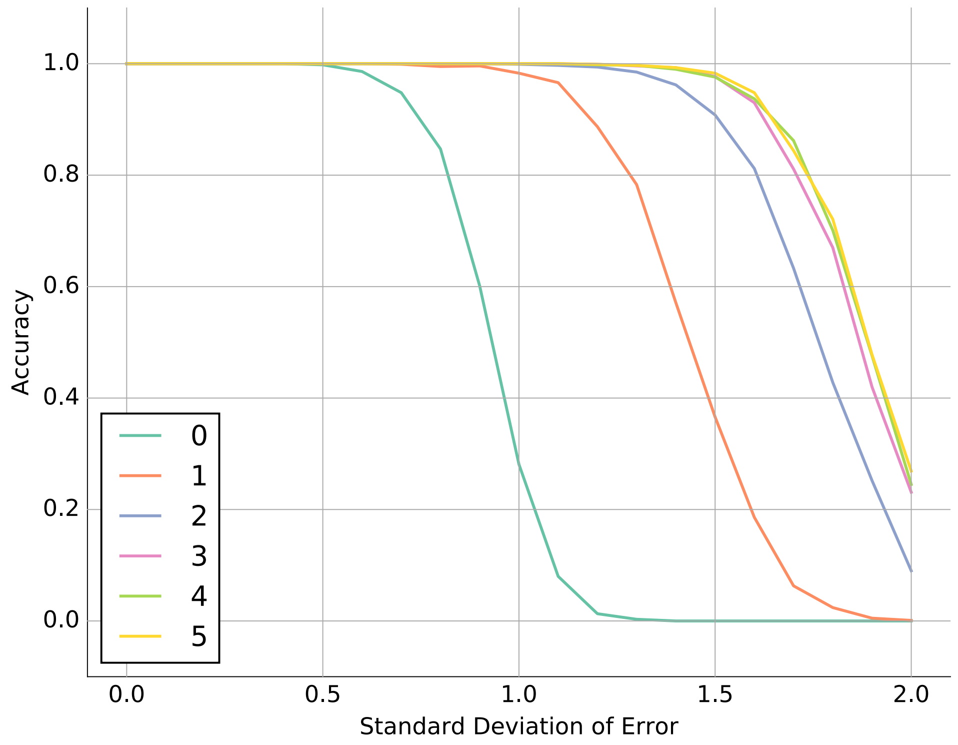
As we outlined in Section 5.4 if we observe multiple power traces from the same key we will be able to improve the error correction results for the algorithm. To analyze this behavior we performed a simulation that employed up to 5 readings per secret key. If the algorithm was able to obtain a majority vote for each byte of the key without ties it would return that key. Otherwise, it would execute a new simulation with that key up to a limit of 5 readings. We decided to use 5 readings as an upper limit since it gave significant results and it is small enough to be observed in a real world scenario.

Similarly we run a thousand random runs for varying values of the parameters σ and τ we run one thousand random simulations. Results can be seen in Table 6.3 for 128-bit key. We find the same pattern as in the previous table, except that in this case error tolerances have been significantly improved. As we saw before results are almost identical for $\tau \geq 4$. Accuracies have improved for all combinations of σ, τ at the expense of times increasing by an average of a factor of three.

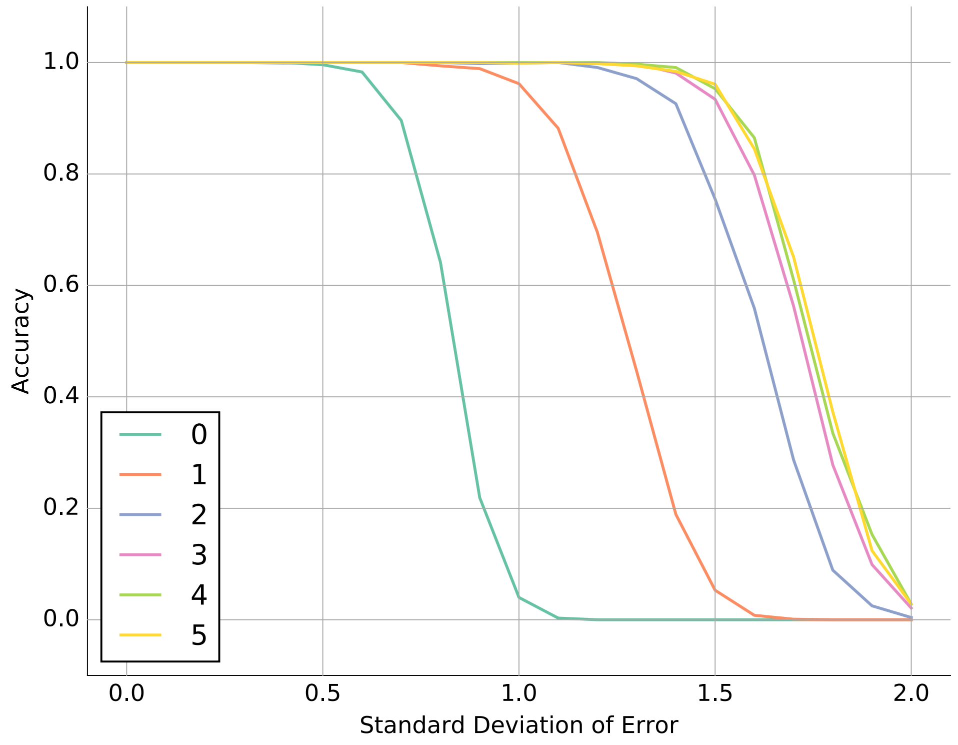
$\sigma \backslash \tau$	0	1	2	3	4	5	6	7	8
0.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.2	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.4	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.6	98.6	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.8	84.7	99.5	100.0	100.0	100.0	100.0	100.0	100.0	100.0
1.0	28.1	98.3	99.9	100.0	100.0	100.0	100.0	100.0	100.0
1.2	1.3	88.7	99.4	99.9	99.8	99.9	100.0	99.9	100.0
1.4	0.0	57.1	96.2	99.3	99.0	99.3	99.4	99.8	99.7
1.6	0.0	18.6	81.2	93.0	93.7	94.8	95.6	92.3	93.3
1.8	0.0	2.4	42.8	67.0	70.1	72.1	69.0	69.1	68.6
2.0	0.0	0.1	9.0	23.1	24.5	26.9	28.2	27.3	27.3
t(ms)	27.92	40.23	77.30	157.27	252.05	342.18	381.46	399.85	413.72

Table 6.3: Summarized accuracies of the algorithm with multiple readings per key (128-bit key)

Accompanying graphics are also included for this case, depicted in Figure 6.2. From the Figures we can see that for all key sizes $N = \{128, 192, 256\}$ we have improved the error tolerance from $\sigma \approx 1.0$ to $\sigma \approx 1.5$. The main difference when comparing these Figures to the ones in the previous section is the slope of the curve in the transition. Without multiple readings, the slope was fairly smooth, similar to that of a sigmoid function. Nevertheless, in the new graphics we observe that the transition curve is more abrupt, specially for 192 and 256 bit keys. This makes sense since we are introducing more layers of complexity to the error correction. The corrections make the algorithm more sensible to errors when it is unable to compensate for the amount of noise being added into the readings. Finally, for the time representation we have changed the grouping by mask size to standard deviation since now, depending on the amount of error the algorithm will do between two and five executions to determine the final key. Thus, we can see that for $\sigma \lesssim 1.0$ two executions are enough to correct all of the errors but for bigger values of the noise we start needing more executions to achieve a clear majority vote. Transition happens at $\sigma \approx 1.5$, which agrees with the accuracy results obtained in the rest of the Figures.

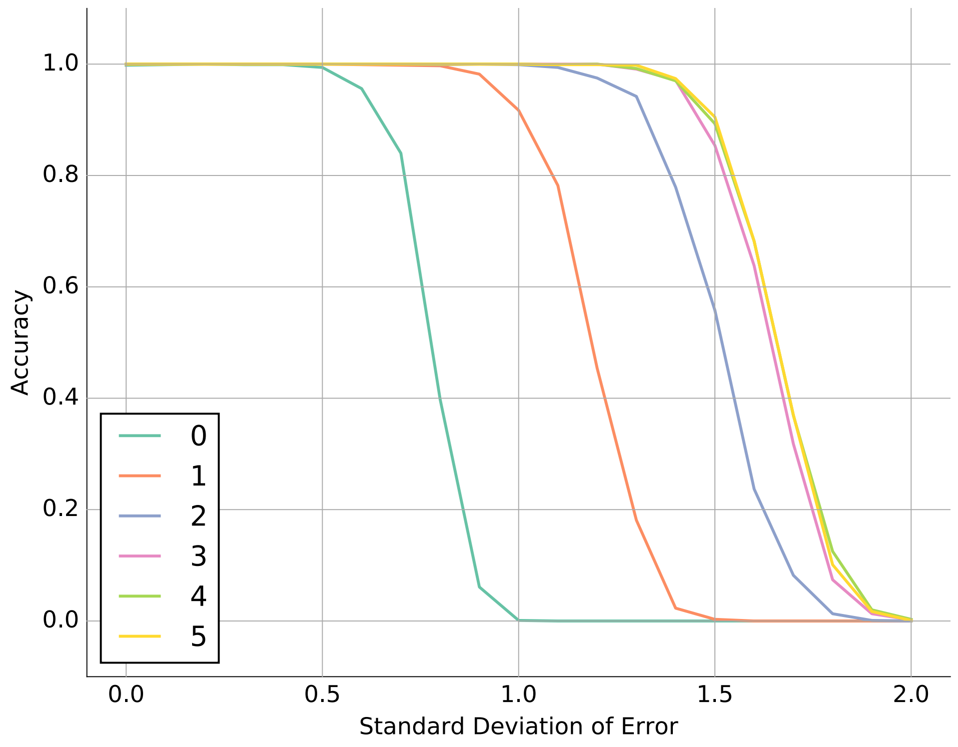


(a) Average accuracies for 128-bit key

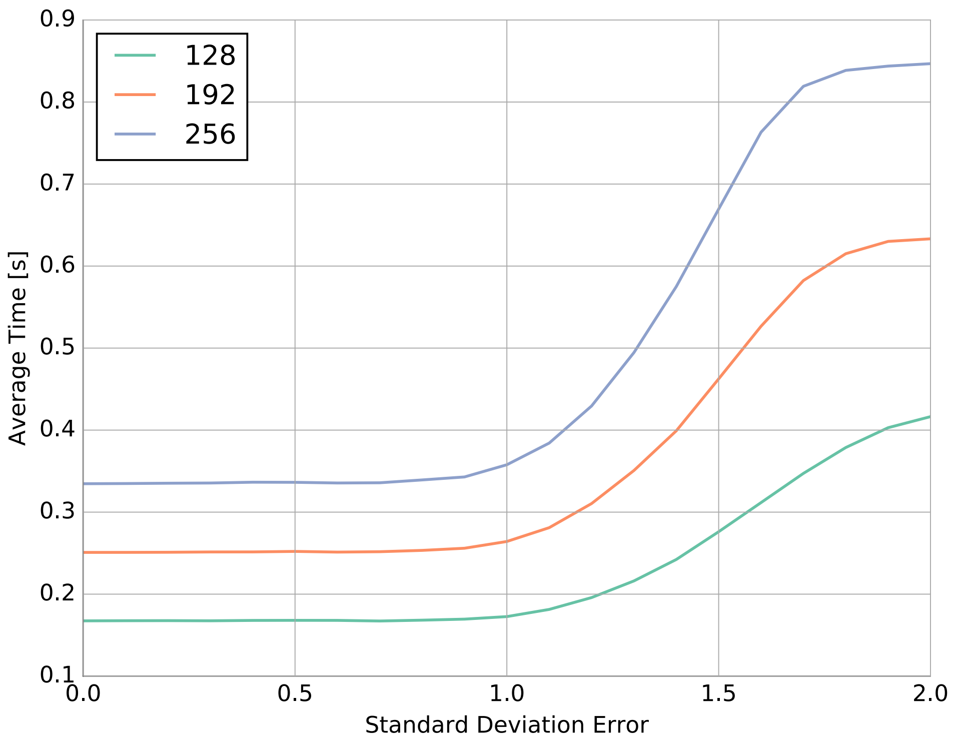


(b) Average accuracies for 192-bit key

Figure 6.2: Results of simulation with multiple readings per key



(c) Average accuracies for 256-bit key



(d) Average Runtimes for various noise levels

Figure 6.2: Results of simulation with multiple readings per key

6.4 Discussion

A main caveat of SPAs is that even they work in theory, they usually need a perfect Hamming trace of the execution. However, in practice we tend to experience a number of inconveniences, such as noise, uncorrelated measurements with the execution of the program and even schemes that masquerade any useful values. Therefore, a good SPA attack will not only work in theory but will also perform reasonably well under these handicaps.

As we already commented in the previous sections, obtaining a value correlated to the Hamming values has been proved to be possible to perform as shown in [MS00]. The TwoFish implementation for several languages is publicly available [Tsc]. Therefore, if we can run the algorithm step by step we will be able to establish the specific locations where the algorithm computes values whose Hamming weights are required for the described attack. Careful timing and some statistical analysis should resolve in the sought values when performing the attack, as [PMO07] has proved.

Among the implementations are a couple of 8-bit assembly languages (Motorola 6800 and Zilog Z80) as well as more complex implementations in 32-bit and 64-bit environments. The attack works directly on both 8-bit implementations since is inherently designed for 8-bit Hamming weights. Nevertheless, examining closely the x86 Assembly and C implementations we can see that the S-box is performed as a lookup table operation. Thus, words are shifted into bytes to perform this lookup so the values needed for the attack can also be found in these implementations. TwoFish's description of the h function with such an arbitrary arrangement of S-boxes makes extremely difficult and inefficient to program an implementation that does not get the individual 8-bit values to perform the S-box substitution. Thus, as long as we can get a reliable power analysis trace, any TwoFish implementation will most likely be vulnerable to the attack presented in this paper.

Most systems vulnerable to SPAs will not give a perfect power analysis trace, so we should expect a considerable amount of noise added to our measurements. As we saw in Section 5.3 and in Chapter 6, modifying the algorithm to perform under error is possible. Moreover, as the results displayed, the algorithm is quite robust to Gaussian noise, specially when we consider the results of [VBC05; CTV09] where the largest considered noise had $\sigma = 0.3$.

Comparing the accuracy graphics with the time plot, we can conclude that the best tradeoff accuracy-time happens at $\tau = 3$. This is probably linked to the fact that even with a high standard deviation is hard to have more than 3 bits consistently changed for the twenty bytes that we sample in the trace. Moreover, higher values of τ are not able to account for higher values of σ. Namely, for $\sigma \lesssim 1.0$, we can just increase the threshold weight τ and we will manage to get 100% accuracy. However, for values $\sigma \gtrsim 1.0$, increasing τ does not increase accuracy and in fact, for $\sigma = 2.0$ the algorithm almost always makes a mistake in at least one byte of the key.

Chapter 7

Conclusions and Future Work

"For the computer security community, the moral is obvious: if you are designing a system whose functions include providing evidence, it had better be able to withstand hostile review."

– Ross Anderson

THE main purpose of this document was to present an efficient and practical Simple Power Analysis Attack to the key schedule of the TwoFish block cipher. We have shown that the TwoFish Encryption Standard is in fact susceptible to a SPA that is based solely in the Hamming weights of the Key Schedule Computation; refuting *Biham* and *Shamir* [BS99] assertion that TwoFish was not susceptible to power attacks. The algorithm successfully obtains the value of the secret with just one run of the algorithm for low levels of noise. All the requirements described in Chapter 4 had been fulfilled, reassuring the success of the project.

The efficiency of the algorithm can be easily corroborated when examining at the execution times. The worst runtime of the algorithm is under one second, rendering the attack is not only feasible but also efficiently computable in modern real time systems.

Error correction was introduced with a combination of Least Mean Squares and Hamming Mask Search, acquiring more than significant shielding to high levels of noise; usually produced by interferences or measurement imprecisions. Noise tolerance was improved when using multiple power readings coming from the same secret key.

7.1 Contributions and Developments

To cope with the key discrimination task, novel techniques had to be developed using the prior knowledge of the single reading algorithm. It is important to note that these multireading techniques and derivations are not limited to the TwoFish cryptosystem, and can be easily extended to analogous SPAs where a approximate candidate key is obtained from a single power reading. The described approach gives a clever way of combining information from separate SPAs. No assumptions were made regarding on the sequence or amount of recordings, producing a simple yet useful framework, that as we showed with the TwoFish results, produces a significant performance improvement.

Additionally, we have also demonstrated that the attack threatens not only 8-bit implementations but most of TwoFish implementations, since the architecture of the cipher forces the key schedule to explicitly compute the byte values that are needed for the described attack. This result also generalizes to other ciphers that would use similar constructs to the ones TwoFish uses such as 8-bit key dependant S-boxes among others.

With the description of the attack comes not only the knowledge of what made TwoFish weak, but what types of constructs are vulnerable to SPAs and why. In a similar fashion to [CTV09], where the authors showed how the weakness of the Serpent key schedule was due to the use of LFSR; we have showed that some parts of the TwoFish description are susceptible to Hamming weight based power analysis. More specifically, as we saw in previous chapters, the weakness of the TwoFish key schedule came from a non-trivial to uncover linear relationship between the rounds of substitution boxes.

TwoFish used a number of xors and S-boxes in the specification of the key schedule. Note that both of these constructs are fundamental building blocks of modern cyptographic protocols. For instance, both of them can be found in Rijndael the current AES standard, which is omnipresent in our society. Exclusive ORs are useful because while revertible, they introduce confusion and diffusion into the ciphertext whereas S-boxes are needed for their inherent non-linearity. Namely, due to the possibility of linear attacks, ciphers need to be non-linear from end to end. However, as we saw in the attack description, the xors created a non-linear system of equations and when including some more information, like the Hamming weights of some intermediate variables, the system became linear and thus the solution was trivial to obtain.

On the other hand, the S-boxes, used for their high non-linearity provided a very useful scheme for noise correction, since they constrained even more the “degrees of freedom” of the possible solution. It is important for this kind of basic system interplay to be known, since it should be avoided when designing new ciphers as well as in the implementations they will require.

With the inclusion of TwoFish, three out of the five AES finalists have been found to be susceptible to SPAs. In all three of the finalists, the SPA focused on weaknesses in the key schedule and was performed using Hamming weight based algorithms. Nevertheless from these attacks we can learn lessons such as LFSR unsuitability or Hamming-XOR linear problem, allowing cryptographers to develop more secure and reliable cryptosystems. Moreover, these three (AES, Serpent and Twofish) were the better regarded ones by the cryptographic community due to the inherent unsuitabilities that the remaining two (MARS, RC6) displayed[1]. This sets an relevant precedent for the next generation of cryptographic standards to consider vulnerabilities not only from the theoretical formulation but from the specific implementation of the algorithm and the key schedule itself. Embedded systems are becoming more and more widespread and with the arrival of the *Internet of Things*, confidentiality and integrity of the information acquired and transmitted by these devices will be an crucial concern.

Lastly, if we compare the results of this work to the attack described in [VBC05], we can see that the results with no noise correction are comparable (16 ms AES vs 3.75 ms TwoFish) and the accuracies are both 100%. On the other hand when we look at $\sigma = 0.35$, all of our approaches outperform theirs both on time ($\sim$15 h AES vs <0.3 s TwoFish) and in accuracy (<95% AES vs 100% TwoFish). Moreover we are able to explore standard deviations up to five times larger without a loss of accuracy proving both the efficiency and the correctness of the attack.

[1] Sch+00.

7.2 Future Work

Due to three of the AES process finalist being susceptible to SPAs, it would be interesting to analyze the possibility of having similar SPA in the MARS and RC6 key schedules. If they were to be susceptible to similar SPAs it would set an unavoidable precedent for the next generation of cryptographic standards to consider. Whilst still secure, the ciphers developed for the AES contest are now almost two decades old; two decades of constant change and advance in the field of Information Technology and Security. A new standard is probably still years away but getting to know better what worked and what did not will help the cryptography community to make more reliable systems.

Moreover, regarding the followed approach a number of considerations for further extensions can be made. The first remark is that we did not use the entirety of the key related information. As we outlined in Section 5.1, each round of the algorithm uses the words S_i, which are derived applying a Reed Solomon Transformation to the secret key. From the Hamming trace of the matrix calculations, a more refined search could be done to correct for errors. Furthermore, in the optimization function we only considered the input and output of a single stage of substitution. A more global approach could consider the input and output of the subsequent stages and search for the minimum of those and backtrack as necessary. Nonetheless, since the runtime would increase in combinatoric way, a smart algorithm would be needed to just consider the most likely cases and prune said search.

To speedup the algorithm another extension can be made. As one can learn from the derivation of Hamming Mask Search, the technique completely ignores the internal configuration of the S-boxes $q0$ and $q1$. Both through the mathematical definition as well as using the values themselves, we should be able to reduce the threshold size τ dynamically depending on the input and output hamming weights, reducing the amount of masks to ever need.

Finally, it would be interesting to gather real world data and perform the attack on a real system in order to consider for the statistical inference problems related to find the appropriate values in trace as well as to confirm the conclusions derived in this work.

Bibliography

[Beh+11] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. “Cython: The best of both worlds”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 31–39.

[BS99] Eli Biham and Adi Shamir. “Power analysis of the key scheduling of the AES candidates”. In: *Proceedings of the second AES Candidate Conference*. 1999, pp. 115–121.

[BW15] Gaetano Borriello and Roy Want. “Embedded computation meets the world wide web”. In: *Communications of the ACM* 43.5 (2015), pp. 59–66.

[Com08] Wikimedia Commons. *Twofish diagram*. 2008. URL: `https://upload.wikimedia.org/wikipedia/commons/e/ee/Twofishalgo.svg` (visited on 07/10/2016).

[Com10] Wikimedia Commons. *Power Attack*. 2010. URL: `https://upload.wikimedia.org/wikipedia/commons/6/6c/Power_attack.png` (visited on 07/10/2016).

[CTV09] Kevin J. Compton, Brian Timm, and Joel VanLaven. “A Simple Power Analysis Attack on the Serpent Key Schedule”. In: *IACR Cryptology ePrint Archive* 2009 (2009), p. 473.

[DH76] Whitfield Diffie and Martin Hellman. “New directions in cryptography”. In: *IEEE transactions on Information Theory* 22.6 (1976), pp. 644–654.

[Fer99] Niels Ferguson. “Impossible differentials in Twofish”. In: *Counterpane Systems. October* 19 (1999).

[Hom13] DI Management Home. *Feistel Network*. 2013. URL: `http://www.di-mgt.com.au/images/feistel.png` (visited on 07/10/2016).

[Hun+07] John D Hunter et al. “Matplotlib: A 2D graphics environment”. In: *Computing in science and engineering* 9.3 (2007), pp. 90–95.

[Int15] Intel. *Rise of the Embedded Internet*. 2015.

[Kah74] David Kahn. *The codebreakers*. Weidenfeld and Nicolson, 1974.

[Kea99] Geoffrey Keating. “Performance analysis of AES candidates on the 6805 CPU core”. In: *Proceedings of The Second AES Candidate Conference*. Addison-Wesley, 1999, pp. 109–114.

[Kel00] John Kelsey. “Key separation in Twofish”. In: *Counterpane Systems. April* 7 (2000).

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. “Differential power analysis”. In: *Advances in Cryptology—CRYPTO’99*. Springer. 1999, pp. 388–397.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[Koc96] Paul C Kocher. “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems”. In: *Advances in Cryptology—CRYPTO’96*. Springer. 1996, pp. 104–113.

[LR88] Michael Luby and Charles Rackoff. “How to construct pseudorandom permutations from pseudorandom functions”. In: *SIAM Journal on Computing* 17.2 (1988), pp. 373–386.

[Luc01] Stefan Lucks. “The saturation attack—a bait for Twofish”. In: *International Workshop on Fast Software Encryption*. Springer. 2001, pp. 1–15.

[Mas94] James L. Massey. “SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm”. In: *Fast Software Encryption, Cambridge Security Workshop Proceedings*. Springer-Verlag, 1994, pp. 1–17.

[MDS02] Thomas S. Messerges, Ezzat A. Dabbish, and Robert H. Sloan. “Examining Smart-Card Security Under the Threat of Power Analysis Attacks”. In: *IEEE Trans. Comput.* 51.5 (May 2002), pp. 541–552. URL: `http://dx.doi.org/10.1109/TC.2002.1004593`.

[Mes01] Thomas S. Messerges. “Securing the AES Finalists Against Power Analysis Attacks”. In: *Proceedings of the 7th International Workshop on Fast Software Encryption*. FSE ’00. London, UK, UK: Springer-Verlag, 2001, pp. 150–164. URL: `http://dl.acm.org/citation.cfm?id=647935.740925`.

[MM99] Fauzan Mirza and Sean Murphy. “An observation on the Key Schedule of Twofish”. In: *In the second AES candidate conference, printed by the national institute of standards and technology*. 1999, pp. 22–23.

[MS00] Rita Mayer-Sommer. “Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards”. In: *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*. CHES ’00. London, UK, UK: Springer-Verlag, 2000, pp. 78–92. URL: `http://dl.acm.org/citation.cfm?id=648253.752540`.

[MVOV96] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[PMO07] Thomas Popp, Stefan Mangard, and Elisabeth Oswald. “Power analysis attacks and countermeasures”. In: *IEEE Design & test of Computers* 24.6 (2007), pp. 535–543.

[RHW11] S. A. M. Rizvi, Syed Zeeshan Hussain, and Neeta Wadhwa. “Performance Analysis of AES and TwoFish Encryption Schemes”. In: *Proceedings of the 2011 International Conference on Communication Systems and Network Technologies*. CSNT ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 76–79. URL: `http://dx.doi.org/10.1109/CSNT.2011.160`.

[Sch+00] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, Tadayoshi Kohno, and Mike Stay. “The Twofish team’s final comments on AES Selection”. In: *AES round* 2 (2000).

[Sch+98] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. “Twofish: A 128-Bit Block Cipher”. In: *in First Advanced Encryption Standard (AES) Conference*. 1998.

[Sch+99] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, and Chris Hall. “On the Twofish Key Schedule”. In: *Proceedings of the Selected Areas in Cryptography*. SAC ’98. London, UK, UK: Springer-Verlag, 1999, pp. 27–42. URL: `http://dl.acm.org/citation.cfm?id=646554.694579`.

[Sch07] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons, 2007.

[Sha45] Claude E Shannon. “A mathematical theory of cryptography”. In: *Memorandum MM* 45 (1945), pp. 110–02.

[Sha49] Claude E Shannon. “Communication theory of secrecy systems”. In: *Bell system technical journal* 28.4 (1949), pp. 656–715.

[Spa06] Ljiljana Spadavecchia. “A network-based asynchronous architecture for cryptographic devices”. In: (2006).

[STM10] L.M. Surhone, M.T. Timpledon, and S.F. Marseken. *Pseudo-Hadamard Transform: Pseudo-Hadamard Transform, Confusion and Diffusion, Hadamard Transform, Matrix (Mathematics), Invertible Matrix, SAFER, Communication Theory of Secrecy Systems*. Betascript Publishing, 2010. URL: `https://books.google.com/books?id=jrr7QwAACAAJ`.

[Tsc] *TwoFish Source Code*. 1998. URL: `https://www.schneier.com/cryptography/twofish/download.html` (visited on 03/01/2016).

[VBC05] Joel VanLaven, Mark Brehob, and Kevin J. Compton. “A Computationally Feasible SPA Attack on AES via Optimized Search”. In: *Security and Privacy in the Age of Ubiquitous Computing: 20th International Information Security Conference (SEC 2005)*. 2005, pp. 577–588.

[VCV11] Stefan VanDerWalt, S Chris Colbert, and Gael Varoquaux. “The NumPy array: a structure for efficient numerical computation”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.

[Wor15] Gartner Says Personal Worlds. *the Internet of Everything Are Colliding to Create New Market s, November 2013*. 2015.

Appendix A

Experiments

In order to provide a well documented attack, all of the employed simulation results are presented in Tables A.1-A.6. The tables showcase result for combinations of σ, τ, N and the number of readings used.

A.1 Results Single Reading LMS-HMS

σ\τ	0	1	2	3	4	5	6	7	8
0.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.1	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.2	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.3	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.4	95.8	99.9	99.9	100.0	100.0	100.0	100.0	100.0	100.0
0.5	83.3	99.3	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.6	61.2	96.0	99.7	100.0	100.0	100.0	100.0	100.0	100.0
0.7	35.1	90.3	98.6	99.9	100.0	100.0	100.0	100.0	100.0
0.8	12.6	80.0	97.1	99.9	100.0	100.0	99.9	100.0	100.0
0.9	5.1	61.8	92.7	99.2	99.4	99.0	99.7	99.5	99.4
1.0	0.5	42.4	86.0	96.6	98.8	97.8	98.2	97.6	98.0
1.1	0.3	25.5	74.6	91.6	93.9	94.4	93.2	94.1	95.6
1.2	0.1	14.5	60.5	81.7	86.9	85.1	86.8	87.0	86.5
1.3	0.0	5.6	43.2	66.4	72.7	71.9	75.3	73.9	72.8
1.4	0.0	2.2	30.6	47.3	54.8	56.2	55.7	58.1	55.6
1.5	0.0	0.8	14.8	32.6	37.8	34.0	36.1	36.3	38.9
1.6	0.0	0.2	8.0	14.7	23.8	21.0	22.7	21.1	22.8
1.7	0.0	0.1	3.7	7.6	9.6	10.5	10.5	10.2	11.2
1.8	0.0	0.0	1.3	3.3	4.0	4.3	4.8	5.6	4.0
1.9	0.0	0.0	0.2	1.6	1.1	1.8	1.7	1.4	1.5
2.0	0.0	0.0	0.3	1.0	0.6	0.3	0.5	0.6	0.5
t(ms)	7.27	11.74	27.71	55.34	95.12	124.38	142.26	142.76	145.08

Table A.1: Single reading accuracies for several σ and τ (128-bit key)

$\sigma \setminus \tau$	0	1	2	3	4	5	6	7	8
0.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.1	100.0	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.2	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.3	99.6	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.4	95.2	99.5	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.5	78.4	98.6	99.7	99.9	100.0	100.0	100.0	100.0	100.0
0.6	48.3	94.4	99.0	100.0	100.0	100.0	100.0	100.0	100.0
0.7	17.3	84.1	98.9	99.8	100.0	100.0	100.0	100.0	100.0
0.8	3.1	69.6	95.9	99.6	100.0	99.9	99.7	99.7	99.8
0.9	0.2	43.3	89.0	98.3	99.1	98.8	99.5	99.0	99.1
1.0	0.0	25.6	78.8	94.8	95.6	97.5	97.5	97.3	97.7
1.1	0.0	13.0	66.1	88.1	91.7	91.8	92.2	92.3	92.1
1.2	0.0	5.0	44.8	74.4	80.4	80.5	81.7	78.7	80.9
1.3	0.0	1.7	27.5	53.4	60.3	61.8	60.4	59.8	63.2
1.4	0.0	0.4	13.6	33.2	40.1	39.0	40.6	42.8	39.0
1.5	0.0	0.0	4.8	15.0	22.7	20.2	21.1	23.1	21.4
1.6	0.0	0.0	2.3	6.7	8.5	10.0	11.6	10.2	11.4
1.7	0.0	0.0	0.9	2.7	2.4	4.2	3.0	4.2	3.0
1.8	0.0	0.0	0.1	0.3	0.8	0.9	1.5	1.4	0.9
1.9	0.0	0.0	0.0	0.2	0.5	0.3	0.2	0.4	0.2
2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
t(ms)	10.71	17.42	41.28	82.71	142.21	186.34	213.34	214.00	217.43

Table A.2: Single reading accuracies for several σ and τ (192-bit key)

$\sigma \backslash \tau$	0	1	2	3	4	5	6	7	8
0.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.1	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.2	100.0	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.3	99.4	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.4	93.9	99.6	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.5	71.7	98.7	99.6	100.0	100.0	100.0	100.0	100.0	100.0
0.6	34.2	92.6	99.5	100.0	100.0	100.0	100.0	100.0	100.0
0.7	10.0	78.5	98.2	99.6	100.0	99.9	100.0	99.9	100.0
0.8	1.1	59.9	93.7	99.5	99.8	99.7	99.8	99.7	99.9
0.9	0.0	32.1	86.8	97.7	99.1	99.7	98.6	98.9	99.2
1.0	0.0	16.8	73.3	93.3	95.5	96.7	96.2	95.8	96.7
1.1	0.0	7.2	55.1	83.1	87.5	89.4	89.1	86.9	88.5
1.2	0.0	1.2	38.1	70.1	75.7	73.7	75.1	75.3	75.1
1.3	0.0	0.3	18.9	44.4	51.1	54.2	52.3	50.8	53.3
1.4	0.0	0.0	7.2	24.5	30.5	28.8	28.5	29.6	31.3
1.5	0.0	0.0	2.2	11.8	13.1	12.8	14.0	14.0	14.2
1.6	0.0	0.0	0.4	2.8	5.0	5.3	3.9	5.0	3.3
1.7	0.0	0.0	0.0	0.6	0.6	1.2	1.0	1.0	0.8
1.8	0.0	0.0	0.0	0.1	0.2	0.2	0.1	0.2	0.3
1.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0
2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0
t(ms)	14.12	23.27	55.50	111.87	193.08	251.06	288.56	289.72	293.90

Table A.3: Single reading accuracies for several σ and τ (256-bit key)

A.2 Results Multiple Readings LMS-HMS

$\sigma \backslash \tau$	0	1	2	3	4	5	6	7	8
0.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.1	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.2	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.3	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.4	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.5	99.8	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.6	98.6	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.7	94.8	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.8	84.7	99.5	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.9	60.1	99.6	100.0	100.0	100.0	100.0	100.0	100.0	100.0
1.0	28.1	98.3	99.9	100.0	100.0	100.0	100.0	100.0	100.0
1.1	8.0	96.6	99.7	100.0	100.0	100.0	100.0	100.0	99.9
1.2	1.3	88.7	99.4	99.9	99.8	99.9	100.0	99.9	100.0
1.3	0.3	78.3	98.5	99.6	99.7	99.7	100.0	99.9	99.9
1.4	0.0	57.1	96.2	99.3	99.0	99.3	99.4	99.8	99.7
1.5	0.0	36.6	90.8	97.7	97.6	98.3	97.9	97.3	97.6
1.6	0.0	18.6	81.2	93.0	93.7	94.8	95.6	92.3	93.3
1.7	0.0	6.3	63.3	81.1	86.2	84.4	85.0	87.0	84.8
1.8	0.0	2.4	42.8	67.0	70.1	72.1	69.0	69.1	68.6
1.9	0.0	0.5	25.2	42.0	47.6	47.8	46.7	46.6	46.3
2.0	0.0	0.1	9.0	23.1	24.5	26.9	28.2	27.3	27.3
t(ms)	28.08	40.14	76.59	155.57	248.59	337.45	376.31	394.82	408.40

Table A.4: Five reading accuracies for several σ and τ (128-bit key)

$\sigma \backslash \tau$	0	1	2	3	4	5	6	7	8
0.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.1	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.2	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.3	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.4	100.0	100.0	99.9	100.0	100.0	100.0	100.0	100.0	100.0
0.5	99.6	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.6	98.3	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.7	89.6	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.8	64.1	99.4	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.9	21.9	98.9	99.8	100.0	100.0	100.0	100.0	100.0	100.0
1.0	4.0	96.2	99.9	100.0	100.0	99.9	100.0	100.0	100.0
1.1	0.3	88.2	100.0	100.0	100.0	100.0	100.0	100.0	100.0
1.2	0.0	69.5	99.1	99.8	100.0	99.8	99.8	99.8	99.9
1.3	0.0	44.6	97.1	99.8	99.7	99.4	99.6	100.0	99.9
1.4	0.0	18.9	92.6	98.1	99.1	98.4	98.9	99.2	98.5
1.5	0.0	5.3	75.5	93.4	95.3	96.1	96.5	95.5	94.5
1.6	0.0	0.8	55.9	79.8	86.5	84.5	84.7	84.6	83.7
1.7	0.0	0.1	28.7	56.3	61.0	65.1	61.1	66.0	64.4
1.8	0.0	0.0	8.9	27.8	33.5	37.3	34.1	32.3	35.1
1.9	0.0	0.0	2.5	9.9	15.3	12.4	16.4	13.9	14.2
2.0	0.0	0.0	0.4	2.1	2.8	2.8	3.5	3.0	3.1
t(ms)	42.62	62.29	120.56	245.10	390.45	532.58	591.92	619.73	641.44

Table A.5: Five reading accuracies for several σ and τ (192-bit key)

$\sigma \backslash \tau$	0	1	2	3	4	5	6	7	8
0.0	99.8	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.1	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.2	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.3	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.4	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.5	99.4	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.6	95.6	99.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.7	84.0	99.8	100.0	100.0	100.0	100.0	100.0	100.0	100.0
0.8	39.8	99.7	99.9	100.0	100.0	100.0	100.0	100.0	100.0
0.9	6.1	98.2	100.0	100.0	100.0	100.0	100.0	100.0	100.0
1.0	0.1	91.7	99.9	100.0	100.0	100.0	100.0	100.0	100.0
1.1	0.0	78.2	99.4	100.0	99.9	99.9	100.0	100.0	100.0
1.2	0.0	45.4	97.5	100.0	100.0	99.9	99.8	99.7	99.9
1.3	0.0	18.1	94.2	99.1	99.2	99.8	99.3	99.4	99.6
1.4	0.0	2.3	77.9	97.1	97.0	97.4	97.5	97.7	98.4
1.5	0.0	0.3	55.8	85.4	89.3	90.5	90.4	89.9	90.4
1.6	0.0	0.0	23.7	63.8	68.2	68.2	69.3	68.6	70.2
1.7	0.0	0.0	8.2	31.8	36.9	37.0	38.8	36.7	37.0
1.8	0.0	0.0	1.3	7.4	12.5	10.1	9.6	14.0	10.7
1.9	0.0	0.0	0.1	1.3	2.0	1.7	2.7	2.0	2.4
2.0	0.0	0.0	0.0	0.2	0.3	0.1	0.0	0.1	0.4
t(ms)	57.23	84.76	165.31	336.30	537.82	730.35	811.54	848.98	879.90

Table A.6: Five reading accuracies for several σ and τ (256-bit key)

Appendix B

Implementation

THE Python 2.7.11 implementation used for the result generation is included in the following sections with corresponding descriptions of the algorithms and scripts used.

B.1 Compute Power Traces

The following auxiliary library was used to generate the simulated power traces. The code is based on the description provided by original paper [Sch+98] and the implementation details provided in the public source code [Tsc].

Listing B.1: Library for computing Power Traces

```python
# -*- coding: utf8 -*-

# File name: twofish.py
# Author: Jose Javier Gonzalez
# Date created: 2015-11-11
# Date last modified: 2016-07-11
# Python Version: 2.7.11

hamming = [bin(i).count('1') for i in range(256)]

def list2bin(L):
    """
    Simple function that converts a simple 0-1 bitstring to its integer representation
    """
    n = 0x00
    for i in range(len(L)):
        n |= int(L[i])
        n <<= 1
    return n>>1

def bin2list(n):
    """
    Simple function that converts a simple 0-1 bitstring to its integer representation
    """
    L = [0]*8
    for i in range(8):
        L[7-i] = n % 2
        n >>= 1
    return L

def _q(x,T):
    """
    Generic definition for q0 and q1 specified in the TwoFish paper
    """
    def ROR4(x,n):
        return ( (x >> n) | (x << (4-n)) ) & 0x0f
    t0, t1, t2, t3 = T
    a0,b0 = x >> 4 , x & 0x0f
    a1 = a0 ^ b0
    b1 = a0 ^ ROR4(b0,1) ^ ((a0 << 3) & 0xf)
    a2, b2 = t0[a1], t1[b1]
    a3 = a2 ^ b2
    b3 = a2 ^ ROR4(b2,1) ^ ((a2 << 3) & 0xf)
    a4, b4 = t2[a3], t3[b3]
    y = (b4 << 4) | a4
    return y

def q0(x):
    t0 = [ 0x8, 0x1, 0x7, 0xD, 0x6, 0xF, 0x3, 0x2, 0x0, 0xB, 0x5, 0x9, 0xE, 0xC, 0xA, 0x4, ]
    t1 = [ 0xE, 0xC, 0xB, 0x8, 0x1, 0x2, 0x3, 0x5, 0xF, 0x4, 0xA, 0x6, 0x7, 0x0, 0x9, 0xD, ]
    t2 = [ 0xB, 0xA, 0x5, 0xE, 0x6, 0xD, 0x9, 0x0, 0xC, 0x8, 0xF, 0x3, 0x2, 0x4, 0x7, 0x1, ]
    t3 = [ 0xD, 0x7, 0xF, 0x4, 0x1, 0x2, 0x6, 0xE, 0x9, 0xB, 0x3, 0x0, 0x8, 0x5, 0xC, 0xA, ]
    T  = [ t0, t1, t2, t3 ]
    return _q(x,T)

def q1(x):
    t0 = [ 0x2, 0x8, 0xB, 0xD, 0xF, 0x7, 0x6, 0xE, 0x3, 0x1, 0x9, 0x4, 0x0, 0xA, 0xC, 0x5, ]
    t1 = [ 0x1, 0xE, 0x2, 0xB, 0x4, 0xC, 0x3, 0x7, 0x6, 0xD, 0xA, 0x5, 0xF, 0x9, 0x0, 0x8, ]
    t2 = [ 0x4, 0xC, 0x7, 0x5, 0x1, 0x6, 0x9, 0xA, 0x0, 0xE, 0xD, 0x8, 0x2, 0xB, 0x3, 0xF, ]
    t3 = [ 0xB, 0x9, 0x5, 0x1, 0xC, 0x3, 0xD, 0xE, 0x6, 0x4, 0x7, 0xF, 0x2, 0x0, 0x8, 0xA, ]
    T  = [ t0, t1, t2, t3 ]
    return _q(x,T)

Q0 = [q0(i) for i in range(256)]
Q1 = [q1(i) for i in range(256)]

# Matrix with the substitutions (k,j) that have to be carried out in the
   k-th step (k decreases) for the j-th block
Q = [
[Q0, Q1, Q0, Q1], # k = 0
[Q1, Q1, Q0, Q0], # k = 1
[Q1, Q0, Q1, Q0], # k = 2
[Q0, Q0, Q1, Q1], # k = 3
[Q1, Q0, Q0, Q1], # k = 4
]

def getHamming(K,n):
    """
    Function that generates the Hamming weights after the operation of V
   xor m_x
    V - represents the output of each step of substitution and xor in
   the TwoFish key Schedule
    m_x - is the x-th byte of the key K

    Returns HX : The hamming values of item
            HX[i][x] is the hamming value after xoring m_x with the
   accumulated value V
    """
    HX = [[0 for x in range(n/8)] for i in range(40)]
    HY = [[0 for x in range(n/8)] for i in range(40)]
    # HX = np.zeros((40,n/8),dtype=int)
    # HY = np.zeros((40,n/8),dtype=int)
    for i in range(40):
        V = [i, i, i ,i]
        for k in reversed(range(1,n/64+1)):
            V = [q[v] for v,q in zip(V,Q[k]) ]
            l = 2*(k-1) + i%2
            for j in range(4):
                m = 4*l+j
                V[j] = V[j]^K[m]
                HX[i][m] = hamming[ V[j] ]
                HY[i][m] = hamming[ Q[k-1][j][V[j]] ]

    return HX,HY
```

B.2 Single Reading without Noise

Here is presented the brute force approach described in Section 5.2 that is able to unequivocally recover the key as long as the Hamming weights are produced from an execution of the TwoFish Key Schedule. This algorithm requires the power trace to not contain any errors or added noise. For results of this approach please refer back to Section 6.1.

Listing B.2: Script for Single Reading without Noise

```python
#!/usr/bin/env python
# -*- coding: utf8 -*-

# File name: break_twofish.py
# Author: Jose Javier Gonzalez
# Date created: 2015-11-11
# Date last modified: 2016-07-11
# Python Version: 2.7.11

# python break_twofish_error [<iterations> = 1000]

from twofish import *
import time
from random import randint
import sys

def breakKey(HX,n,K):
    """
    Function that returns the key used in a encryption of the TwoFish
   cipher given the Hamming Values HX
    Uses a system of
    Returns:
        NK - the estimated key
    """
    NK = [None]*(n/8)             # The value that is going to be
   estimated from the hamming values
    V = [0 for i in range(20)]    # Accumulated value through the h
   function

    for parity in [0,1]:                             # Even and odd cases
   use different parts of the key since keys are generated in pairs (PHT)
        for j in range(4):                           # The function h
   works in 32 bit works, 4 bytes at a time
            V = [2*i+parity for i in range(20)]
            for k in reversed(range(1,n/64+1)):      # The number of
   steps of the function h will depend on the length of the key
                q = Q[k][j]                          # Each byte will use
   a different substitution box depending on the step and the position
   in the word, same for all iterations
```

```
                m = 8*(k-1)+j+4*parity              # index of the word with position j at step k

                V = [q[v] for v in V]               # We apply the corresponding S-box

                for x in range(2**8):               # We make a brute force search to find the subkey m
                    for i in range(20):
                        if bin(V[i]^x).count('1') != HX[2*i+parity][m]:
                            break # If the candidate value does not produce a correct output hamming, is not valid
                    else:                           # If satisfies all, we are done searching
                        break
                NK[m] = x                           # Save the key into the array
                V = [v^x for v in V]                # Updates values in parallel


    return NK

def testBreakKey(N=128,iterations=100):
    """
    Simple function to test the breakKey function with a random key
    Receives:
        N - Size of the key being used
        iterations - Number of iterations to test

    Returns:
        The percentage of successful key guesses
    """
    successful = 0
    for t in range(iterations):
        K = [randint(0,255) for x in range(N/8)]
        HX,HY = getHamming(K,N)
        K2 = breakKey(HX,N,K)
        if K == K2:
            successful += 1
        # else:
        #     print K
        #     print K2
        #     print np.array(K)^np.array(K2)

    return float(successful)/iterations

if __name__ == '__main__':
    it = 100

    if len(sys.argv) > 1:
        it = int(sys.argv[1])

    print "IT = %d" % it
    print "====="+"="*len(str(it))
    for n in [64,128,192,256]:
        start = time.clock()
        p = testBreakKey(n,iterations=it)
        end = time.clock()
        print n,p, (end-start)/it*1000
    print "All tests were successful"
```

B.3 Single Reading with Noise

The excerpt presented here is built on top of the script presented in the previous section and implements the LMS and HMS techniques described in Sections 5.3.1 and 5.3.2 in order to make the algorithm resistant to high levels of noise. When run individually the scripts simulates for the given number of iterations a number of experiments. The experiments will have varying amounts of noise (specified by the standard deviation σ) and for different factors of correction (specified by the maximum Hamming weight of the masks τ). For results of this approach please refer back to Section 6.2

Listing B.3: Script for Single Reading with Noise

```
#!/usr/bin/env python
# -*- coding: utf8 -*-

# File name: break_twofish_error.py
# Author: Jose Javier Gonzalez
# Date created: 2015-11-11
# Date last modified: 2016-07-11
# Python Version: 2.7.11

# python break_twofish_error [<iterations> = 100 [mask_size]]

from twofish import *
import time
from random import randint
import numpy as np
import sys
import json
from collections import OrderedDict

def breakKey_error(HX,HY,n,K,mask_size=0):
    """
    Function that returns the key used in a encryption of the TwoFish cipher given the Hamming Values HX
    Uses a system of
    Returns:
        NK - the estimated key
    """

    M = [1, 9, 37, 93, 163, 219, 247, 255, 256] #Partial sums of Pascal triangle in order to determine the amount of maximum amount of bits to toggle
    perm_mask = sorted(range(2**8),key=lambda x: hamming[x])[:M[mask_size]]# #Permutation Masks ordered by hamming weight

    def _create(n,size=8):
        """
        Converts a number to its bitstring but instead of using 0s and 1s
        it uses 1s and -1s to accommodate the fact that we are solving a system
        of equations of the form x xor b
        """
        L = [0]*8
        for i in reversed(range(size)):
            L[i] = -2*(n%2)+1
            n >>= 1
        return L

    HX = HX.round().astype(int)
    HY = HY.round().astype(int)


    NK = [None]*(n/8)               # The value that is going to be estimated from the hamming values
    V = [0 for i in range(20)]      # Accumulated value through the h function
    A = np.zeros((20,8),dtype=int) # Matrix with the bit values (converted to 1 and -1) for a specific v in W to solve the hamming system
    B = np.zeros((20,1),dtype=int) # Matrix with the hamming values used as the other part of the system


    for parity in [0,1]:                            # Even and odd cases use different parts of the key since keys are generated in pairs (PHT)
        for j in range(4):                          # The function h works in 32 bit works, 4 bytes at a time
            V = [2*i+parity for i in range(20)]
            for k in reversed(range(1,n/64+1)):     # The number of steps of the function h will depend on the length of the key
                q = Q[k][j]                     # Each byte will use a different substitution box depending on the step and the position in the word, same for all iterations
                m = 8*(k-1)+j+4*parity     # index of the word with position j at step k
                for i in range(20):             # We will need to look at 20 iterations that use the same bytes of the key at each step
                    ii = 2*i+parity             # Id of iterations for each parity
                    V[i] = q[V[i]]                  # We apply the corresponding S-box
                    B[i] = HX[ii][m]-hamming[V[i]]                  # We create the independant vector from the hamming of the output minus the hamming of the input
                    A[i,:] = np.array(_create(V[i]))                  # We create the 1,-1 row to solve the system
```

```python

                X = (np.linalg.lstsq(A,B)[0]).round().astype(int)
                X = np.clip(X,0,1) # To remove ...,-2,-1 or 2,3...
                x = list2bin(X)

                z = x #Best estimator

                min_errors = 8*40
                for pm in perm_mask: #Will only enter if mask_size > 0
                    y = x^pm            #candidate key
                    errors = 0
                    for i in range(20):
                        ii = 2*i+parity
                        errors += abs( hamming[ V[i]^y ] - HX[ii][m] )
                        errors += abs( hamming[ Q[k-1][j][ V[i]^y ] ] - HY[ii][m] )
                    if errors < min_errors:
                        min_errors = errors
                        z = y

                NK[m] = z
                V = [v^NK[m] for v in V]                # Updates values in parallel

    return NK

def testBreakKey_error(N=128,iterations=100,std=0.0,mask_size=0):
    """
    Simple function to test the breakKey function with a random key
    Receives:
        N - Size of the key being used
        iterations - Number of iterations to test
        std - Standard deviation of the gaussian noise added to the hamming values

    Returns:
        The percentage of successful key guesses
    """
    successful = 0
    scores = []
    for t in range(iterations):
        K = [randint(0,255) for x in range(N/8)]
        HX,HY = getHamming(K,N)
        HX = np.array(HX,dtype=float)
        HY = np.array(HY,dtype=float)
        if std != 0.0:
            a,b = len(HX),len(HX[0])
            noise = np.random.normal(0,std,a*b).reshape(a,b) #add gaussian noise
            HX += noise
            noise = np.random.normal(0,std,a*b).reshape(a,b) #add gaussian noise
            HY += noise
        K_est = breakKey_error(HX,HY,N,K,mask_size)
        if K == K_est:
            successful += 1
        scores.append(np.mean([int(k == k_est) for (k,k_est) in zip(K,K_est)]) )

        # else:
            # print K
            # print K_est
            # print np.array(K)^np.array(K_est)
    return float(successful)/iterations, np.mean(scores)

if __name__ == '__main__':
    accuracy, scores, times = OrderedDict(), OrderedDict(), OrderedDict()
    it = 1000
    mask_size = 0

    if len(sys.argv) > 1:
        it = int(sys.argv[1])
        if len(sys.argv) > 2:
            mask_size = int(sys.argv[2])

    print "IT = %d" % it
    print "M = %d" % mask_size
    print "============"
    # stds = np.arange(0,1.5,0.05)
    stds = [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2.0]
    for s in stds:
        print s
        for n in [128, 192, 256]:
            start = time.clock()
            p,sc = testBreakKey_error(n,it,s,mask_size)
            end = time.clock()
            print n, p, round(sc,2), (end-start)/it*1000
            accuracy[str((n,s,mask_size))] = p
            scores[str((n,s,mask_size))] = sc
            times[str((n,s,mask_size))] = (end-start)/it
        print

    with open('results_%d.json' % mask_size,'w') as fp:
        json.dump([accuracy,scores,times],fp,indent=4)
```

B.4 Multiple Reading with Noise

The last scripts extends the functionality of Listing B.3 in order to accommodate for multiple readings of the same key as it was discussed in Section 5.4. A maximum number of reading has to be specified, nevertheless, for speedup purposes the algorithm will not request added readings if with the given ones has enough information to take a majority vote on all the bytes of the secret key. If it reaches the `MAX_READINGS` it will continue and use all the available information. For results of this approach please refer back to Section 6.3

Listing B.4: Script for Multiple Reading with Noise

```python
#!/usr/bin/env python
# -*- coding: utf8 -*-

# File name: break_twofish_error_avg.py
# Author: Jose Javier Gonzalez
# Date created: 2015-11-11
# Date last modified: 2016-07-11
# Python Version: 2.7.11

# python break_twofish_error_avg [<iterations> = 100 [mask_size]]

from twofish import *
import time
from random import randint
import numpy as np
import sys
import json
from collections import OrderedDict
from break_twofish_error import breakKey_error

MAX_READINGS = 5

def testBreakKey_error_avg(N=128,iterations=100,std=0.0,mask_size=0):
    """
    Simple function to test the breakKey function with a random key
    Receives:
        N - Size of the key being used
        iterations - Number of iterations to test
        std - Standard deviation of the gaussian noise added to the
    hamming values

    Returns:
        The percentage of successful key guesses
    """
    threshold = 2
    def _add_noise(H):
        a,b = len(H),len(H[0])
        noise = np.random.normal(0,std,a*b).reshape(a,b) #add gaussian noise
        return H + noise

    successful = 0
    scores = []
    for t in range(iterations):
        K = [randint(0,255) for x in range(N/8)]
        HX,HY = getHamming(K,N)
        HX = np.array(HX,dtype=float)
        HY = np.array(HY,dtype=float)

        majority = False
        K_ests = []
        it = 0
        while not majority:
            it += 1
            if std != 0.0:
                HXn, HYn = _add_noise(HX), _add_noise(HY)
            else:
                HXn, HYn = HX, HY
            Kn = breakKey_error(HXn,HYn,N,K,mask_size)
            K_ests.append(Kn)
            majority = all([ max(np.bincount(ks)) >= threshold for ks in zip(*K_ests)])
            if it >= MAX_READINGS:
                break

        K_est = [max(set(ks), key=ks.count) for ks in zip(*K_ests)]
        if K == K_est:
            successful += 1
        scores.append(np.mean([int(k == k_est) for (k,k_est) in zip(K,K_est)]) )

        # else:
            # print K
            # print K_est
            # print np.array(K)^np.array(K_est)
    return float(successful)/iterations, np.mean(scores)

if __name__ == '__main__':
    accuracy, scores, times = OrderedDict(), OrderedDict(), OrderedDict()
    it = 1000
    mask_size = 0

    if len(sys.argv) > 1:
        it = int(sys.argv[1])
        if len(sys.argv) > 2:
            mask_size = int(sys.argv[2])

    print "IT = %d" % it
    print "M = %d" % mask_size
    print "============"
    # stds = np.arange(0,1.5,0.05)
    stds =
   [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2
    for s in stds:
        print s
        for n in [128, 192, 256]:
            start = time.clock()
            p,sc = testBreakKey_error_avg(n,it,s,mask_size)
            end = time.clock()
            print n, p, round(sc,2), (end-start)/it*1000
            accuracy[str((n,s,mask_size))] = p
            scores[str((n,s,mask_size))] = sc
            times[str((n,s,mask_size))] = (end-start)/it
        print

    with open('results_avg_%d.json' % mask_size,'w') as fp:
        json.dump([accuracy,scores,times],fp,indent=4)
```