Profiling Side-Channel Attacks on Cryptographic Algorithms

Neil Hanley

June 2014

A Thesis Submitted to the
National University of Ireland, Cork
in Fulfillment of the Requirements for
the Degree of
Doctor of Philosophy

Supervisor: Dr. William P. Marnane
Head of Department: Prof. Nabeel A. Riza

Department of Electrical and Electronic Engineering,
National University of Ireland, Cork.

Abstract

Traditionally, attacks on cryptographic algorithms looked for mathematical weaknesses in the underlying structure of a cipher. Side-channel attacks, however, look to extract secret key information based on the leakage from the device on which the cipher is implemented, be it smart-card, microprocessor, dedicated hardware or personal computer. Attacks based on the power consumption, electromagnetic emanations and execution time have all been practically demonstrated on a range of devices to reveal partial secret-key information from which the full key can be reconstructed.

The focus of this thesis is power analysis, more specifically a class of attacks known as profiling attacks. These attacks assume a potential attacker has access to, or can control, an identical device to that which is under attack, which allows him to profile the power consumption of operations or data flow during encryption. This assumes a stronger adversary than traditional non-profiling attacks such as differential or correlation power analysis, however the ability to model a device allows templates to be used post-profiling to extract key information from many different target devices using the power consumption of very few encryptions. This allows an adversary to overcome protocols intended to prevent secret key recovery by restricting the number of available traces.

In this thesis a detailed investigation of template attacks is conducted, along with how the selection of various attack parameters practically affect the efficiency of the secret key recovery, as well as examining the underlying assumption of profiling attacks in that the power consumption of one device can be used to extract secret keys from another. Trace only attacks, where the corresponding plaintext or ciphertext data is unavailable, are then investigated against both symmetric and asymmetric algorithms with the goal of key recovery from a single trace. This allows an adversary to bypass many of the currently proposed countermeasures, particularly in the asymmetric domain.

An investigation into machine-learning methods for side-channel analysis as an alternative to template or stochastic methods is also conducted, with support vector machines, logistic regression and neural networks investigated from a side-channel viewpoint. Both binary and multi-class classification attack scenarios are examined in order to explore the relative strengths of each algorithm. Finally these machine-learning based alternatives are empirically compared with template attacks, with their respective merits examined with regards to attack efficiency.

Acknowledgements

Like any Ph.D. submission, this thesis would not have been possible without the help and support of a number of people. First and foremost I would like to thank Liam Marnane, initially for allowing me the opportunity to undertake a Ph.D. under his supervision, and subsequently for his support and encouragement when it ran considerably over time. His door was always open whenever I had a question, and I am particularly grateful for his willingness to meet weekends after I had moved away from Cork.

I am also indebted to Mike Tunstall for sharing (and continuing to share) his knowledge on all things crypto. His constant help and willingness to collaborate on work even after he left UCC was much appreciated. Likewise, I would also like to thank members of the coding & crypto group - Andy, Brian, Rob, Maurice, Mark, Weibo, for acting as sounding boards for work and ideas. Particular thanks also to Andrey for his help with the machine learning side of things. I would also like to thank Máire O'Neill of Queen's University Belfast for her patience while I finished my Ph.D. while working there.

I'd like to thank everyone in the Department of Electrical and Electronic Engineering at UCC for helping me during my time there. Thanks to Aidan, Dave, Eoin, James, Kev, Niamh, Orla, and all the other postdocs and postgrads for the craic during lunch and all the card games.

I would also like to thank my parents for their unwavering support and constant reassurance throughout my study. Lastly, but by no means least, I would like to thank my long-suffering girlfriend Dee who is only just getting to know me without 'the guilt'!!!. Thank you for your unbelievable patience (and great cooking) all those evenings & weekends I spent in work while completing my thesis.

Declaration

I hereby state that all of the work undertaken in this thesis is original in content and was carried out by the author. Work carried out by others has been duly acknowledged in the thesis. The work presented has not been accepted in any previous application for a degree.

Signed: _____

Date: _____

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

3-DES	triple-DES
AC	alternating current
ADC	analog-to-digital converter
AES	Advanced Encryption Standard
ASIC	application-specific integrated circuit
CIOS	coarsely integrated operand scanning
CMOS	complementary metal-oxide-semiconductor
CPA	correlation power analysis
dB	decibel
DC	direct current
DES	Data Encryption Standard
DFT	discrete Fourier transform
DH	Diffie-Hellman
DOM	difference of means
DPA	differential power analysis
DSP	digital signal processing
ECC	elliptic curve cryptography

ECDLP . . . . . elliptic curve discrete log problem

ECDSA . . . . . elliptic curve digital signature standard

EM . . . . . electromagnetic

FA . . . . . fault analysis

FFT . . . . . fast Fourier transform

FIR . . . . . finite impulse response

FPGA . . . . . field programmable gate array

GCHQ . . . . . Government Communications Headquarters

GMM . . . . . Gaussian mixture models

HMAC . . . . . keyed-hash message authentication code

HMM . . . . . hidden Markov model

IC . . . . . integrated circuit

IDEA . . . . . International Data Encryption Algorithm

IPA . . . . . inferential power analysis

ISO . . . . . International Organization for Standardisation

LDA . . . . . linear discriminant analysis

LR . . . . . logistic regression

LS-SVM . . . . . least squares support vector machines

LSB . . . . . least significant bit

LUT . . . . . look-up table

MAF . . . . . moving average filter

MIA . . . . . mutual information analysis

MOSFET . . . . . metal oxide semiconductor field effect transistor

NIST . . . . . National Institute of Standards and Technology

NN . . . . . neural network

NSA . . . . . National Security Agency

PC . . . . . personal computer

PCA . . . . . principal component analysis

PEA . . . . . photonic emission analysis

PI . . . . . perceived information

PUF . . . . . physically unclonable function

QDA . . . . . quadratic discriminant analysis

QP . . . . . quadratic programming

RAM . . . . . random access memory

RBF . . . . . radial basis function

RFID . . . . . radio-frequency identification

RNG . . . . . random number generator

RSA . . . . . Rivest-Shamir-Adleman

SCA . . . . . side-channel attack

SM . . . . . stochastic method

SNR . . . . . signal-to-noise ratio

SPA . . . . . simple power analysis

SPN . . . . . substitution-permutation network

SSL . . . . . secure socket layer

SVM . . . . . support vector machine

TA . . . . . template attack

TLS . . . . . transport layer security

USB . . . . . universal serial bus

Nomenclature

$\hat{s}$	Projected secret value.
λ	Neural network and logistic regression regularisation parameter.
$\mathbb{F}_p$	Finite field of characteristic p, and with p elements.
$\mathcal{C}$	Set of all possible ciphertexts.
$\mathcal{E}$	Elliptic curve.
$\mathcal{F}$	Function to map a class label y to a secret s.
$\mathcal{H}$	Support vector machine hyperplane.
$\mathcal{K}$	Set of all possible classes.
$\mathcal{L}$	Lagrange multiplier.
$\mathcal{O}$	Elliptic curve point at infinity.
$\mathcal{P}$	Set of all possible plaintexts.
$\mathcal{S}$	Set of all possible secrets.
μ	Mean vector.
Σ	Covariance matrix.
τ	Mask value.
$\Theta^{(i)}$	Neural network weights for layer i.
$\tilde{n}$	Number of features retained in a trace.
$\tilde{x}$	Transformed matrix of traces of size $m \times \tilde{n}$, each trace a row vector.

C	Support vector machine cost function.
$c^{(i)}$	Ciphertext output for a trace $x^{(i)}$ with a plaintext $p^{(i)}$.
D	Eigenvalues of a set of traces.
F_c	Filter cut-off frequency.
F_s	Sampling frequency.
HW	Hamming weight function.
I	Identity matrix.
K	Kernel function.
L	Leakage function.
m	Number of training samples in x.
$m^{(i)}$	Number of training samples in x belonging to the class $o^{(i)}$.
N	Filter window length.
n	Number of features in a trace.
$o^{(i)}$	Unique instance of the class $\mathcal{K}$.
P	Trace projection matrix.
$p^{(i)}$	Plaintext input for a trace $x^{(i)}$.
S	Substitution function.
s	Secret to be recovered.
s^*	Guess of secret value.
U	Eigenvectors of a set of traces.
x	Matrix of traces of size $m \times n$, each trace a row vector.
$x^{(i)}$	Trace vector.
$x_j^{(i)}$	Specific time point of a trace.
$x^{(i,j)}$	Trace vector from the subset of x with labels from the class $o^{(j)}$.
$y^{(i)}$	Class label corresponding to the trace $x^{(i)}$ where $y^{(i)} \in \mathcal{K}$.
$y_j^{(i)}$	Bit j of the binary representation of class label.

Chapter 1

Introduction

Cryptography is the science of keeping communications secure such that only the intended recipients can read a message. The value of this was clear to the ancient Romans, who encrypted their military communications using the *Caesar Cipher*, one of the first known algorithms used to encrypt information. For as long as encryption algorithms have been used, people have also looked to break them, and advances in cryptographic algorithm design often go step in step with advances in cryptanalysis. Historically, the use of cryptography was limited to the military and governmental organisations, with research in the area often classified for reasons of national security and subject to export controls due to cryptography being classed as military technology. This can be seen in the fascinating work done at Bletchley Park in breaking the German Enigma encoding machines during World War II which was only released to the public many years later.

While Claude Shannon's pioneering work on information theory (and subsequently on the *One-Time Pad*) was published in the late forties [197], this was declassified wartime work, and it was not until the mid to late seventies that research into cryptography really began to become mainstream in universities and research labs. It was then that IBM developed the Data Encryption Standard (DES) encryption standard in conjunction with the National Security Agency (NSA) [162], and the seminal work on asymmetric cryptography such as Diffie-Hellman (DH) [60] and Rivest-Shamir-Adleman (RSA) [192] was published. Indeed the secretive nature of cryptographic research was such that the mathematical basis for both DH and RSA had previously been discovered by an English mathematician Clifford Cocks working for Government Communications Headquarters (GCHQ), the UK intelligence agency, however the work was only declassified in 1998.

Authentication	Proves that a user is who they claim to be
Non-Repudiation	Prevents a user subsequently denying that they sent the data. This is particularly important for online commerce
Confidentiality	Provides secretary between parties such that only users with knowledge of the key can read the data.
Integrity	Ensures that data has not been tampered with in transit.

Table 1.1: Cryptographic functions.

While the banking and financial sectors have made widespread use of cryptography, the advent of the internet and the always connected world of smart-phones, has ensured that it has turned from an esoteric technology originally only used by the military to something that is used by the majority of the internet connected people around the world on a daily basis, often without even realising it as they log into their email accounts, Facebook, *etc.* Indeed the recent revelations about widespread monitoring of the internet by the NSA and GCHQ by the ongoing *Edward Snowden* document cache only highlights the importance of securing online communications through the widespread adoption of cryptographic algorithms and protocols.

End to end encryption between two entities requires the use of a cryptographic protocol such as transport layer security (TLS)/secure socket layer (SSL) for example, which is a widely used protocol for internet communications. The goal of a protocol is to provide the cryptographic functions as defined in Table 1.1 as required. These protocols are in turn constructed from cryptographic primitives which can be seen as the building blocks of cryptographic systems. These building blocks can further be classified into symmetric and asymmetric algorithms. Symmetric algorithms utilise the same key for both encryption and decryption operations. This allows designers to construct extremely fast and secure algorithms that can encrypt large amounts of information quite efficiently. Block ciphers such as DES and the Advanced Encryption Standard (AES) [163], and stream ciphers such as RC4 are examples of symmetric-key algorithms that form the basis of many protocols.

The disadvantage of symmetric-key algorithms is the requirement for both parties to share the same key. The key-sharing problem amplifies with scale as every pair of people who want to communicate require unique keys. The previously mentioned DH key exchange

overcomes this limitation by allowing two parties who have never previously met to establish a shared key for secure communications. Further research into asymmetric encryption led to the development of RSA which has a public encryption key that anybody can use to encrypt a message, however only the owner of the corresponding private decryption key can recover the message. More recently, elliptic curve cryptography (ECC) [123, 152] has been widely adopted for asymmetric purposes due to its shorter key lengths, particularly in the embedded domain.

Additional primitives are required to achieve the cryptographic functions in Table 1.1. Hash functions are one-way functions that provide an irreversible *fingerprint* of a block of data which is useful for providing integrity, and secure random number generators (RNGs) are essential to any protocol. Indeed the recent controversy over the dual elliptic curve deterministic random bit generator [18], shows that tampering with the RNG can be a somewhat efficient and subtle way to undermine a cryptographic system.

More recently, public cryptographic competitions have been utilised to select new algorithms which provides an insurance that no back doors are present in a new standard. This was the case with both AES and the recent SHA-3 hash function competition run by National Institute of Standards and Technology (NIST), as well as eSTREAM, the European Union funded ECRYPT stream cipher project. As well as algorithm design, these competitions have lead to great advances in cryptanalysis also, particularly when combined with parallel computing using graphics cards, or dedicated hardware such as application-specific integrated circuits (ASICs) or reconfigurable devices such as field programmable gate arrays (FPGAs). Indeed a brute-force search of DES keys now takes less than a week with a reconfigurable hardware cracker such as COPACOBANA [128].

Traditionally, attacks on cryptographic primitives have focused on analysing inputs and outputs of systems, however in the late nineties the first scientific publications of timing [125] and power [126] attacks showed that the implementation of an algorithm must also be taken into account, especially in the context of embedded security where an attacker might have direct access to a device. This was followed up with further research showing that electromagnetic emanations could also leak key information [77, 184]. Since then a wide body of research has been produced[1] by a large number of universities and research labs around the world, and a large number of conferences such as CHES [26], CT-RSA [22], SAC [122], and COSADE [183], to name but a few, have dedicated side-channel tracks.

[1] As of the 1^{st} April 2014, the original timing and power analysis papers by Kocher *et al.* have 2583 and 4035 citations respectively according to Google scholar

	Active	**Passive**
Non-Invasive	Glitching, Temperature, Voltage, ...	Timing, Power, EM, ...
Semi-Invasive	Light, Radiation, ...	EM (decapsulation), Optical Inspection, ...
Invasive	Short Circuiting, permanent circuit change, ...	Probing, ...

Table 1.2: Attack taxonomy.

While academic interest in side-channel attacks against cryptographic algorithms grew after the publications by Kocher *et al.*, some vulnerabilities were known much earlier against non-cryptographic targets. For example in 1985, van Eck looked at using electromagnetic radiation from the cathode ray video display units that were in use at the time to reconstruct what was on the screen [215]. Recently declassified information on the TEMPEST program conducted by the NSA shows that they were aware of the importance of blocking (and potential for exploiting) electromagnetic emanations [57]. Indeed acoustic side-channel attacks have more recently published against keyboard strokes [235] and dot matrix printers as used in banks [14].

Attacks themselves can be classified into *active* and *passive* depending on if they affect the cryptographic output of the computation being performed. For example voltage glitching can be used to introduce faults into a computation and subsequently recover the key [32]. Passive attacks in contrast do not affect the cryptographic result. Attacks can also be grouped according to their effect on the target device, with attacks that don't leave permanent damage to a device classified as non-invasive, with attacks that permanently damage or destroy it as invasive. A taxonomy of attacks is given in Table 1.2 [142].

The work of this thesis examines *Power Analysis Attacks*, as originally introduced by Kocher *et al.* [126], which utilises the power consumption of a secure device while performing an cryptographic operation to recover some secret about the device. Power attacks can be further broken into non-profiling or profiling attacks.

Non-profiled attacks include simple power analysis (SPA) which assumes that an attacker has access to a single or very few power traces which can be visually inspected to extract key information through some timing or distinct power consumption pattern. This class of attacks includes the classical differential power analysis (DPA) [126] and the widely utilised correlation power analysis (CPA) [34], as well as related attacks which compare many

recorded power traces with some hypothetical power model for a given key, determining the correct key through some distinguisher. Collision attacks [196] fall somewhere in-between SPA and DPA, and look to identify parts of a power trace where the same variable or operation is processed by comparing the power consumption at that point, with the number of traces required for successful key recovery dependent on the target device and context.

Profiling Attacks are the main focus of this thesis. They assume a stronger attack model with an attacker having access to an identical or similar device to *learn* the power consumption characteristic of a device by building templates [42] or stochastic methods (SMs) [195] for example. This model of the power consumption can then be used to break a different target device quickly and easily. They can be viewed as a statistical learning problem for which there is a wide range of literature available such as [30, 63, 95, 106]. Various adversarial models give attackers different capabilities ranging from having full control over the identical device and knowing all secret information [42], to simply knowing that it has some bias in its RNG [3]. They are an important class of attacks as the availability of a profiling device gives an attacker a wide range of possibilities, as well as being able to mount very powerful attacks. They are also beneficial in the context of penetration testing when evaluating the susceptibility of a device to side-channel attacks (SCAs).

Academic publications have shown the real world implications of side-channel analysis, such as the attacks against the KeeLoq remote entry system [67], attacks against contactless payment cards [169], the bit-stream encryption in Xilinx FPGAs [155], and more recently the YubiKey multi-factor authentication token [171]. In the last two decades research into SCAs has moved from an niche academic research topic to a large multi-million dollar industry with companies such as Cryptography Research (`http://www.cryptography.com/`) and Riscure (`https://www.riscure.com/`) selling testing equipment and countermeasures, the former having been bought out by Rambus for $\$345.5M$.

1.1 Summary of Contributions

The following summarises the contributions discussed in this thesis,

- A thorough evaluation of template attacks (TAs) is performed, and the effect of practical attack considerations is examined. When conducting a profiling TA there are many inter-related parameters to choose, and an empirical study is conducted on various software platforms.

- The basic premise of a TA requires that the profiling device has a similar power consumption such that templates built on one device are applicable to another. This assumption is examined by conducting attacks across twenty different smart-cards to examine its validity.
- Power attacks in general assume some knowledge of plaintext or ciphertext information. An attack is developed that can recover key information based on the power consumption alone.
- An efficient TA on the key schedule across multiple rounds is presented, and it is shown how the original key can subsequently be recovered. Again no knowledge of input or output texts are required.
- It is shown how to build templates to determine if the inputs to a multiplication are equal or not using only power consumption hence distinguishing between multiplication and squaring operations. An attack on RSA is outlined and the effect of the key bit-size examined.
- A TA on elliptic curve digital signature standard (ECDSA) is performed based on distinguishing multiplication and squaring operations. The effect of various countermeasures on the attacks is examined, and practical considerations examined.
- A comparison between different machine learning methods in the context of SCA is performed to investigate what algorithm is most advantageous to use for different contexts.
- Logistic regression (LR) and neural networks (NNs) learning algorithms are used for the first time in the context of SCA, and used to recover key information from various cryptographic algorithms.
- A new variation of the construction of the SM is presented which is advantageous where a large number of training examples is available, and empirically compared to alternative models.

1.2 Thesis Layout

The background chapter, Chapter 2, gives a wide ranging overview of power analysis in general. Power trace components and trace acquisition setups are explained, as well as why power attacks are feasible. Some of the many different attack distinguishers are briefly

examined and attack metrics explained. The effect of pre-processing techniques such as filtering is also explored here.

Next in Chapter 3 a through investigation of TA is performed. Using AES as a baseline comparison, different attack choices such as the chosen power model, the effect of data normalisation, feature selection and target value is explored. The required training set size for the target device is examined, and how to determine if a larger training set size will increase the attack feasibility. A brief examination of the how the platform effects the attack is also conducted, as well as the premise that similar devices have similar power consumption.

Unknown plaintext attacks are then conducted against AES in Chapter 4. Two methods are proposed, a higher order attack where multiple templates are built for different operations around where the key is used, as well as how to efficiently build templates directly on the sub-keys. The effect of masking in the first scenario is also examined.

TA against secure asymmetric algorithms are then examined in Chapter 5, again with the goal of using a single power trace only and no input or output information. It is shown how to utilise the difference in Hamming weight of multiplication and squaring operations to build templates that can recover key information from RSA and ECC implementations.

Finally in Chapter 6 various non-parametric machine learning methods are explored as alternatives to Gaussian based TA and SM. An empirical comparison is performed for two different datasets to determine the different strengths and weaknesses of the algorithms.

Chapter 2

Background

The field of SCA is by its very nature multidisciplinary, combining number theory for the analysis of cryptographic algorithms, engineering and signal processing for the acquisition and analysis of side channel information, and statistical analysis to extract secret information. While the statistical methods for information extraction are what this thesis is chiefly concerned with, the other steps are equally important in order to successfully mount an attack on a device. This background section deals with the *engineering* aspects of SCAs, from expected attack scenarios, to the physical aspects as to why the attacks actually work, as well as a brief examination of some signal processing techniques to reduce the noise of side-channel traces. The notation and success metrics as used throughout the thesis are also introduced here.

2.1 Embedded Systems

Embedded security is becoming ever more important as the world is increasingly connected. Many of us carry in our pockets smart-phones that can easily allow us talk, send texts, check our bank balances, surf the internet *etc.*, through WiFi, GSM, HSDPA, Bluetooth, NFC .... In our wallets we carry bank-cards, transit cards, access cards, all of which now have some sort of processing capability to enable its functionality. Newer passports now contain biometric data that is accessed via contact-less technology. All these devices contain (and leak) personal information which needs to be protected. These low power, low cost devices can be particularly susceptible to SCAs due to the simplicity of their underlying architecture. The leakage of information will only become worse as the new

generation of wearable "smart" devices become more mainstream and personal computing devices begin to communicate with each other. Looking beyond the security of personal devices, motor vehicles are an example where security is becoming ever more important due to the increasing electronic nature of cars. Loss of security in these *cyber-physical* systems could lead to events that are considerably worse than your bank account being emptied or your identity stolen. Interesting research in the area of embedded security for cars can be found in the proceedings of conferences such as ESCAR (`https://www.escar.info/`).

Due to power and cost restrictions, along with the fact that embedded devices often only need to perform specific tasks, powerful general purpose processors such as those found on desktop computers are usually not required. For security token type devices, small low-power 8 or 32-bit micro-controllers are often used. For example, ATMEL® have dedicated hardware encryption blocks on some of their low cost AVR XMEGA-A devices [13], as well as security focused CryptoMemory EEPROMS [12] (both of which have been attacked using power analysis in [121] and [15] respectively).

For large scale deployment dedicated ASICs are often used, where a higher initial development cost can be offset by the relatively lower cost of each individual unit both in power and monetary terms. For passively powered radio-frequency identification (RFID) devices this might be the only option due to power restrictions. Reprogrammable FPGAs also have interesting characteristics with regards to security applications, as they can be easily reprogrammed *in situ* should updates be required to protocol or algorithm specifications. While they have higher power requirements due to their reprogrammable nature, they are particularly useful for developing attacks, and countermeasures, such as with the SASEBO boards [190].

While in general SCAs are a threat for embedded devices, that is not to say that they can't be applied remotely to large-scale server systems either. Also it must be noted that an attacker might own the device under attack and that he stands to gain by extracting the secret information in it. An example of this could be breaking a Pay-TV system to show all available channels, or to arbitrarily add value to a transit or charge card such as performed by Oswald *et al.* in [169]. Hence, given that an adversary can possibly have unlimited access to an embedded security device, as well as controlling environmental conditions such as temperature, supply voltage, *etc.*, it is clear that creating secure embedded computing devices is far from a trivial problem which the straightforward implementation of cryptography alone cannot solve.

2.2 Cryptographic Algorithms

To secure communications and provide the functions specified in Table 1.1, various cryptographic protocols are required such as TLS. These are defined by standards bodies such as the International Organization for Standardisation (ISO) or NIST to allow for compatibility between devices. Cryptographic protocols, can be broadly viewed as the application of cryptographic algorithms in a specified manner. As mentioned in the introduction, the algorithms themselves can be roughly split into symmetric (private key) and asymmetric (private and public key) cryptography.

Symmetric cryptographic algorithms use the same key for encryption and decryption, and are generally quite fast hence are used for the bulk encryption of data. Block ciphers such as AES and DES or stream ciphers such as Grain, RC4 or A5/2 are examples of symmetric algorithms. Asymmetric algorithms use a different key for encryption and decryption, allowing the encryption key to be made public to allow anybody use it, however only the holder of the corresponding decryption key can unlock the data. RSA based on integer factorisation, DH or ElGamal based on discrete logs, and ECC based on the elliptic curve discrete log problem (ECDLP) are some examples of asymmetric algorithms. Due to the relative lack of speed when compared to symmetric algorithms, generally they are used for key exchange and message signatures to provide authentication, rather than encryption of the data itself. Hash functions form a third grouping, and provide an irreversible one-way function on the data. While no key *per-se* is used in the algorithm, they can be used as the basis for constructions to provide message authentication and data integrity such as the keyed-hash message authentication code (HMAC).

When implementing cryptographic algorithms, they are themselves composed of smaller building blocks. For example, finite field arithmetic is usually required in *classical* algorithms (*i.e.* algorithms that are *not* post quantum) as rounding leads to a loss of information and could lead to incorrect decryption. AES requires the multiplication of polynomials modulo an irreducible polynomial, while ECC is commonly performed in either a prime field $\mathbb{F}_p$ or binary extension field $\mathbb{F}_{p^n}$.

2.3 Side-Channel Attacks

In academic literature, classical *side-channel attacks* generally refer to passive timing, power or electromagnetic attacks, as outlined in Table 1.2 previously, or fault attacks

which encompass a wide range of attacker models and capabilities. Fault analysis (FA) can be quite powerful when used in conjunction with cryptanalysis methods or with passive attacks. There are other potential side-channels such as *acoustic* [14, 235] or photonic emission analysis (PEA) [73], however these somewhat more esoteric methods are not of relevance here.

2.3.1 Timing Analysis

Trivially, timing leakages could be something as simple as an array comparison when checking a login pin or password. If the password is checked byte-by-byte, then the length of time required to verify it is determined by the number of correct bytes at its beginning. They were the first published SCA, presented by Kocher at Crypto '96 [125], where he showed how to recover key bits from asymmetric algorithms by accurately measuring how long they take to execute. As execution time was dependent on the key value, secret information could be recovered. Initially timing attacks were most effective against simpler targets such as smart-cards *etc.*, however it was shown by Brumley *et al.* that remote attacks against servers were possible also [36]. It is not just asymmetric algorithms that are vulnerable however, certain implementations of the *MixColumns* operation in AES were shown to be vulnerable [127] also, and cache-timing attacks due to look-up operations when indexed by some secret value are also feasible [23]. Processor optimisations such as early termination of multiplication operations when the upper operands are zero can also lead to serious timing leakages [87]. It must also be noted that it is not just cryptographic algorithms that are vulnerable to timing attacks, protocol level implementations are also vulnerable as shown in the recent *Lucky Thirteen* attack by AlFardan and Paterson [9].

2.3.2 Power Analysis

Power analysis can be broadly divided into three categories with considerable overlap, with many attacks combining aspects from two or all of the areas.

Simple Power Analysis

SPA is the most straightforward type of attack, and can be used to trivially extract key bits from a power trace where key dependent operations are performed [126]. For asymmetric algorithms such as RSA [192], the obvious way of implementing the exponentiation is though repeated squarings and multiplications. If the power consumption difference

between these operations can be determined, then the key can be easily read out from a single trace. As the squaring operation can be performed more efficiently than a multiplication [149], for low power devices such as smart-cards this is a real concern. A similar analysis also applies to ECC [123, 152].

While in general, applying SPA to symmetric algorithms is not quite as straightforward, it is sometimes possible where conditional branches are present such as during the DES key schedule [126] or as an alternative to measuring timing information during an aforementioned vulnerable AES *MixColumns* implementation [127]. SPA is particularly useful however for selecting areas of time in which to perform differential or profiling attacks which allows for a significant computational saving.

Non-Profiling

Broadly speaking, non-profiling attacks partition the trace set based on a function of some key hypothesis. This is repeated for all valid keys and some statistical distinguisher is used to determine which is the correct guess. These non-profiling attacks are considered in more detail in §2.7 after the acquisition and make-up of power traces are examined.

Profiling

As profiling attacks are the main concern of this thesis and are dealt with in it later chapters, if suffices to say here that profiling attacks assume a somewhat stronger attacker model in that an identical or similar device is available to model the power consumption prior to the attack. How much control or knowledge of the key they are assumed to have over said device is open to interpretation per adversarial model, hence this assumption is not as restricting it may first seem. For example, in [169] a non-profiling attack was first used to recover a key prior to subsequently using the broken device to build templates. In [3], it was shown how a device with a faulty RNG suffices to build templates, while more recently in [137] it was shown how two devices with different unknown keys could be used. It was also suggested in [92] that public verification functions could be used to build templates using the device under attack itself. The first published profiling attack was presented by Fahn and Pearson [70], where they introduced inferential power analysis (IPA) to make detailed models of the power consumption of a device prior to an attack.

2.3.3 Electromagnetic Analysis

Electromagnetic (EM) leakages were first demonstrated as source of side-channel leakage by Quisquater and Samyde [184], and by Gandolfi *et al.* [77]. Rather than recording the power consumption of a device, an EM probe is used to measure the changing electrical or magnetic fields around a circuit due to current flow within it. While the analysis is quite similar to that as is performed for power traces, there are certain advantages to examining EM leakage. For example the ability to localise the acquisition to a specific region rather than measure the power of the chip as a whole, as well as, practically, not having to directly access the supply current which might not always be easily accessible. On the other hand, the probe selection, placement and orientation are extra factors to consider, as well the fact that low-amplitude signal might require decapsulation of the chip in order to measure it. While EM attacks are not examined in this thesis, the statistical analysis techniques are equally applicable to EM traces.

2.3.4 Fault Analysis

The area of FAs is quite broad and encompasses a wide range of attacks that all essentially take a target device outside of its recommended operating conditions in order to induce an error during the computation. Taking it outside its operating range can mean anything from increasing the ambient temperature, reducing, glitching or cutting the supply voltage, varying the clock frequency, blasting the chip with a laser, or even inducing an EM fault via the spark plug of a car [118], to name but a few. There is also the potential for inadvertent faults due to bugs in chip design as highlighted by the older Intel 80286 `popf` or Pentium FDIV bugs, a risk that increases with chip complexity.

Once an attacker can inject (or knows how to inject in the case of a hardware bug) a fault, some very elegant attacks can be derived such as the *Bellcore* attack of Boneh *et al.* [32]. While this attack can be performed by injecting a fault at any point in time, some require much more specific timing or even location based faults. Combining fault and power analysis can also lead to effective attacks such as previously mentioned attack an the ATMEL® CryptoMemory EEPROMS [15]. An overview of various FA and methods can be found in [16].

2.4 Trace Acquisition

The first step of any power attack is the acquisition of traces from a target device. The *quality* of the traces has a major bearing on both the feasibility of an attack, as well as the number of traces required to recover a key. Indeed version three of the DPA contest [211] was an acquisition competition to see who could minimise the number of traces required for an attack through the setup alone.

An example attack setup is given in Figure 2.1, where a controlling personal computer (PC) synchronises the sending and receiving of messages to the device under attack, while the power consumption is recorded using an oscilloscope. The external power supply and function generator are not strictly necessary, however they can help provide *cleaner* traces. Where a device is powered via universal serial bus (USB) there can be oscillations on the line due the PC power supply unit. Likewise an external function generator will likely provide an more stable clock than an on-board oscillator, even a very slight drift can cause de-synchronisation over the course of an RSA or ECC operation. Generally this is less of an issue for symmetric block ciphers such as AES or DES due to their shorter computation time.

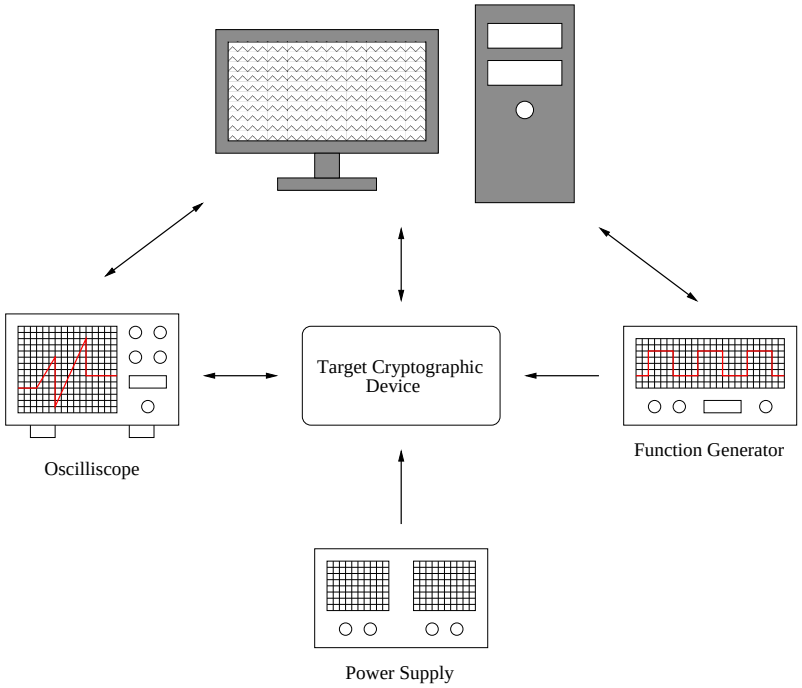


Figure 2.1: Example power analysis setup.

To measure the power consumption a resistor is usually placed in series with the power

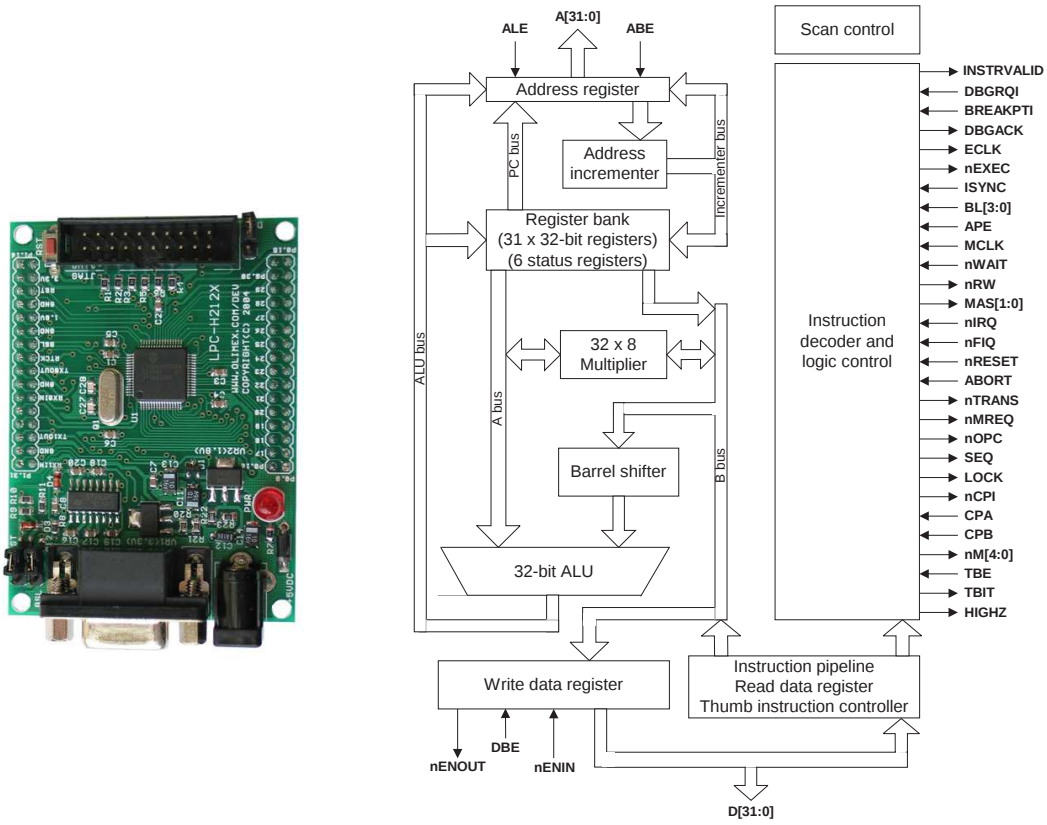
supply and the target chip, with the varying current drawn by the chip causing a change in voltage that can be measured. This causes a slight fluctuation at the device V_{dd}, however if a small resistor is chosen this can generally be safely ignored. The resistor can equally be placed on the ground line where a differential probe is not available. A proposal for an active power measurement circuit has also be demonstrated by Bucci *et al.* in [37], where they use Op-Amps to provide a stable voltage to the target device as well as amplifying the current measurement.

A consequence of using a small resistor to measure the drop in voltage across, is that only a small fluctuation is available to record. To prevent excessive quantisation noise, the power trace should occupy the entire range of values of the scope resolution, hence an amplifier might be required. Analog pre-processing with amplifiers, demodulators, filters, *etc.*, can reduce the quantisation noise by ensuring the recorded signal which contains the side-channel information occupies the majority of the range of the oscilloscope analog-to-digital converter (ADC). This was used to good effect in [119] to perform low-cost attacks on contact-less smart-cards. Another low cost attack setup was shown in [166], hence it is clear that SCA is easily accessible to a wide range of attackers.

Another important aspect is that of triggering, as the traces need to be aligned with each other for a successful attack. For controlled experiments, an output pin can be easily set high to allow consistent triggering at the point of interest. Any de-synchronisation is then likely due to offsets between the clocks of the device and the oscilloscope. However when attacking a *real-world* device a suitable trigger must be found (such as the communication bus) and the traces subsequently aligned.

Unless otherwise specified, the attacks in this thesis are performed against an ARM7 microprocessor [138], and is typical of what would have been found in a previous generation smart-phone. The software implementations of target algorithms were written by Dr. Mike Tunstall, and the traces were acquired using a LeCroy AP034 differential probe across a $10\,\Omega$ resistor on the V_{dd} side of the ARM7 microprocessor, using a LeCroy WaveRunner 104Xi oscilloscope. The clock frequency of the device was supplied by a 7.3728 MHz crystal oscillator, and the power supply was provided via the JTAG programmer unit rather than directly from an AC/DC converter which caused excessive noise. The sampling frequency F_s was set to $250\,\text{MSs}^{-1}$ and the 20 MHz bandwidth limiter of the scope was used. A dedicated trigger signal was used at the start of an acquisition to allow easy trace alignment. The ARM7 microprocessor was manufactured by NXP, and the target board was the LPC-H2124 from Olimex, as shown in Figure 2.2(a). Note this board had a dedicated

resistor for SCA in its V_{dd} line. All communication was performed through the serial port. *No* averaging was performed to reduce signal noise. The ARM7-TDMI core has a Von Neumann architecture with a single 32-bit data bus carrying both instructions and data. A block diagram of the core architecture is given in Figure 2.2(b) [138].



(a) Evaluation board.

(b) Architecture overview.

Figure 2.2: Target ARM7 microprocessor.

2.5 Power Trace Components

Complementary metal-oxide-semiconductor (CMOS) is the most widely used technology for constructing integrated circuits (ICs) such as the microprocessors, ASICs and FPGAs that are widely used for embedded security devices. Circuits typically consist of complementary pairs of metal oxide semiconductor field effect transistors (MOSFETs) as shown in the inverter circuit of Figure 2.3. The overall power consumption of a CMOS device can be viewed as containing static and dynamic power, $P_{total} = P_{stat} + P_{dyn}$. As only one of the transistors can be on at any time, this arrangement leads to a low static power

consumption, with P_{dyn} the dominant factor in the power consumption, Equation 2.1.

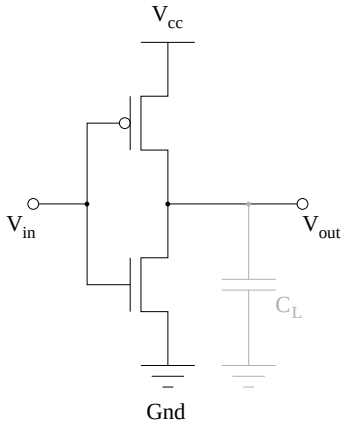


Figure 2.3: CMOS inverter.

$$P_{dyn} = f \cdot C_L \cdot \alpha \cdot {V_{DD}}^2 \tag{2.1}$$

Here f denotes the clock frequency, C_L the capacitive load, α the switching activity, and V_{DD} the supply voltage of the circuit. The switching activity α refers to the number of transistors that change state, which itself is dependent on the data and operations being performed, which is the underlying concept of SCAs. A good overview of the components of a power trace in the context of SCA can be found in [145].

Again following [145], from an attackers viewpoint the power consumption can be viewed as containing three parts as in Equation 2.2. The signal itself consists of a *constant* and *exploitable* part, P_{const} and P_{exp} respectively, with the random fluctuations due to natural variations and quantisation noise represented by P_{noise}. The signal-to-noise ratio (SNR) of a signal gives an idea of how *noisy* a signal is, which has a direct bearing on an attack outcome. The constant part of the signal, or the direct current (DC) component, is that which is, for a given point in time, the same regardless of what data is processed, *i.e.* the operation being performed or a bus load. As this is constant across all the traces it has no effect on the outcome and can easily be removed. The exploitable part due to bits switching is that which an attacker seeks to exploit. Note that where only a subset of the bits switching is targeted, the remaining bits can be viewed as algorithmic noise as, where the data being processed is random, the power consumption of these bits follows a normal distribution. This is true also for the naturally occurring noise P_{noise} which has been shown in many works to follow a normal distribution also.

$$P_{tot} = P_{const} + P_{exp} + P_{noise} \tag{2.2}$$

An example of this is given in Figure 2.4 where the distribution of a single point in time of 10 k different traces from the ARM7 microprocessor is given. It is clear that the data follows a Gaussian distribution, with a mean and standard deviation of 135.67 mV and 1.64 mV respectively. Note that the x-axis of the histogram is labelled mV as the values represent the voltage drop across the shunt resistor which is proportional to the power consumption as explained in the previous section. Each bin is a distinct voltage level as recorded by the oscilloscope. While the 8-bit ADC of the oscilloscope has 256 distinct levels, for the single time point of the trace examined only 18 of those values are actually utilised. This is as a larger range of values occur over the length of the power trace hence the vertical oscilloscope resolution must be set accordingly (over the entire trace $>$ 220 of the ADC levels are utilised). It does mean however, that the quantisation noise can be greatly reduced where the location of the target power consumption can be narrowly focused.

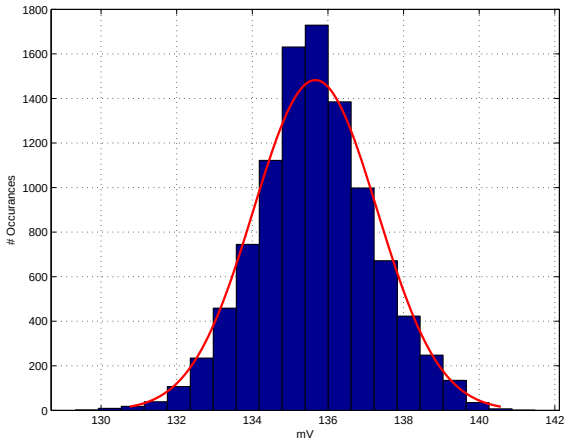


Figure 2.4: Gaussian noise distribution of a trace point.

In order to attack a device, it can be beneficial for an adversary to model the expected power consumption. As P_{dyn} is the dominant factor of the power consumption[1], use of the Hamming distance is a straightforward and effective method to model P_{exp}. The Hamming distance is a count of the number of bits that change state between two points in time as shown in Equation 2.3, where $\mathcal{R}_{j,i}$ is the i^{th} bit of register $\mathcal{R}$ at time j. While the previous state is usually required when attacking hardware implementations, in the case of software it is not always available or required. For example the previous value on a

[1]Note that as manufacturing processes decrease, it becomes increasingly important to consider P_{stat}.

bus might be an op-code call hence will be some constant unknown value when attacking the output of that operation. Alternatively, lower end microprocessors sometimes pre-set their data-bus to an all zero or one state. In this scenario the Hamming weight model can be used, which is essentially the Hamming distance model with an all zero previous state. In fact when attacking software implementations, the Hamming weight models is often sufficient. More complex models are available, such as suggested in [179] where they apply a weighting to the $1 \rightarrow 0$ transition which consumed a slightly different amount of power than the $0 \rightarrow 1$ transition for their setup. This can obviously be further extended to have a vector of weights for each separate bit. These weights need to be calculated somehow which implies some sort of profiling or regression based methods. In general however, despite their simplicity, the Hamming weight and Hamming distance models provide accurate enough approximates to allow successful SCA in many scenarios.

$$HD(\mathcal{R}_{t_1}, \mathcal{R}_{t_2}) = \sum_{i=0}^{n-1} \mathcal{R}_{t_1,i} \otimes \mathcal{R}_{t_2,i} \tag{2.3}$$

The point in time used to generate the histogram in Figure 2.4 was selected by taking the location that returned maximum SNR after calculating it for each time point in the trace. The SNR was calculated following Equation 2.4 [145, Ch. 4], and it was assumed that the leakage followed the Hamming weight model, with the output of the first *S-Box* of the first round taken as the signal. To calculate Equation 2.4, the mean of each Hamming weight value is first computed. For σ^2_{signal}, each trace is replaced by its equivalent mean. The variance is then calculated across the mean replaced signals. For σ^2_{noise}, the corresponding mean is subtracted from each trace with the variance calculated across the resultant noise vectors. To put the noise of the power trace in context, the SNR at this point was calculated to be 0.97. As the SNR also depends on the utilised power model (*i.e.* exploiting less bits increases the algorithmic noise), the SNR when the data was assumed to leak only a single bit of information was calculated at 0.14. For reference, the SNR for the Hamming weight and bitwise power models after the post-processing stages as described in §2.10, were 0.23 and 1.66 respectively.

$$SNR = \frac{\sigma^2_{signal}}{\sigma^2_{noise}} \tag{2.4}$$

2.6 Notation and Methodology

Throughout the thesis, the following notation is utilised. A set of power traces x is of size $m \times n$ where m is the number of traces (or samples) and n is the number points (or features) in each trace. A single trace (*i.e.* row vector) from this set is denoted $x^{(i)}$, with $x_j^{(i)}$ a single point in time of this trace, or x_j a single point in time across the entire trace set. The plaintexts (*resp.* ciphertexts) corresponding to this trace set are denoted by p (*resp.* c), with a single input again denoted by $p^{(i)}$ (*resp.* $c^{(i)}$), and the valid set of values by $\mathcal{P}$ (*resp.* $\mathcal{C}$). The secret (or key) s from the set $\mathcal{S}$ is estimated by an adversary by $\hat{s}$ after making a hypothesis on a number of various values for s^*. Each trace $x^{(i)}$ can be assigned to a class $y^{(i)} \in \mathcal{K}$ where a class can be some intermediate value or the result of a leakage mapping such as the Hamming weight. The unique values of y are denoted by $o^{(1)}, o^{(2)}, \ldots, o^{(|\mathcal{K}|)}$, where $|\mathcal{K}|$ is the number of elements in $\mathcal{K}$. Estimates and guesses on y are denoted by $\hat{y}$ and y^* respectively as with s. The individual bits of the binary representation of y are denoted by y_j. While performing a SCA it is usual to only attack a portion of the secret at a time, however no distinction is made here between partial and full secret recovery as it should be clear from the context what is being referred to. Additional notation is introduced as required.

When selecting a subset of m traces from a larger acquisition set to perform an attack, they are always randomly selected to avoid any potential bias. This applies in all cases whenever a subset is required, including when selecting both training and testing sets when conducting profiling attacks. Training traces are used for the building of profiling models, with a separate testing (or attack) set used to verify to model.

With regards the profiling attacks on the ARM7 microprocessor, for AES the training and testing sets are taken from the same device but in two separate runs. When acquiring the training traces, a random key is used for each trace, while for the testing traces a constant key is used. For the attacks on the Montgomery multiplication operation, the traces were acquired in the same run and are randomly allocated to training and test sets each time an attack is performed. For the attack on ECDSA the training and testing sets had to be acquired in different acquisitions runs, as will be clear in §5.7.

2.6.1 Cross Validation

Cross validation is a widely used method for estimating prediction error from the training data, as well as for choosing any learning parameters that may be required. Rather than

simply calculating the training classification error and using that as a basis for parameter selection, the training set is split up into CV sets, with $CV - 1$ sets used as training data and a single set used for testing, with the model trained CV times. This removes the bias of testing on the same data that has been used to generate the templates which generally will predict a lower error than will be obtained on unseen data [95, Ch. 7]. Once the parameters are selected, the templates are then generated over the entire training data set. Obviously the greater the cross validation number the greater the computational complexity, with the *leave one out* method where $CV = m$ generally only used for very small data sets. Care must be taken not to choose too small a CV parameter, where the reduction in the training set size is such that the individual classes can no longer accurately modelled. Any pre-processing of the data such as normalisation or transforms must be calculated from the *reduced* CV training set.

As the acquisition of traces for this thesis is not restricted, cross validation is not used and a separate testing set is used for parameter selection. A third validation set could also have be used to estimate the expected error on unseen data after all parameters are selected, but in practice there was little difference between this validation error and that of the testing set. The exceptions with regards to cross validation are for the parameter selection in the attack on ECDSA in §5.7, and for support vector machines (SVMs) in §6.4 where a grid search is implemented for parameter optimisation.

2.7 Non-Profiling Attacks

While there are many different methodologies to perform a non-profiled SCA, when attacking block ciphers they all roughly follow the same sequence of steps. Initially some target intermediate value or operation y that is a function of the secret s and some known input p is chosen. Many attacks are also equally valid against a known output ciphertext c. A suitable target value when attacking block ciphers is often the output of a *S-Box* due to its non-linear properties [181]. For all possible values of the secret $s^* \in \mathcal{S}$ a hypothetical intermediate value is calculated and a power model applied if required. Some statistical distinguisher is then used to determine the most likely key guess $\hat{s}$. As generating hypothesis for all possible keys is infeasible for any modern cipher, the key is broken into appropriately sized *chunks* and the attack is simply performed multiple times.

While the same steps are used for attacking both hardware and software implementations, when attacking hardware it will be likely that some previous register value will have to

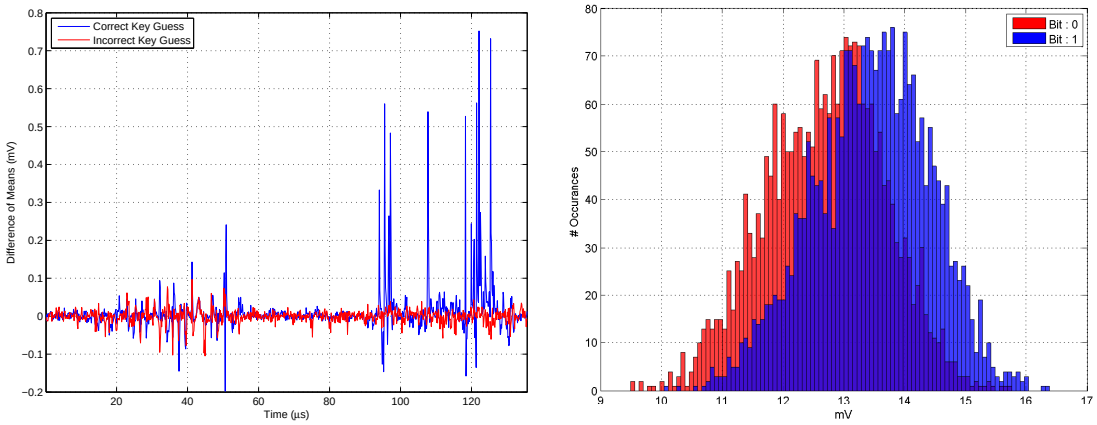
be used as already mentioned. Generally speaking, attacking hardware targets will likely require a greater number of traces to succeed due to both the nature of the leakage, as well as the parallel nature of designs which induces a larger amount of algorithmic noise on the traces [207].

2.7.1 Differential Power Analysis

The original DPA attack introduced by Kocher *et al.* [126], simply partitioned the traces into two sets based on a projected bit value $y = \mathcal{F}(p, s^*)$. Examining the difference between the means of these two sets as in Equation 2.5, when $s^* \equiv s$ then peaks should occur as the sets have been correctly partitioned. These peaks occur at the points in time wherever y is processed, and for $s^* \neq s$ no peaks should occur although so called *ghost peaks* can occur due to relationships in the data.

$$\Delta = \left(\frac{\sum_{i=1}^{m} y^{(i)}\, x^{(i)}}{\sum_{i=1}^{m} y^{(i)}} \right) - \left(\frac{\sum_{i=1}^{m} \left(1 - y^{(i)}\right)\, x^{(i)}}{\sum_{i=1}^{m} \left(1 - y^{(i)}\right)} \right) \tag{2.5}$$

As an example to highlight the effectiveness of the attack despite its simplicity, in Figure 2.5 an attack against an ARM7 microprocessor implementation of AES using the setup as previously described is shown. An overview of the AES algorithm is given in Appendix A. Only two keys are tested, the correct one and a single incorrect value. The target bit value is chosen as the least significant bit (LSB) of the *S-Box* output in the first round such that $y^{(i)} = \text{bitget}\left(S\left(p^{(i)} \otimes s^*\right), 0\right)$. The correct key can be clearly distinguished and the peaks in Figure 2.5(a) show the points in time where the LSB is processed (note the majority of the leakage actually occurs during the use of y in the *MixColumns* operation rather than the *S-Box* output). The histograms of each set at the point of the largest peak are given in Figure 2.5(b) and it can be seen that while there is a significant overlap between the sets, they can still be clearly distinguished. While $m = 5\,\text{k}$ traces were used for the example, for this set of traces the attack is feasible with considerably less traces. The large data set was to provide a clear visualisation of the histograms. Some post-processing as described later in §2.10 was also applied to the traces prior to the attack.



(a) Difference of means traces.

(b) Histogram at peak difference of means.

Figure 2.5: Difference of means attack.

2.7.2 Correlation Power Analysis

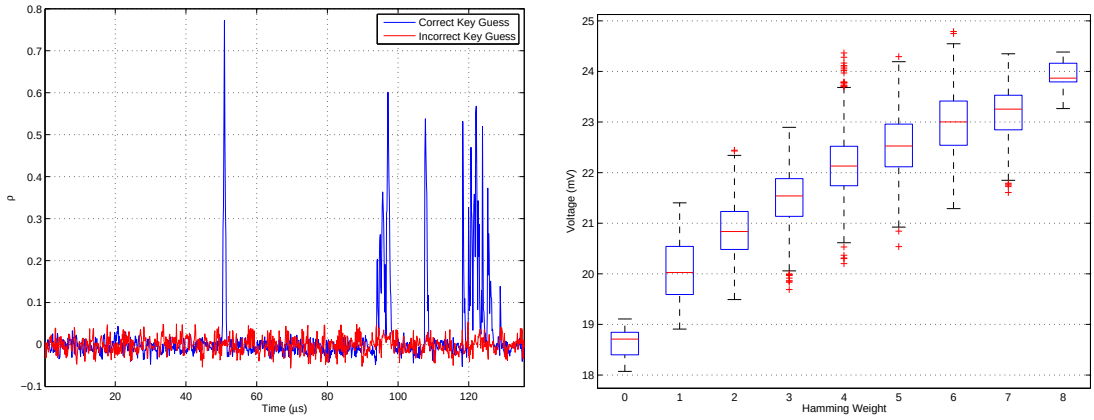
One of the most widely used methods of performing SCAs is with the use of Pearson's correlation coefficient in a CPA attack [33, 34]. This provides a measure of the linearity between two normally distributed variables, *i.e.* the recorded power traces and a hypothetical power consumption based on a key guess. Where an accurate power model is used this can be a very powerful attack which can recover a secret in the presence of considerable noise. Conversely, where a poor or inaccurate model is used then an attack might not be possible at all.

The formula for the correlation coefficient is given in Equation 2.6 where y is the hypothetical power consumption, and like the DPA attack, this is applied to every point in a trace. For a given correlation coefficient, confidence intervals can also be used to determine how many traces an attacker would expect to need in order to recover a secret following the rule of thumb as given in [145, Chap. 6].

$$\rho = \frac{\sum_{i=1}^{m} \left(x^{(i)} - \bar{x}\right)\left(y^{(i)} - \bar{y}\right)}{\sqrt{\sum_{i=1}^{m} \left(x^{(i)} - \bar{x}\right)^2} \sqrt{\sum_{i=1}^{m} \left(y^{(i)} - \bar{y}\right)^2}} \tag{2.6}$$

Using the same $m = 5\,\text{k}$ traces AES as before, a CPA attack is performed targeting the output of the *S-Box* once again and assuming a Hamming weight power model, $y = \text{HW}\left(S\left(p^{(i)} \oplus s^*\right)\right)$. There are multiple peaks in Figure 2.6(a) representing different parts of the AES algorithm where y is used. The peaks are somewhat clearer than in the DPA case as the leakage due to multiple bits is being utilised. Box-plots of the trace values at the

point corresponding to the peak correlation and grouped by their Hamming weight are given in Figure 2.6(b). The linear relationship can be clearly seen and shows that the Hamming weight model is pertinent choice for this device.



(a) Pearson's correlation coefficient traces.

(b) Boxplot at peak correlation.

Figure 2.6: Correlation attack.

When attacking hardware devices such as FPGAs, or a smart-card with a co-processor, it is beneficial to attack as many bits as is reasonably possible from a computational viewpoint, as this reduces the algorithmic noise of the traces. It was shown in [34] that when attacking a subset of bits b_{sub} in a bus of width b, the maximum correlation is given by Equation 2.7. An example of the practical effect of this is given in [214] where an increasing number of parallel DES *S-Boxes* in a smart-card coprocessor are attacked.

$$\rho_{sub} = \rho\sqrt{\frac{b_{sub}}{b}} \tag{2.7}$$

From a practical viewpoint, performing CPA on larger datasets where the number of samples $m >> 1\,\text{k}$ and the number of time points $n >> 10\,\text{k}$ where each point will typically be a single or double precision value of 4 or 8 bytes, can require a significant amount of memory if the traces are loaded directly into memory for the calculation. Hence it is often more efficient to implement to correlation function recursively as outlined in [119], or with the proposed efficiency improvements in [139]. This has the added advantage where the peak correlation is being plotted as a function of the number of required traces as the intermediate correlations are being calculated regardless.

2.7.3 More Non-Profiling Attack Methods

While the two attacks as described are widely used SCAs, there are many others. Any statistical distinguisher that can be used to differentiate set partitions can be used as an attack method. These are typically referred to as univariate attacks as they exploit the leakage of a single time point of a trace. There are many variants of the original difference of means (DOM) attack such as:

All-or-Nothing DPA [151] This is a straightforward extension of the DOM attack which looks to maximise the DPA peak by partitioning the traces into sets such that $\{x\,;\, y_i = 0\ \forall i\}$ and $\{x\,;\, y_i = 1\ \forall i\}$. This has the disadvantage of leaving traces where the bits of y contain both 0 and 1 unused for that particular DOM trace however.

Generalised DPA [151] Here the traces are partitioned into two sets based on if the Hamming weight is greater or equal to $\frac{b}{2}$ where b is the number of bits under consideration and is given by $b = \lceil \log_2 (|\mathcal{K}|) \rceil$, with $\mathcal{K}$ the set of values y can take.

Enhanced DPA [27] Multiple DOM attacks are performed seperately in an enhanced DPA attack for each bit y_i, with the resultant DOM curves summed in order to maximise the leakage utilisation.

T-Test DPA [53] This utilises the popular *t-test* distinguisher which is a difference of means test which also accounts for the respective variance of the two sets. In the context of SCA, this generally will be more accurate than a straightforward DOM test and require fewer samples to succeed. It can also be used in conjunction with any of the DOM type attacks.

Partitioning PA [131] In partitioning power analysis, the traces are split into multiple sets based on some power model and a weighted sum of means calculated. This is quite similar to CPA, however weighting coefficients must be selected.

Absolute Sum DPA [61] This is quite similar to the *enhanced DPA* method, except that absolute values are summed to ensure bit-dependent leakage doesn't cancel out.

Variance power analysis [204], and differential cluster analysis [20] have also been suggested, with the clustering method actually a form of unsupervised machine learning. Equally, rather than using Pearson's correlation coefficient which assumes the data is normally distributed, non-parametric rank correlation methods such as Spearman's, Kendall's Tau or the Gamma statistic can be used as suggested in [20], or Gini correlation as exam-

ined in [201]. These rank correlation methods don't assume a linear dependency between the model and the trace data, rather an ordinal one. This can be advantageous in hardware scenarios where the leakage is more complex. Other useful non-parametric distinguishers that have been proposed for SCA include mutual information analysis (MIA) [80], regression analysis [58], and the Kolmogorov-Smirnov test [227], each with its own advantages and disadvantages. For example, an interesting feature of MIA is its generality and its natural extension to multivariate attacks [79], however optimal parameter selection is non-trivial [217] and poor choices can result in the attack failing.

With a wide range of distinguishers available, the obvious question to ask is which is "best" [98]. Due to the fact that SCAs are by their nature implementation dependent with the underlying platform and target algorithm dictating the strategy, no single distinguisher is most efficient in all cases. There are a number of publications which investigate the relative merits of distinguishers through both theoretical and empirical analysis, for example [61, 82, 141, 146]. Many of these show that the attacks can be viewed as being asymptotically equivalent, and can be re-written as a function of each other.

2.8 Comparison Metrics

To compare various SCA methods some sort of metric is required. Two commonly used methods are the *success rate* and the *guessing entropy* [206]. To calculate the success rate, multiple attacks are performed with the results averaged to estimate how the attack would perform given a random set of traces from the same setup. A 1^{st} order success rate returns on average how often the correct key is recovered as a function of the number of traces. A 2^{nd} order success rate is how often the correct key is selected among the two most likely keys, *etc.*. The guessing entropy is also calculated as a function of the number of traces, and returns the expected number of key guesses required after applying the distinguisher. This allows an evaluation of attacks where maybe the majority of keys can be quickly eliminated, however some (possibly related) keys are harder to distinguish between. An information theoretic measure is also introduced in [206] to evaluate leakages independent of adversarial models. Another comparison framework was introduced in [225] which, in addition to key ranking similar to the success rate, also suggests the use of distinguishing margins and standard scores.

With regards to comparison of profiling attacks which are the focus of this thesis, it is of interest to compare the efficiency of both the profiling and attack stages. To compare

the profiling step various empirical measures were suggested in [81], while in [205] the information theoretic measures of [206] were suggested. In both cases the success rate was used in the classification step.

In this thesis, the goal is the minimise the expected error given a single trace to recover a secret. This is equal to 1 minus the success rate for a single trace. Intuitively, minimising the expected error for a single randomly selected trace will also maximise the success rate where multiple traces are available for secret recovery. Wherever the success rate is used (*e.g.* in an amplified TA) then it is the 1^{st} order success rate. Learning curves as commonly used in machine learning are used here rather than information theoretic methods, with the efficiency of the training and testing stages evaluated using the training and test error rates. The training error is the error when classification is performed on the data set used to build the profiling model, with the testing error the error of a separate trace set unused in the profiling step. The learning curves are these error rates plotted against some variable parameter such as the number of samples or features used to build the model.

An advantage of using learning curves is that the bias-variance trade-off of the model can be easily visualised allowing the selection of parameters to minimise testing error. A high bias model (under-fitting) can be seen by high training and test error, while a high-variance model (over-fit) can be seen by low training error and a high testing error [95]. Hence the appropriate steps can be taken to improve the classifier rather than blindly trying everything. For example, to improve a high variance model an appropriate approach would be to increase the training set size, or maybe reduce the number of features selected. Conversely, given a high bias model the addition of extra training samples will unlikely improve the results and the addition of extra features is a more suitable path to take.

Finally, as previously mentioned in §2.6 when outlining the notation used, when attacking some cryptographic algorithm via SCA the key is usually recovered step by step. For example in the case of an embedded software implementation, if attacking AES the architecture of the algorithm lends itself to attacking the key a byte at a time, whereas if targeting DES in the same scenario then 4-bits might be more appropriate. For asymmetric algorithms such as RSA and ECC the key is generally recovered bit-by-bit, with an attacker maybe using the knowledge of the recovered key bit to extract the next one. In this work, unless otherwise specified, the error rates are for the recovery of a single byte of the AES key or a single bit of an asymmetric algorithm.

2.9 Countermeasures

Various countermeasures have been proposed to protect against SCA, however in practice provable security is difficult to achieve and a more realistic goal is to make the financial cost of breaking a system greater than the projected gain. Depending on the adversarial threat model and the algorithm under consideration, countermeasures based on secure logic styles [213], masking [41, 85] or randomisation [52] might be appropriate, to name but a few. In general a countermeasure will have some cost associated with it, such as longer computation time, extra silicon or FPGA utilisation, increased power consumption *etc.*., which must be factored into account when designing a secure system. Further detail is provided where relevant in later chapters. As the use of countermeasures can be overcome in a variety of ways, such as with *higher-order* attacks, a combination of them is usually preferred where possible. Finally it is worth pointing out that Cryptography Research has wide ranging patent claims on many countermeasure implementations.

2.10 Trace Pre-Processing

The success rate of SCAs can be improved through the application of trace pre-processing methods prior to performing the attack itself. These can be filtering methods to reduce noise on the raw traces or transformations to project the traces into a subspace. Another application of pre-processing methods is to reduce the computational complexity of an attack by removing redundant information prior to performing the attack without a significant loss of information. This can have large effect on both the length of time it takes to perform the attack, as well as the storage requirements. It must be noted however, that the ARM7 microprocessor traces used for the attacks here allow for relatively successful attacks without any pre-processing. Noisier platforms such as FPGAs, or implementations with noise inducing (or desynchronising) countermeasures will have a greater benefit from pre-processing. Pre-processing can also lead to attacks where previously none might have been expected or possible. For example, Agrawal *et al.* in [2] apply demodulation to recover amplitude modulated EM key dependent signals.

2.10.1 Direct Current Component Removal

Ideally the DC component of a power trace is unchanging across the entire trace set hence removing it should have no effect on the outcome of a SCA. However in reality a DC

drift can sometimes be observed across the length of a power trace, particularly when recording long trace vectors. Even more noticeable is the DC drift across a large set of traces acquired over a day or more. As the ambient room temperature changes, this can have a visible effect on the DC offset of the power traces. Likewise the temperatures of both the target device, and the oscilloscope probes will affect the DC component value, hence equipment should be allowed "warm up" prior to acquisition by recording dummy samples.

A straightforward way to remove the DC component at source is to set the oscilloscope probe to use alternating current (AC) coupling. A capacitor within the oscilloscope is then used to block the DC component of the analogue signal collected by the probe before it is ever converted to a digital signal via the ADC. This might not always be desirable however, hence two methods for removing the DC component are now examined:

Mean Subtraction: This is the easiest way of removing the DC component and simply consists of subtracting the average value of a trace from each point in the trace. Generally this method is very effective, however where there is a significant DC difference between various sections of the trace, a DC component will still remain post-subtraction. It has the advantage of affecting all points in the trace equally however.

High-pass Filter: A high-pass filter can be used to remove the DC component of a signal [200, Ch. 14], however its effectiveness depends on the filter parameters. Due to the vast number of filter types and parameters, the comparison here is limited to examining the application of windowed linear-phase finite impulse response (FIR) filters while using a Blackman window [94]. The choice of a Blackman window was due to its slightly wider lobe and greater stop-band attenuation, however experimental evaluation found little difference in practice between it and other window shapes such as the Hamming or Bartlett. Fixing the attenuation at the cut-off frequency F_c to 6 decibel (dB), the filter order N and F_c are varied to see the effect on the traces.

As the end goal of any pre-processing is the improve the practicality of a SCA, the success rate of a CPA attack against the first *S-Box* in the first round of AES is calculated for each method. While there are limitations in using this as a metric for comparing pre-processing methods (what is optimal for a CPA is not necessarily optimal for other attacks), it is sufficient for the purposes here as later attacks will be compared on the same data set anyway. Figure 2.7 gives the success rate for various N and F_c filter parameters, compared to subtracting the mean and the original data set. All possible bytes are hypothesised for

the key, and the correlation is calculated across the entire length of the first round (25 k points in total).

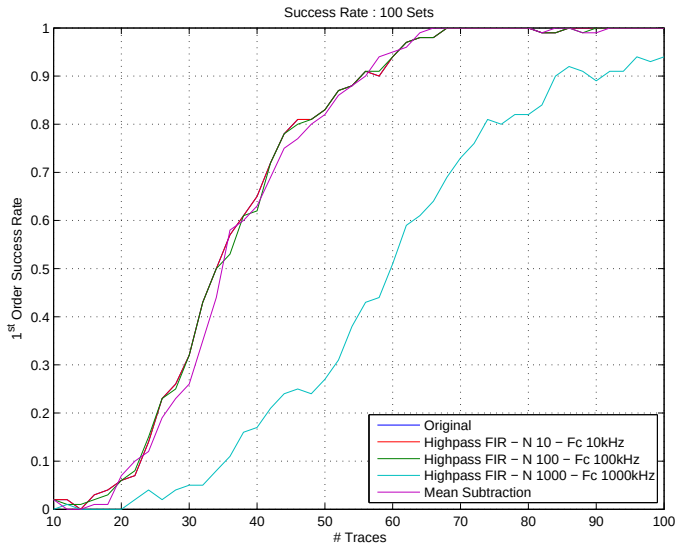


Figure 2.7: Success rate of DC removal methods.

It can be seen that simply subtracting the DC component, or filtering with $[(F_c = 100\,\text{Hz}, N = 100), (F_c = 100\,\text{Hz}, N = 100)]$ high pass filters makes little difference to the success rate of the attack. However for the filter with $(F_c = 1\,\text{MHz}, N = 1\,\text{k})$, there is a significant drop off in the success rate. Note that these are not particularly suitable filter parameters to remove a DC component in any case, as many other low-frequency components are also attenuated. They were chosen to highlight that poor filter selection can have a negative effect on SCA. As the effect on the success rate in minimal for the other options, for all following attacks on the AES algorithm implemented on the ARM7 microprocessor, the DC component is simply removed by subtracting the mean as this is computationally the fastest method.

2.10.2 Filtering

As outlined in §2.5, noise is always present on power traces due to both acquisition noise (such as quantization) and randomly occurring noise. This noise lowers the overall SNR and increases the number of traces required to perform a successful SCA. Low-pass filters can be used to remove noise at high frequencies thereby increasing the SNR, hence the effectiveness of the attack. Many oscilloscopes also provide an analogue bandwidth limiter (generally about 20 MHz or 25 MHz) that suppresses signals above that frequency. This can be quite effective at increasing the SNR of a trace as spurious noise is removed

before digitisation, possibly allowing a smaller vertical voltage range to be selected which in turn reduces the quantization noise. As mentioned previously, custom filters can also be implemented in the analogue domain in an attempt to reduce quantization noise, however we are only concerned with digital filters here. Bandpass filters are another option to reduce noise on a trace. As shown in [145, Ch. 4], the peak that occurs on the rising edge of a clock cycle is enough to perform an attack as this is where the majority of the power consumed due to bit transitions occurs (glitching is also another source of power consumption that must be accounted for, particularly when implementing countermeasures [147]). As these peaks occur at the clock frequency, a bandpass filter can be used to remove frequencies outside this range. The use of filters to reduce noise prior to performing an attack, while common in the cryptographic community, is not extensively dealt with in the literature, with [17, 132, 170, 202] some of the papers that directly look at application of digital signal processing (DSP) methods to enhance SCA. However the selection of filter parameters is dependent on the data at hand. Hence there is no all-encompassing set of filters and parameters that can be recommended for all attack setups.

Moving Average Filter: While an extremely easy and efficient filter to implement, the moving average filter (MAF) is actually optimal for reducing random *white* noise while retaining a sharp step response [200, Ch. 15]. However for frequency domain signals it is among the worst types of filter so should not be used where frequency separation is required. The MAF is simply a rectangular windowed FIR filter. This allows for an extremely fast implementation as a filtered point of a trace, x_i, can be used to calculate x_{i+1} with only a single addition and subtraction. A longer window length allows for a cleaner trace as the noise reduction is equal to the square-root of the window length (*i.e.* a rectangular window length of 100 will reduce the noise by a factor of 10). However the longer the window length, the poorer the step response which can impact an attack as seen for the case where $N = 101$ in Figure 2.8. This can be compensated somewhat by an attacker by increasing the sampling rate, trading acquisition and processing time, and storage space for noise reduction. While power traces will generally have Gaussian rather than white distributed noise, due to the simplicity and speed of the filter it is worth examining.

Lowpass Filter: The effect on the success rate of a windowed FIR low-pass filter, again using a Blackman kernel, of different window lengths is given in Figure 2.9. It is less efficient to implement than the MAF as it does not allow for a recursive implementation due to the shape of its window. It performs similarly in the time domain with regards noise suppression while being far superior for frequency sepa-

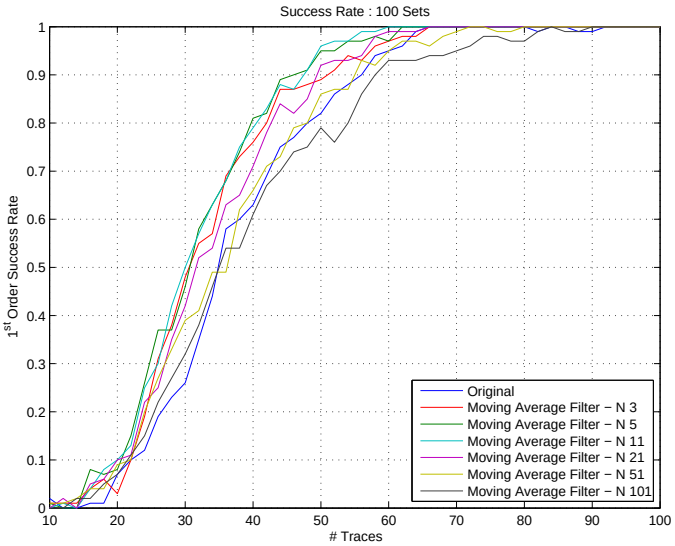


Figure 2.8: Success rate of moving average filters.

ration. Once again, the cutoff frequency is kept constant, and the window length N which is varied. As the clock frequency of the ARM7 microprocessor is set at 7.3728 MHz, the cut-off frequency of the filter F_c is chosen at 10 MHz suppressing any frequencies above this range including clock harmonics. It can be seen that all filter kernel lengths reduce the number of traces required for a given success rate, even the $N = 101$ point filter which in the MAF case caused a large reduction in the attack effectiveness.

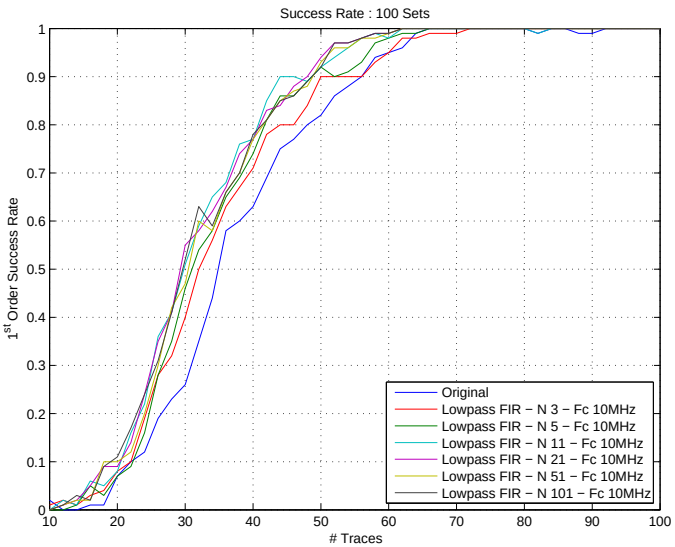


Figure 2.9: Success rate of lowpass filters.

Bandpass Filter: Once again the Blackman FIR filter is used, and the bandpass filter passband frequencies are set to ±500 kHz either of the clock frequency. As can be seen in Figure 2.10, in this case larger window lengths actually increase the success rate of the attack as a sharper cut-off frequency is achieved. While the removal of the DC component is not strictly necessary if using a bandpass filter as it will be attenuated anyway, for consistency in the comparison with the other options the same DC-free traces were used.

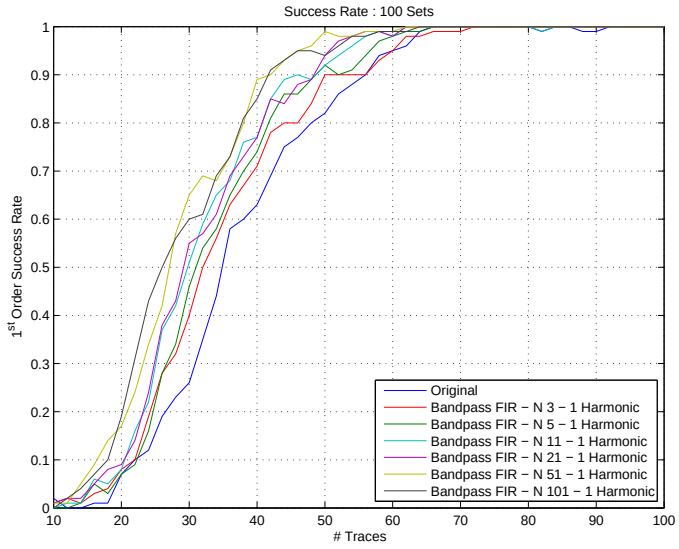


Figure 2.10: Success rate of bandpass filters.

The most effective of each of three filter types are compared in Figure 2.11. The application of a bandpass filter about the clock frequency leads to the best CPA attack for this particular setup, indicating that the interesting power leakage largely occurs at the clock frequency. Hence for all attacks on the AES algorithm implemented on the ARM7 microprocessor in the following chapters, the mean is first subtracted to remove the DC component, and an $N = 51$ point Blackman FIR bandpass filter with the passband set to ±500 kHz about the clock frequency is applied to reduce the noise on the trace.

2.10.3 Trace Reduction

The processing and storage requirements of a SCA are directly proportional to the length of a power trace, hence dependent on both the algorithm being targeted and the sampling rate. In the case of the traces under consideration, with a sampling rate of 250 MSs^{-1}, each round of AES has ≈ 25 k points therefore requiring over 250 k points per trace, with each point requiring 8 bytes of storage to store in double precision format. In the case

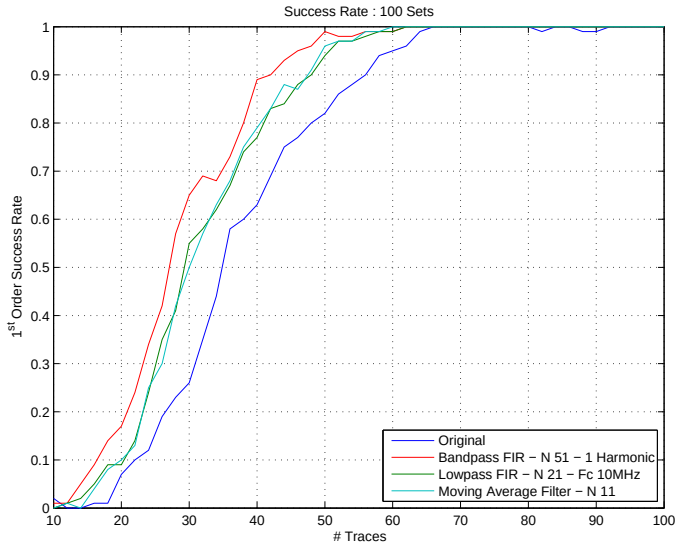


Figure 2.11: Success rate comparison of various filters.

of asymmetric key algorithms such as RSA or ECC, storage and processing requirements can be orders of magnitude larger due to the longer computation times. Reduction of the sampling rate is one method to reduce the trace length (synchronous sampling methods have also been shown to be effective [166]), but this is constrained by the Nyquist theorem which states that the sampling rate must be at least twice the maximum frequency in a system for an accurate representation of the signal [200, Ch. 3]. Due to the clock harmonics in the system, the sampling rate needs to be considerably higher than the clock to prevent aliasing on the traces. Examining the frequency response of the traces under consideration, 11^{th} order clock harmonics were still visible around 81 MHz (although greatly reduced in amplitude) hence the sampling rate of 250 MSs^{-1} was chosen.

In [145, Ch. 4], it is shown that only a single point from the duration of a clock cycle is required to perform SCAs. This allows the removal all all but one point per clock cycle in a trace without adversely affecting an attack outcome as mostly only redundant information is removed. It also has the advantage of realigning power traces acquired from a device where there is a very slight clock oscillator drift. These compressed traces allow for both quicker attack evaluation, as well as reduced storage requirements. A method to reconstruct clock edges from a power trace is given in [65], however for the AES traces here, $\lfloor \frac{F_s}{F_{clk}} \rfloor$ is used to approximate how many points per clock. This will not exactly split up the trace into clock periods unless it is an even division, however as the floor rather than ceiling function is used no data will be lost. Note this can *only* be used for deterministic algorithms with no key or data dependent branches as de-synchronisation

will occur otherwise. For the ECDSA traces of §5.7 the method of [65] is used. Various methods are outlined in [145, Ch. 4], a few of which are now outlined:

Maximum point: It can be assumed that the maximum point in a clock period is where the most bit transitions occur hence this point can be used as representative of the clock period as a whole.

Integration: This method sums up all the points in a clock period and uses this value to represent to clock period. This can decrease the SNR where many points consisting of mostly noise are summed together.

Sum absolute values: This too sums up all the points, but the absolute value is take prior to summation.

Dot product: This is equivalent to the summation of the squares of each point. The idea here is that the signal component will amplify itself through the multiplication hence the summation with points consisting of mostly noise will have a lesser effect.

Using the DC free and bandpass filtered traces from before, the success rate is once again calculated using the different trace reduction methods. As illustrated in Figure 2.12, all the summation trace reduction methods actually decrease the attack effectiveness due to the noise components consisting of a greater portion of a trace point post compression. Selecting the maximum point per trace on the other hand increases the success rate of the attack slightly as small desynchronising effects are removed, hence this compression technique is combined with mean subtraction and the use of a bandpass filter for the remainder of thesis where this set of traces are used.

2.10.4 Trace Transformations

Further signal processing of the traces can be advantageous for SCA, particularly where countermeasures have been applied. Performing attacks in the frequency domain [78] or the use of principal component analysis (PCA) to transform the data prior to an attack [21] can allow an attack to succeed where otherwise it might have not have due to desynchronisation or excessive noise. Similarly where randomised clocking countermeasures have been utilised, methods have been suggested to reconstruct the original signal [114, 156, 216] through trace transformations and signal processing.

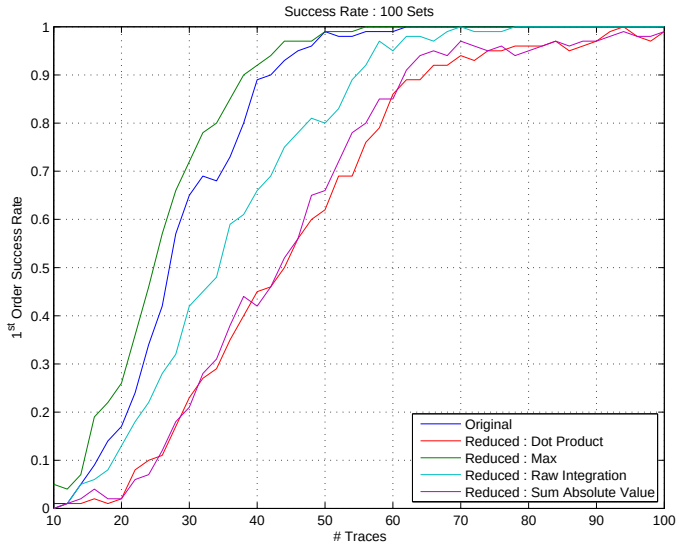


Figure 2.12: Trace reduction comparison.

2.11 Conclusion

In this background chapter, a broad overview of embedded systems and why they are vulnerable to SCAs has been presented, along with various categories of SCA and the ideas behind them. The trace acquisition setup as used in this thesis, along with the notation and success metrics which are utilised throughout have also been introduced.

A justification of the pre-processing steps applied to the ARM7 microprocessor AES traces in this work was presented through experimental analysis, and it was found that the following three pre-processing steps help increase the efficiency of a CPA attack against this dataset:

- Subtraction of the trace mean to remove the DC component.
- Applying a $N = 51$ point Blackman windowed FIR bandpass digital filter with the cutoff frequencies set to the clock frequency of $7.3739\,\text{MHz} \pm 500\,\text{kHz}$
- Reducing the traces to a single point per clock cycle by extracting the peak value from each cycle.

These pre-processing steps are applied to all traces used in this thesis acquired from an ARM7 microprocessor while sampling at $250\,\text{MSs}^{-1}$.

Chapter 3

Template Attacks

3.1 Introduction

TAs, first introduced by Chari *et al.* [42], take a different approach to key extraction than DPA, CPA or their variants. Rather than modelling the expected power consumption of a device based on some leakage model, TAs look to accurately model the power consumption using an identical device that an attacker has access to. Due to this strong, possibly all-encompassing, attacker model, a TA can be viewed as the strongest SCA possible attack in an information theoretic sense as stated in [42]. There are variants to this model, for example where an attacker does not have full control of the profiling device however knows that it contains a faulty RNG [3], or where an attacker simply performs a DPA attack to recover the key [169], but for the rest of this chapter we assume that the key is known for the profiling stage. The validity of profiling on one device, and attacking a second device can also be questioned, especially in smaller fabrication processes [188]. This question is further explored in §3.5, however in general it is assumed that different devices have sufficiently similar leakage to allow the mounting of a TA. As TAs represent the strongest possible attacker model, they are also particular useful for device manufacturers and vendors in evaluating side-channel resistance in a worst case scenario analysis testing as suggested in [206].

In this chapter, an overview of template attacks is first given in §3.2. This is followed up in §3.3 with an exploration of practical attack considerations and their effect on subsequent classification success rate, including a comparison of different classification methods in §3.3.6. These attacks are evaluated in a known plaintext attacker model with the set of

ARM7 microprocessor AES traces acquired as described in §2.4. The attacks work equally as well in a known ciphertext scenario, as is the case for many SCA, hence only known inputs are considered here. The effect of artificially adding noise to traces to simulate poor acquisition setups or noisier target devices is investigated in §3.3.7, and in §3.4 various implementations of AES on different platforms are compared to show how the underlying target device affects the attack. Finally in §3.5 multiple identical devices are used to empirically test the validity of the premise of TAs for a given target platform.

3.2 Template Attacks

A significant advantage of TAs is the ability to recover keys with a few or only a single power trace, overcoming implementations or countermeasures which restrict the number of traces an adversary can acquire. The flip side of this being the sometimes restrictive assumption that an identical device is available for profiling the power consumption. A TA is a two-stage attack, the computationally intensive profiling or training stage on the identical device, followed by the key recovery or classification stage of the target device. The templates generated during the profiling stage can subsequently be reused for fast key recovery from any similar device, potentially giving an attacker a large return for the initial workload. The first published two-stage attack was IPA by Fahn and Pearson [70], which made use of averaging to determine the location and leakage of individual key bits.

While it is assumed that an adversary can program, or knows the key of the profiling device, it is not assumed that the underlying source code is known. Knowledge of the source code would allow for a much stronger attack, most likely leading to a trivial key extraction. There are scenarios where an identical device with a known key is not required however, or the attacker assumptions are not as prohibitive as they might first seem. For example an attacker might own a cryptographic device such as a smart card with some unknown secret key. As the device is his own, he has unrestricted physical access to it and can spend significant time and effort recovering the key using traditional DPA. Subsequently he can build templates using this information for quick key extraction from similar devices that he does not have unfettered access to. This was among the ideas presented by Oswald an Paar in [169], however the authors were unable to transfer the success of the TA across different profiling and target devices (it was suggested by the authors this was possibly due to some desynchronising type countermeasure). Similarly, it was suggested in [93], that in an asymmetric key cryptosystem where a public verification function is available that uses the same sub-functions as a signature generation function,

this function could be used to generate templates under the assumption that the public key is known. No practical evaluation of this has been demonstrated to date however, and very likely there are many practical issues to overcome. It does highlight however that the TA model is not as restrictive or unrealistic as it might initially seem.

3.2.1 Template Training

The first stage of a TA is the training or profiling stage. A set of m power traces x, of length n are collected with their corresponding plaintext p and key s inputs. The target key space is given by $\mathcal{S}$ and contains $|\mathcal{S}|$ elements. The traces are assigned a label $y \in \mathcal{K}$ such that $y = \mathcal{F}(p, s)$. The function $\mathcal{F}$ is chosen such that it maps y to a secret s given p. This does not necessarily have to be a unique mapping, however unless it is bijective the classification stage will require more than one target trace. The unique values in the set $\mathcal{K}$ are denoted by, $o^{(1)}, o^{(2)}, \ldots, o^{(|\mathcal{K}|)}$. If the noise on the traces is additive and follows a Gaussian distribution, the traces can be assumed to be drawn from the multivariate normal distribution as given in Equation 3.1.

$$\mathcal{N}\left(x \mid \mu^{(i)}, \Sigma^{(i)}\right) = \frac{1}{\sqrt{(2\pi)^n \left|\Sigma^{(i)}\right|}} e^{-\frac{1}{2}\left(x-\mu^{(i)}\right)\left(\Sigma^{(i)}\right)^{-1}\left(x-\mu^{(i)}\right)^{\top}} \tag{3.1}$$

Where $\mu^{(i)}$ and $\Sigma^{(i)}$ represent the mean vector and noise covariance matrix of the class $o^{(i)}$, and $^{\top}$ represents the transpose operation. The training stage then consists of empirically estimating the mean vector $\hat{\mu}^{(i)}$ and noise covariance matrix $\hat{\Sigma}^{(i)}$ pair, for each instance of $o^{(i)}$, as defined in Equation 3.2 and Equation 3.3. Here $i \in \{1, \ldots, |\mathcal{K}|\}$, and $x^{(j,i)}$ represents the j^{th} acquisition of the class $o^{(i)}$, where $j \in \{1, \ldots, m^{(i)}\}$ and $m^{(i)}$ is the number of traces available for $o^{(i)}$ such that $\sum_{i=1}^{|\mathcal{K}|} m^{(i)} = m$.

$$\hat{\mu}^{(i)} = \frac{1}{m^{(i)}} \sum_{j=1}^{m^{(i)}} x^{(j,i)} \tag{3.2}$$

$$\hat{\Sigma}^{(i)} = \frac{1}{m^{(i)} - 1} \sum_{j=1}^{m^{(i)}} \left(x^{(j,i)} - \hat{\mu}^{(i)}\right)\left(x^{(j,i)} - \hat{\mu}^{(i)}\right)^{\top} \tag{3.3}$$

This estimated mean vector and noise covariance matrix pair $\left(\hat{\mu}^{(i)}, \hat{\Sigma}^{(i)}\right)$ is then the template associated with $o^{(i)}$ and completely specifies its noise distribution. An example operation that might be profiled could be the output of an intermediate function in a

cipher where a known input is combined with an unknown secret. The output of the first round AES *S-Box* is suitable in this regard as shown in [181], and used in many of the experiments here. The choice of target intermediate value can determine the success or otherwise of the attack, and is further investigated in §3.3.5. The template for each $o^{(i)}$ is constructed from a large number of traces, $x^{(j,i)}$ where $j \in \{1, \ldots, m^{(i)}\}$. However, the actual value of $m^{(i)}$ will vary widely from one device to another, being dependent on a host of different variables such as the size of the bus in the device under attack, the operation being performed, the power model *etc.*, which all contribute differing amounts of noise.

The traces used for the following experiments were acquired using randomly distributed inputs rather recording them on a *per class* basis. Where the mapping function $\mathcal{F}$ is bijective, the value of each $m^{(i)}$ will be similar, however if the Hamming weight power model is used it must be kept in mind that randomly selected traces will lead to unbalanced sets. Where all $m^{(i)}$ are large enough this is not a issue, as the sampling error between the $\mu^{(i)}$ and $\hat{\mu}^{(i)}$ will be small. As the noise is assumed normally distributed, the standard error of the mean is given by $\frac{\sigma}{\sqrt{m^{(i)}}}$, allowing a relative comparison of the accuracy of each template. A exploration of the effect of the total training set size m is examined in §3.3.4.

Where noise is an issue, it is also possible to perform TAs in the frequency domain as demonstrated in [186] by applying the discrete Fourier transform (DFT) to the traces. Trace compression methods such as described previously in §2.10.3 should not be used where the DFT is being performed, but depending on how many points in the DFT, generally the computational complexity of generating the templates in the frequency domain won't be an issue.

Regardless of whether a trace compression method is used or not, for the construction of templates further reduction is required to extract the features that the templates will be based on. One option is to sum the absolute differences between each of the mean traces and select the required number of highest points [186], which can be viewed as a variant of the DPA attack. In fact most DPA or CPA type attacks will allow some relevant features to be extracted as performing these attacks give the points in time where the target operation or value is processed, which is what is being modelling with the building of templates. Another approach is to use statistical pre-processing tools such as PCA [11] or linear discriminant analysis (LDA) [203] to transform the traces into a subspace of maximum variance. The attack is then performed on this set of transformed traces. A closer examination of various feature extraction and processing methods is given in 3.3.3.

3.2.2 Template Classification

To recover key information an attack, or testing, trace is required from the device under attack, preferably recorded under the same conditions. The trace must first be reduced in size and processed using the same steps that were used when generating the templates. For each possible class $o^{(i)} \in \mathcal{K}$, the likelihood of the trace corresponding to it is calculated using the multivariate Gaussian distribution from Equation 3.1, and plugging in the estimated values of $\left(\hat{\mu}^{(i)}, \hat{\Sigma}^{(i)}\right)$. The likelihood of $o^{(i)}$ can then be converted to a probability by applying Bayes' theorem as given in Equation 3.4.

$$\Pr\left(o^{(i)} \mid x\right) = \frac{p\left(x \mid o^{(i)}\right) \Pr\left(o^{(i)}\right)}{\sum_{j=1}^{|\mathcal{K}|} p\left(x \mid o^{(j)}\right) \Pr\left(o^{(j)}\right)} \tag{3.4}$$

Here $\Pr\left(o^{(i)}\right)$ is the prior probability of the class occurring, and $p\left(x \mid o^{(i)}\right)$ is given by $\mathcal{N}\left(x \mid \mu^{(i)}, \Sigma^{(i)}\right)$. Applying the maximum likelihood principal, the estimated class is then given by Equation 3.5. If all operations are equiprobable then the application of Bayes' theorem is unnecessary as it simply scales the likelihood values, and Equation 3.5 can be applied directly to the likelihood values.

$$\hat{s} = \mathcal{F}\left(\arg\max_{o} \Pr\left(o^{(i)} \mid x\right), p\right)^{-1} \tag{3.5}$$

The success rate of the attack is increased if a set of power traces for a constant secret key is available such that $m > 1$. In this scenario, Bayes' theorem can be applied iteratively if the power traces are statistically independent thereby increasing the power of the attack as given in Equation 3.6 [173]. Note this is equivalent to Equation 3.4 when $m = 1$. Once again the maximum likelihood is used to return the estimated key $\hat{s}$.

$$\Pr\left(o^{(i)} \mid x\right) = \frac{\left(\prod_{k=1}^{m} p\left(x^{(k)} \mid o^{(i)}\right)\right) \cdot \Pr\left(o^{(i)}\right)}{\sum_{j=1}^{|\mathcal{K}|} \left(\left(\prod_{k=1}^{m} p\left(x^{(k)} \mid o^{(j)}\right)\right) \cdot \Pr\left(o^{(j)}\right)\right)} \tag{3.6}$$

3.2.3 Template Attack on AES

As an example TA, an unprotected ARM7 microprocessor AES implementation is attacked, with the traces acquired as described in §2.4. To build the templates, 10 k power traces of the first round of the encryption algorithm are recorded with different uniformly random plaintexts and keys for each trace. The choice of 10 k is arbitrary, as are all parameter selections, and simply provides an initial baseline comparison. The testing traces are

recorded separately on the same device with uniformly random plaintexts and a constant key. Note that both the plaintext and key need not be random to generate the templates, just the target intermediate value. Pre-processing as described in §2.10 is applied to traces prior to template training. The target operation of the attack that the templates are built for is the output of the first *S-Box* of the first round. Any key dependent intermediate value can be chosen, however the *S-Box* is particularly suitable as shown in §3.3.5.

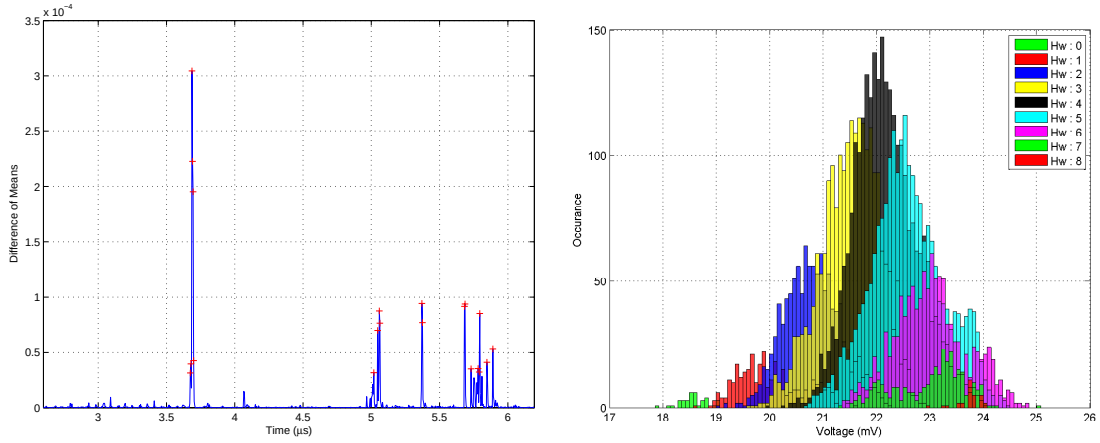
Template Training

Assuming the ARM7 microprocessor follows a Hamming weight power model, the output byte of the *S-Box* will give $|\mathcal{K}| = 9$ possible classes. To select the points of interest from the recorded traces, the sum of squares of differences (SOSD) method is used, as suggested by [42, 186]. As shown in Equation 3.7, this consists of summing the squares of the differences between each pair of mean traces for each class. The points in this difference trace are the points that have the highest variance when grouped according to the target class, which are the points that have the largest power component dependent on the target leakage operation. The $\tilde{n} \leq n$ features with a significant peak are retained for the building of templates. This is not the only way to select the relevant features to build templates with, and various methods are explored in §3.3.3.

$$\Delta = \sum_{i=1}^{|\mathcal{K}-1|} \sum_{j=i+1}^{|\mathcal{K}|} \left(\hat{\mu}^{(i)} - \hat{\mu}^{(j)} \right)^2 \tag{3.7}$$

The selected points are highlighted by the red crosses in Figure 3.1(a). For the example here we choose all the distinct peaks, which leads us to to select $\tilde{n} = 20$ points. The effect of selection too many or too few points is also further explored in §3.3.3. The initial peak is the point in time where the output of the *S-Box* itself is processed, while the subsequent smaller peaks are due to the use of the target value in the *MixColumns* operation. This corresponds to the correlation peaks as seen in §2.7.2 as broadly the same leakage is being utilised. Figure 3.1(b) shows the distribution of the all the trace points x_j located at the largest peak coloured according to their respective Hamming weights. It is this separation of Hamming weight classes that allows templates to be constructed.

To evaluate the templates, the traces used to generate the templates are themselves classified first to calculate the *training error*. These traces are classified by taking the maximum probability returned by Equation 3.4, as due to the Hamming weight distribution, each template does not have an equal prior probability. The 10 k training traces are classified



(a) Feature selection.

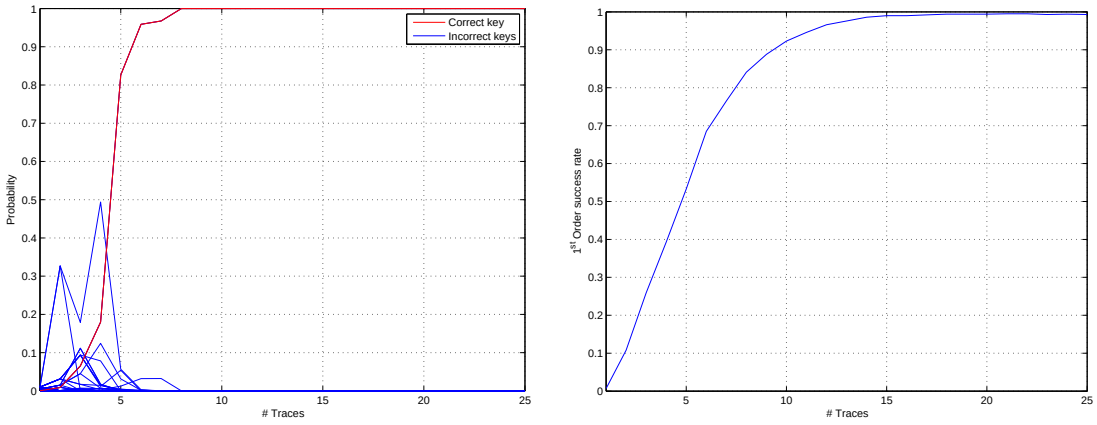
(b) Hamming weight distribution.

Figure 3.1: Template generation.

with an error rate of 0.318, which can be viewed as lowest error rate achievable with these particular templates, with the error of the testing traces expected to be higher.

Template Testing

To test the templates 25 k traces acquired from the same device and using an identical setup were acquired, this time with a uniformly random plaintext and constant key as previously mentioned. The traces were processed in the same manner as before, and the testing error rate on this unseen data was 0.3461, *i.e.* approximately a third of the traces were assigned the incorrect class $o^{(i)}$, each of which represented a different Hamming weight. The recovery of the correct Hamming weight of the *S-Box* output does not in itself return the key, it provides the attacker with a subset of keys. Therefore an amplified TA is required to recover the secret s [186]. The set of attack traces is randomly split up into 1 k subsets of 25 traces, and Equation 3.6 is applied to see if the key byte can be recovered from the 25 traces. All 256 byte values are tested and combined with the plaintext to calculate a hypothesis for Hamming weight of the *S-Box* output. The relevant probabilities returned by the templates are selected, with Figure 3.2(a) showing the result of an example attack with the correct key plotted in red. This attack is repeated for all 1 k subsets independently, with the success rate as defined in §2.8 plotted in Figure 3.2(b). It is clear from this plot that 25 traces will allow you to recover the key byte with a probability close to one for this particular setup.



(a) Amplified template attack.

(b) Success rate : $1k$ sets.

Figure 3.2: Template testing.

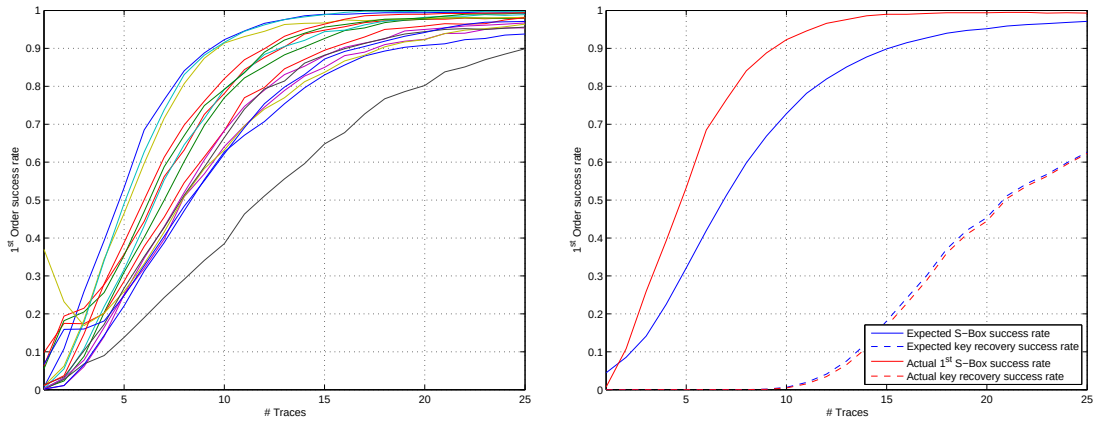
3.2.4 Full Key Recovery vs Partial Key Recovery

It is worth mentioning that the success rate given in Figure 3.2(b) is only for a single byte of the key. Repeating the attack 16 times against all the *S-Boxes* to recover the entire key, it is clear from Figure 3.3(a) that each *S-Box* does not leak information equally. To calculate the expected success rate for an arbitrary key byte from this setup, the average of all 16 success rates is taken as shown by the solid blue line in Figure 3.3(b). For comparison the success rate from the first *S-Box* as used previously is given as the solid red line. The overall expected success rate for the entire key can then be calculated by raising the expected byte success rate to the number of bytes in the key, *i.e.* in this case 16. The actual success rate is given as reference also by computing the product across each of the individual key bytes and is quite similar as expected, with the difference due to numerical round-off errors. This shows that the results of the key recovery for a single byte cannot be arbitrary extrapolated to the key as a whole.

3.3 Practical Attack Considerations

As highlighted by [186], there are many practical issues to be considered when performing a TA. Attack choices depend on both the quality and quantity of your data, and affect both the error rate and computational complexity. A comparative analysis of various parameters is now given using the output of the first *S-Box* of the first round as the target value which allows the recovery of a single key byte, unless otherwise stated.

When comparing the effect of various parameters on the classification performance, it is



(a) Individual *S-Box* success rates.

(b) Expected success rates.

Figure 3.3: Full key recovery.

often more convenient to use the error rate given a single trace rather than the success rate of an amplified attack with multiple traces. Hence, where the secret value can be recovered in a single trace, the error rate is used as the comparison metric.

3.3.1 Power Model

Building templates is not restricted to the Hamming weight of the target value alone. Some common models that can be used and their properties are outlined below.

Single-bit model: The single-bit model targets the same leakage as the original DPA paper [126]. That is, templates are built for a single bit of the target value.

Multi-bit model: This expands on the single-bit model by building templates for each individual bit of the target value using the same set of power traces. These bits are subsequently combined using the probability of their occurrence to allow byte recovery.

Hamming weight model: This model, as used in the previous experiments, follows what is thought to be an good model of the power consumption for the device at hand. It assumes that the power consumption is proportional to the number of bits set to 1 in the target value. This is a valid model where the bus of a microprocessor is a constant value, as can often be the case due to pre-charging, or where the same op-code is called prior to the data being loaded on the bus.

Hamming distance model: Not applicable here, but this model assumes that the power consumption is proportional to the number of bit-flips between the current and

	Single-bit	Multi-bit	Hamming weight	Identity
Training	0.1118	0.1447	0.3180	0.0020
Testing	0.1165	0.1526	0.3477	0.2764

Table 3.1: Power model error rates.

previous value. This power model is generally more suitable for hardware based attacks such as FPGAs or ASICs as demonstrated in [1].

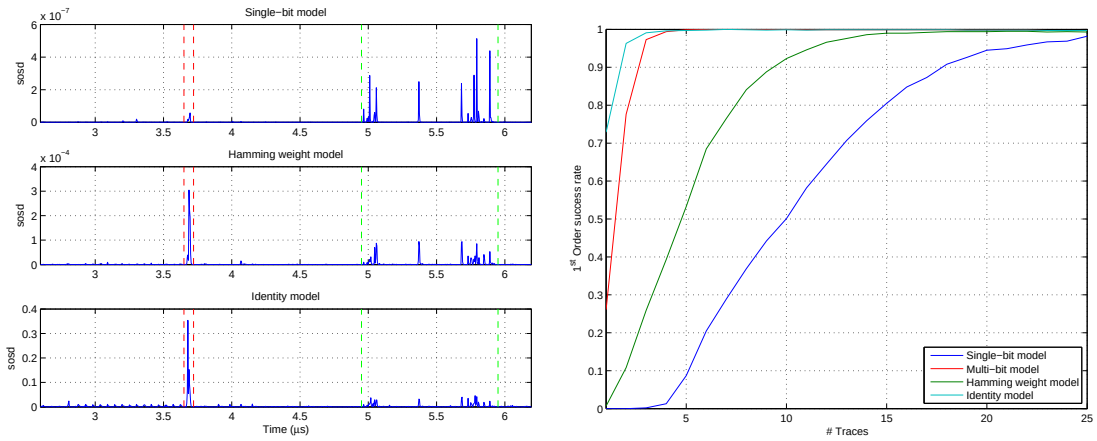
Identity model: Here, the templates are built for the target intermediate value directly. This incurs a greater computational complexity as more templates need to be trained and classified, *i.e.* in the case of a byte-sized target value, 256 as opposed to 9 templates when using the Hamming weight model. Therefore additional training traces are also required to accurately model the extra templates.

The experiment was rerun using 10 k training traces once again, with 20 features retained for each model using the SOSD method, and 1 k testing traces were used to calculate the test error as given in Table 3.1. As feature selection is calculated with the mean $\hat{\mu}^{(i)}$ of each class $o^{(i)}$, these 20 features are not necessarily the same across models. These values represent the model classification error rather than the byte recovery error. Examining Table 3.1, some interesting points appear. The first is that in all cases, the testing error is worse than the training error which is as expected as the model is fit to the training set. However, it is worth noting that the difference between the errors for the single-bit model is only 0.0063, while for the identity model it is 0.2744. This is due to the fact that on average $10\,\text{k} \div 2 = 5\,\text{k}$ traces are available to generate each single-bit template, but only $10\,\text{k} \div 256 \approx 40$ are available to generate each identity template. Also a random guess in single-bit case will give an expected error of $\frac{1}{2}$, while for the identity case it is $\frac{1}{256}$. Note that the multi-bit errors are greater than the single-bit, with this error value taken as the mean for all 8-bits. This difference is due to the fact that not all bits will leak information equally, similar to the byte leakage as previously shown. Most interesting however, is that the Hamming weight error rates are much higher than that of the identity model, even though the correct key cannot be determined from a single correct Hamming weight classification, unlike the identity model case.

In Figure 3.4(a), the difference points of the trace that are used to generate the templates are highlighted. For the bit models, the section of the trace around the *MixColumns* operation from $5 \rightarrow 6\,\mu\text{s}$ as bracketed by the green lines, seems to have the most data dependency with only a small bit of the leakage due to the *S-Box* output itself, which

occurs about 3.7 µs as highlighted in red. For the Hamming weight model, leakage from both operations is utilised, and for the identity model the strongest leakage comes from the *S-Box* operation which contains a double peak unlike the other models. This double peak is actually due to the data leakage on both the input and output of the *S-Box*. As it is the only model where the inputs and outputs map uniquely, this second peak only occurs for this model. The multi-bit case is not plotted, as it is 8 individual bit models which will all be similar to the single-bit plot in Figure 3.4(a).

The overall key byte recovery success rate for the models is given in Figure 3.4(b). As can be seen, the most accurate model for returning a key byte is the identity model due to the extra information obtained by modelling the second peak, followed by the multi-bit model. The multi-bit model is considerably more successful than the single-bit model as much greater utilisation is made of the available leakage (similar to the difference between single-bit DPA and enhanced DPA). Hence, unless otherwise specified, the identity model is used in future experiments. In the case of the attacks on the AES *S-Box*, each class $o^{(i)}$ represents a different byte value and $|\mathcal{K}| = 256$.



(a) Feature selection.

(b) Success rate : $1k$ sets

Figure 3.4: Power model.

3.3.2 Data Normalisation

Where data is badly conditioned, it can be worthwhile applying a data normalisation technique prior to feature selection. This prevents a single dominant feature with a large range from having a disproportionate effect on the subsequent classification. Some normalisation techniques are as follows:

Norm: This method makes no underlying assumption about the distribution of the data.

	Original	Norm	Range	Scale	Z-Score
Training	0.0021	0.0057	0.0017	0.0017	0.0016
Testing	0.2760	0.3790	0.2250	0.2250	0.2180

Table 3.2: Data normalisation error rates.

The mean of each data column x_j is subtracted from itself, and is then divided by its norm such that the features have roughly the same scale.

Range: The principal of this method is similar to that of the norm, however instead of dividing by the norm of the column vector, the range of the data is used instead.

Scale: This method assumes the data has a uniform distribution with the different points having distinct ranges. Scaling the data linearly transforms it such that it is in the range $[0, 1]$

Z-Score: Assuming that the points of the trace are normally distributed with each point in time having a distinct mean and variance, calculating the z-score involves subtracting the mean and dividing by the standard deviation so each point has zero mean and unit standard deviation.

Other non-linear normalisation techniques such as taking the percentiles of the data might be useful in certain scenarios, however as an underlying assumption of template attacks is that the noise is Gaussian these are not considered. While the normalisation parameters for the training set are estimated from the set itself, these parameters are then reused to scale the testing set accordingly. This is different to the approach by Montminy *et al.* [154], where the testing traces are normalised using parameters estimated from the test set itself. As the objective here is to minimise the error given a single attack trace, it cannot be assumed that multiple traces are available to accurately estimate the normalisation parameters. Using templates built with 10 k traces and 20 features using the identity model, the error rates for the different normalisation methods are given in Table 3.2. As taking the z-score of the data returns the lowest empirical error, this is reused through the thesis unless otherwise specified. Normalisation has an added advantage of leading to less numerical errors when computing the templates, as well faster convergence when training the machine learning techniques in Chapter 6. Note also that the slight difference for the original error rate compared with Table 3.1 is due to the random selection of traces.

3.3.3 Feature Selection

Selecting the points to use in building templates is a non-trivial task. Too many points and the templates won't be an accurate representation of the target value (*i.e.* the templates will mostly be modelling random noise), and too few points will give poor classification results. Different methods for ranking the points according to interesting leakage are now examined:

Sum of squares of difference of means (sosd): This is the method that has been used to date and is given in Equation 3.7.

Sum of squares of t-difference of means (sost): This method was suggested in [81] when it was noticed that templates performed poorly when the training set size was restricted. Student's t-test is a statistical method for distinguishing two sets taking into account the variance of the sets as well as the distance between the means. This was also suggested as an alternative to the classical DOM attack as outlined in §2.7.3. The points to build the templates are chosen as the largest returned by Equation 3.8.

$$\Delta = \sum_{i=1}^{|\mathcal{K}-1|} \sum_{j=i+1}^{|\mathcal{K}|} \left(\frac{\hat{\mu}^{(i)} - \hat{\mu}^{(j)}}{\sqrt{\frac{\left(\hat{\sigma}^{(i)}\right)^2}{m^{(i)}} - \frac{\left(\hat{\sigma}^{(j)}\right)^2}{m^{(j)}}}} \right)^2 \tag{3.8}$$

Correlation: This is a logical extension to make for feature extraction given that both previous methods can be used for classical DPA. As calculating the correlation with the known intermediate value will show the points in time where that data is processed, these points can then be chosen to build the templates. Pearson's linear correlation is used here, but for non-parametric data Spearman's rank correlation can be used. The equation for calculating the linear correlation coefficient has been given previously in Equation 2.6.

Principal component analysis (PCA): This method of feature selection was introduced for template attacks by Archambeau *et al.* in [11]. PCA is a data dimensionality reduction technique which projects the traces into a subspace where the principal directions are orthogonal to each other and ranked according to variability. Only directions with a high relative rank are retained for template construction. It is a linear transformation and computes the subspace directions using linear combinations of the original data space [63, Ch. 3]. A projection matrix P calculated

according to Equation 3.9 [11], can be used to project both the training and test traces into the subspace and all template calculations performed there. To calculate P, the covariance matrix Σ of the centred means for each class is first calculated. Taking the eigendecomposition of Σ to get the eigenvectors U and eigenvalues D, then allows the calculation of P. Note that D is a diagonal matrix so can be inverted element-wise.

$$\begin{aligned} \bar{\mu} &= \frac{1}{|\mathcal{K}|} \sum_{i=1}^{|\mathcal{K}|} \hat{\mu}^{(i)} \\ \Sigma &= \frac{1}{|\mathcal{K}|} \sum_{i=1}^{|\mathcal{K}|} \left(\hat{\mu}^{(i)} - \bar{\mu}\right) \left(\hat{\mu}^{(i)} - \bar{\mu}\right)^{\top} \\ \Sigma U &= DU \\ P &= \frac{1}{\sqrt{|\mathcal{K}|}} \hat{\mu}\, U\, D^{-\frac{1}{2}} \end{aligned} \tag{3.9}$$

The eigenvalues D can be used to select the directions representing a "large" variance. Alternatively the cumulative sum of the eigenvalues can be calculated, and the number of components to be retained determined by how many are required to reach a given percentage of the variance. As an example, the three eigenvectors corresponding to the three largest eigenvalues are given in Figure 3.5(a), for when PCA is applied to the AES training set. As the transformation is computed via matrix multiplication $\tilde{x} = x\,P$, the first point of a projected trace is the weighted sum of the original trace according to the weights of the first column of P, the second point the weighted sum of the second transformation column *etc.*.

Fisher's linear discriminant: This method also transforms the data to a reduced subspace however this time the inter class to intra class ratio is maximised [203]. Where PCA seeks the directions that best represent the data, Fisher's linear discriminant seeks the directions that allow for optimal separation of the data [63]. To calculate the projection matrix, an inter-class scatter matrix Σ_B and an intra-class scatter matrix Σ_W are first calculated. The projection matrix can then be calculated with an eigendecomposition as shown in Equation 3.10.

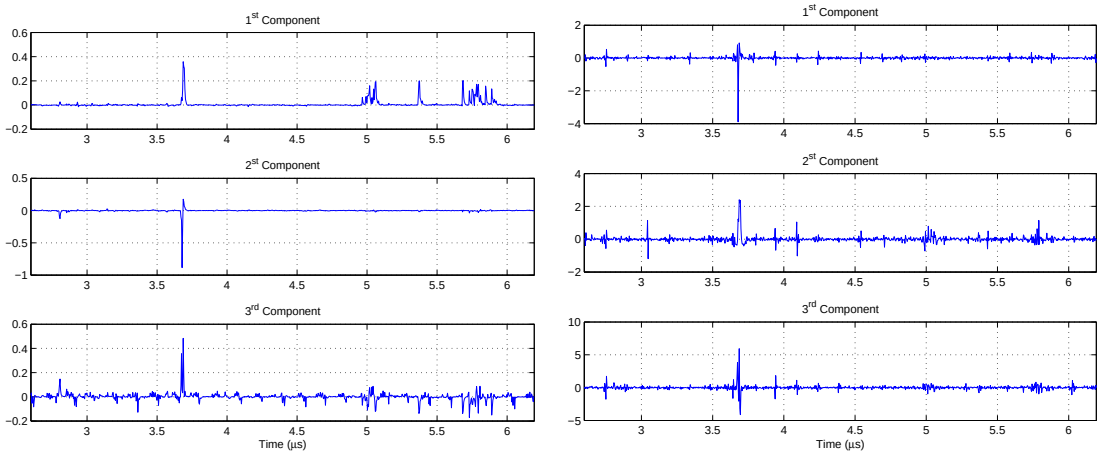
$$\Sigma_B = \sum_{i=1}^{|\mathcal{K}|} m^{(i)} \left(\hat{\mu}^{(i)} - \bar{\mu}\right) \left(\hat{\mu}^{(i)} - \bar{\mu}\right)^\top$$

$$\Sigma_W = \sum_{i=1}^{|\mathcal{K}|} \sum_{j=1}^{m^{(i)}} \left(x^{(j,i)} - \hat{\mu}^{(i)}\right) \left(x^{(j,i)} - \hat{\mu}^{(i)}\right)^\top$$

$$\left(\Sigma_B^{\frac{1}{2}} \Sigma_W^{-1} \Sigma_B^{\frac{1}{2}}\right) U = DU$$

$$P = \Sigma_B^{\frac{1}{2}} U \tag{3.10}$$

For a full description of Fishers's linear discriminant in the context of SCA including numerical optimisation tricks see [45, 203]. Once again after the transformation matrix P is calculated, both training and testing traces are projected to the subspace and the attack performed there. The first three columns of the transformation matrix P are given in Figure 3.5(b). It can be seen that, similar to PCA, the features which are assigned the greatest weights are those around the time 3.7 µs which is where the *S-Box* operation occurs. Note also that the maximum number of features that can be returned is $|\mathcal{K}| - 1$.



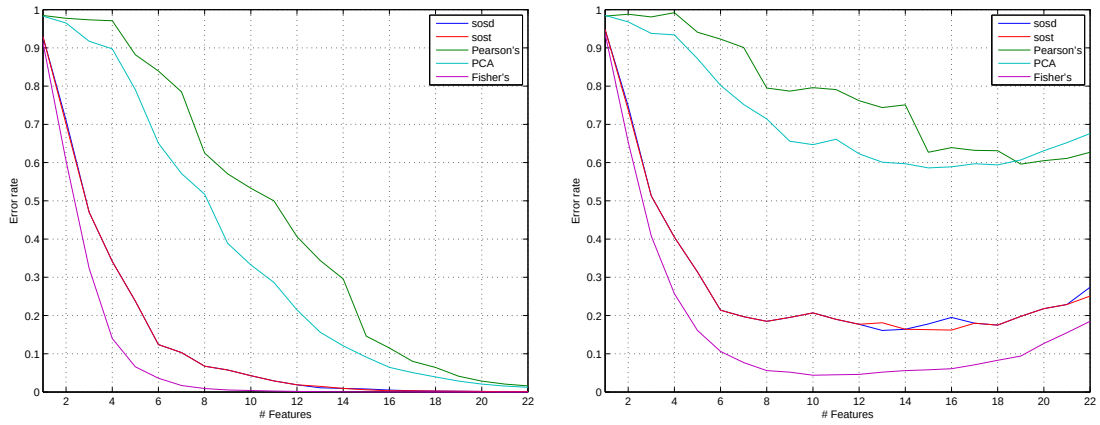
(a) PCA transformation vectors.

(b) LDA transformation vector.

Figure 3.5: Transformation vectors.

Other methods of feature selection have been proposed such as *normalised inter-class variance* (NICV) [28] or the use of an F-test [45], however these are not considered here. Using these methods as outlined above, the learning curves for the template training and testing error rates are given in Figure 3.6(a) and Figure 3.6(b) respectively, with the error plotted as a function of the number of retained features. These templates are built

using the identity model and 10 k normalised training traces, while 1 k traces are used for testing. As can be seen, the training error decreases with the number of features retained until an error rate close to zero is achieved for all feature selection methods. Conversely, while the testing error in Figure 3.6(b) initially decreases as features are added, it starts to increase towards the end. This is due to an over-fitting of the templates on the training set, hence the model doesn't generalise well to unseen data. For the testing error which is what we are interested in minimising, the use of Fisher's linear discriminant for dimensionality reduction gives clear classification performance advantages, hence is used for feature selection for the following experiments.



(a) Training error. (b) Testing error.

Figure 3.6: Effect of feature selection.

3.3.4 Training Set Size

The size of the required training set is an important parameter in the context of a TA, especially where an attacker might be somehow bounded in the acquisition stage of the profiling traces. An interesting and relevant question is how few traces are required for a successful TA attack, this issue was explored in [68, 81, 205], or how low an error can be achieved given unlimited traces. Figure 3.7 shows the learning curve for the error rate as a function of the training set size. The training errors are given by the dashed lines, with the test error given by the solid lines.

For the identity model plotted in red, 12 features are retained using Fisher's linear discriminant and it can be seen that increasing the training set size reduces the testing error (as well as slightly increasing the training error) up to about 20 k samples, after which the extra samples have little effect. Hence the addition of extra training samples is un-

likely to further decrease the error. Where the training sample size is less than $\approx 7\,\text{k}$, the noise covariance matrix $\Sigma^{(i)}$ for the $|\mathcal{K}| = 256$ classes for each byte cannot be accurately modelled as only $\lesssim \frac{7\,\text{k}}{256} \approx 27$ samples per class are available. To evaluate classification performance with smaller training set sizes, the multi-bit model using the SOST method of feature extraction was used as only two classes need to be estimated giving many more samples per class. The addition of extra training samples has no effect on the multi-bit model, however even with 1 k traces there are already ≈ 500 traces to model the leakage of each bit. However the relatively high error rate of ≈ 0.22 means an amplified TA will most likely be required. Note that the normalisation and feature selection parameters were recalculated on the number of training samples under consideration each time.

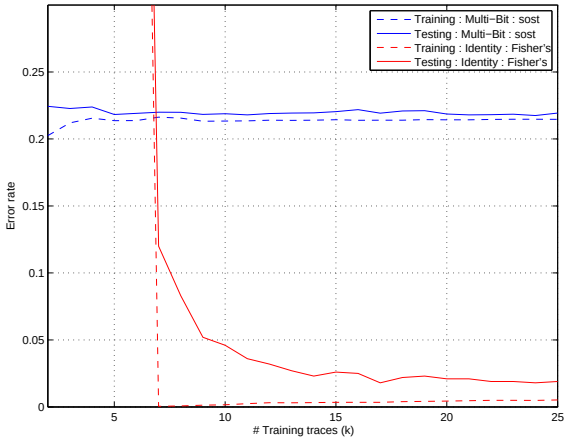


Figure 3.7: Effect of training set size.

3.3.5 Target Intermediate Value

The chosen target intermediate value or operation has a large bearing on the success of an attack. It is shown in [181] that the more resistant an *S-Box* is to linear cryptanalysis, the more susceptible it is to CPA attacks. Generally, as the primary function of the *S-Box* is to introduce non-linearity in the case of a block cipher, it can be a suitable candidate function to attack in non-profiling attacks. A study of DPA against Boolean and arithmetic operations is available in [133]. In the case of profiling attacks, such as TA, the *S-Box* too is a suitable candidate function as shown in Figure 3.8. Here templates were built for the first output byte of all AES functions up to round two[1]. The section of the power traces consisting of communications between the PC and ARM7 microprocessor board was not

[1]Note that recovery of the plaintext, *MixColumns* or round two input bytes will not allow key recovery, the classification is only to illustrate the effect of selecting various intermediate values.

considered for the attack, hence the classification of the plaintext and key reflect leakage during the operation of the algorithm only. The training set size is 25 k samples, hence only the identity model is used, and 12 features retained using Fisher's linear discriminant.

When using the identity model, there is a bijective relationship between the round one input and the *S-Box* bytes, hence building a template for one is equivalent to building a template for the other (the error curve of the round one input is hidden by the plot of *S-Box*). The relative classification success of the *S-Box* operation compared to the others is not solely due to the reasons outlined in [181]. The plaintext and *MixColumns* operations perform poorly as each value is only used once in the algorithm. The key and sub-key perform slightly better as, the key is stored in memory for reuse across different plaintexts hence needs to be loaded prior to the *AddRoundKey* function, while the sub-key needs to be computed prior to use (only the key rather than the expanded keys are stored in memory). The *S-Box* output is subsequently reused four times in the *MixColumns* operation, as well as utilising what leakage is relevant for the round input. Therefore more data-dependent points where these bytes are processed are available.

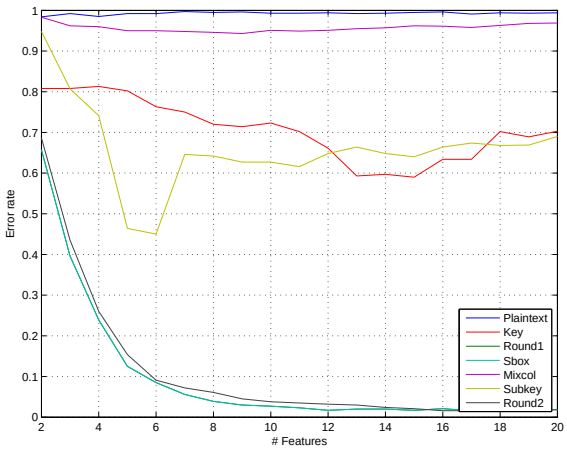
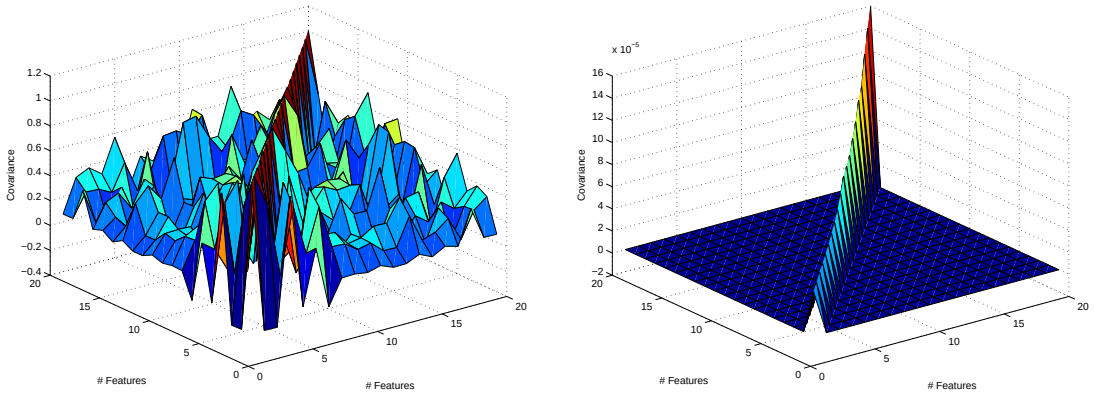


Figure 3.8: Error rates for various intermediate values.

3.3.6 Classification Methods

It is suggested in [145, Ch. 5] that in order to avoid numerical errors when inverting the covariance matrix, as well as reducing computation time, that *reduced templates* are used. A reduced template attack is where only the diagonal of the covariance matrix is used (*i.e.* the variance of each feature), with all off-diagonal matrix elements set to zero, which is essentially the Naïve Bayes learning algorithm. Inverting the matrix then simply consists of inverting each point on the diagonal individually, hence it also scales

well with an increasing number of features. The underlying assumption behind this is that the various leakages used to generate the templates are mostly independent, and the information leakage due to taking the covariances into account is lost due to numerical errors in the matrix inversion. To verify this assumption, the covariance matrices of the *entire* 25 k training set are given in Figure 3.9. The 20 features in Figure 3.9(a) are chosen through SOST, while in Figure 3.9(b) they are chosen with Fisher's linear discriminant. In both cases the features are ordered from the from the most "informative" to the least.



(a) Feature selection : SOST (b) Feature selection : Fisher's linear discriminant

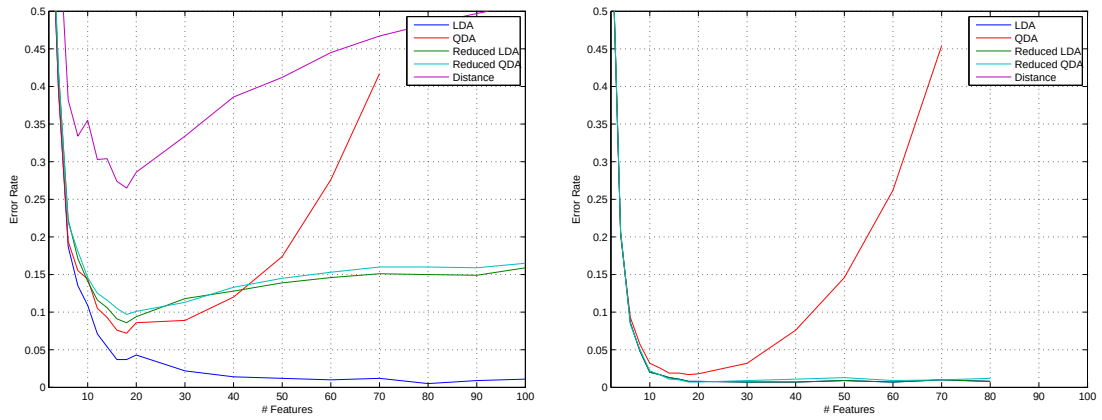
Figure 3.9: Training data covariance matrices.

In the SOST case it is clear that there is a significant covariance between the difference features hence the use of reduced templates will be sub-optimal. The diagonal (*i.e.* the variance of each feature) is 1 due to the normalisation of the data. Where the features are selected with Fisher's linear discriminant however, the off diagonal elements are all zero as the projection of the data to the subspace has de-correlated the various features. Hence in this case the use of reduced templates will have little or no effect on the classification performance. Note also that the variance of the transformed features increases as they become less "informative", as they contain a greater noise component.

A TA such that each class is modelled as a multivariate Gaussian with a mean vector and covariance matrix, is known as quadratic discriminant analysis (QDA) in statistical learning as detailed in [95, Ch. 4]. Another option, also detailed in [95, Ch. 4] is to assume that all classes have the same covariance matrix which is known as LDA, which was also suggested as a classifier for SCA in [45]. This fits the model of SCA as the covariance matrix represents the noise on the traces, and there is no reason to believe that the different classes representing data values would have different noise components.

The error rates for LDA, QDA, reduced LDA and QDA, and the Euclidean distance (this

assumes the covariance matrix carries no information and it is set to the identity matrix) are given for SOST feature selection in Figure 3.10(a), and for Fisher's linear discriminant in Figure 3.10(b). The Euclidean distance performs the worst in both cases, for the case of Fisher's linear discriminant it performs no better than a random guess and is not shown on the plot. While QDA initially performs relatively well, as the features which don't contribute as strongly to the classification performance are added, then numerical errors occur as not enough traces are available to model the noise for those less informative points. LDA performs strongly in both cases, with the reduced methods also performing equally as well in Figure 3.10(b) where the data has been transformed such that the features are strongly uncorrelated.



(a) Feature selection : SOST (b) Feature selection : Fisher's linear discriminant

Figure 3.10: Classification methods.

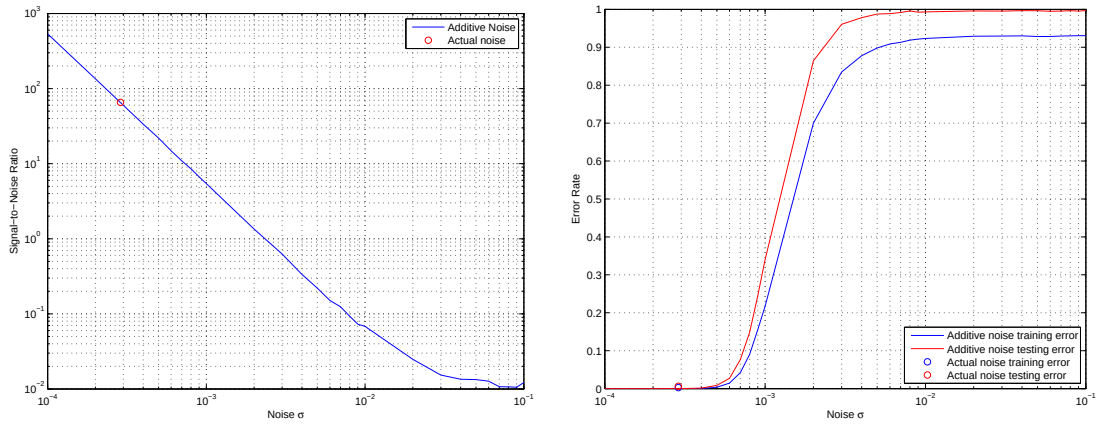
An interesting point is that the use of LDA as a classifier with a straightforward feature selection method such as SOST, performs equally as well as feature selection with Fisher's linear discriminant combined with any "reduced" classifier. LDA and Fisher's linear discriminant are closely related however so this is not too surprising (the terms are often used interchangeably), with LDA seeking to assign a label to data and Fisher's linear discriminant looking the assign a continuous variable which allows efficient discrimination of the data. Regardless of feature selection, in general LDA is a more efficient and robust classifier than QDA for SCA and is used for future work unless otherwise specified.

3.3.7 Noise Effect

An important aspect of any SCA is how it performs in the presence of increased noise. Additive noise is often used in conjunction with other countermeasures as a simple yet

effective method to improve the resistance of a cryptographic implementation against SCA (note that on its own, an adversary can overcome the addition of noise by simply utilising more traces). Additionally different target devices will have greatly varying noise characteristics, and acquisition setups or any on-chip parallel processing can an all contribute to the difficulty of performing an attack. As the noise is assumed Gaussian (as previously shown in Figure 2.4), averaging multiple traces with the same input can be used to reduce it. However, in reality this is equivalent to requiring more traces to perform an attack hence is not used here.

As previously mentioned in §2.5, the SNR can be used to measure the relative strength of the signal component compared to the noise component of a trace. Using a training set of 25 k traces each trace is replaced by its equivalent empirical mean, and increasing additive Gaussian noise is added to evaluate to SNR calculated as shown in Figure 3.11(a). The SNR of the actual traces is also given for reference by the red circle. The addition of Gaussian noise to mean-free traces has been used previously in papers such as [188] to examine the effect of increasing noise on attacks. In Figure 3.11(b), TAs are performed on the traces with artificially added noise using a training set of 25 k traces, the identity model, Fisher's linear discriminant to select 20 features, and LDA to classify the data. It can be seen that as the noise increases, eventually the testing error rate goes to 1 and a TA is no longer possible. The error rate for the actual noise is also marked for reference, also by the circles.



(a) Maximum SNR as additive Gaussian noise is increased.

(b) Template attack as additive Gaussian noise is increased.

Figure 3.11: Effect of noise.

	F_{clk}	F_s	Filter	Reduction
PIC	4 MHz	250 MSs^{-1}	Low-pass 6 MHz	Max point per clock
8051	11 MHz	250 MSs^{-1}	Low-pass 16 MHz	Max point per clock
ARM7	7.37 MHz	250 MSs^{-1}	Band-pass 6.87 $\rightarrow$ 7.87 MHz	Max point per clock

Table 3.3: Power trace acquisition & pre-processing parameters.

Training traces (m)	9 **k**
Testing traces	1 **k**
Data normalisation	*z-score*
Model	*single-bit, Hamming weight, identity*
Feature extraction	SOST
Number of features	2 $\rightarrow$ 100
Target Byte	First byte of *S-Box*

Table 3.4: Template training parameters.

3.4 Platform Effect

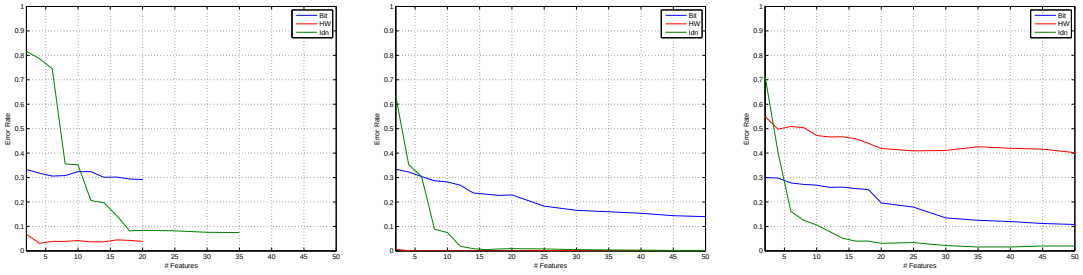
Like all SCA, TAs are platform dependent. Following on from the examination of noise, three separate software platforms are now attacked to highlight differences due the to underlying target platform, and how those differences will affect subsequent analysis. The software AES target platforms are a low-cost PIC smartcard, a low-cost Atmel 8051 microcontroller[2], and the ARM7 microprocessor chip used previously for reference. While the template training parameters are kept constant between the platforms, difference acquisition setups were not and are given in Table 3.3. The different filters parameters were selected through empirical evaluation on a per device basis.

In all cases, the parameters from Table 3.4 are used to build the templates. Restricting the number of training traces to 9 k was required for the comparison, as only 10 k traces in total were available for two of the target platforms, and the SOST method of feature selection was chosen over Fisher's linear discriminant as the number of features for the *Bit* and *Hamming weight* models would have been restricted to 1 and 8 respectively otherwise.

It is clear in Figure 3.12 that the identity model performs well on all platforms where enough features are taken into consideration. As mentioned previously in §3.3.5, the extra information contained at the *S-Box* input helps in this regard. Note also that

[2]These power traces were supplied by Dr. Elisabeth Oswald of the University of Bristol while conducting the research in [92]. They are a subset of the traces used extensively in [145]

further traces for the *Bit* and *Hamming weight* models are required to recover the key byte. It is interesting to note however, that compared to the PIC smartcard and 8051 microcontroller platforms, the Hamming weight model clearly does not accurately reflect the power consumption of the ARM7 microprocessor (This is also borne out in a CPA attack which has a lower absolute correlation peak for the correct key guess compared to the other platforms). Hence, while broad recommendations can be made with regards to how to approach an TA for a given device, analysis is required to find the optimum parameters.



(a) PIC smartcard (b) 8051 microcontroller (c) ARM7 microprocessor

Figure 3.12: Testing error on various software platforms.

3.5 Model Validity

A pertinent question when examining TAs, is how valid is the assumption that templates built using one device are transferable to extract keys from another device. Many publications to date assume that the leakage from the target device is identical to that of the training device (*i.e.* the same device is used to acquire both training and testing traces). In [188], the authors use the concept of perceived information (PI) to account for discrepancies between the template models and actual target device leakage. The targets of their experimentation are nano-scale devices however, and inter-chip power variation increases as the chip fabrication process decreases. In [69], the authors examine the *portability* of templates by comparing acquisitions taken from the same chip at different times (they compare two sets of traces, taken four years apart with different acquisition setups) and supply voltage. Rather than the PI approach, they use attack metrics to justify their work, which is also the approach used here. Also, in [154] PIC smartcards are used to examine the validity of electromagnetic emanation based templates as it has been shown that electromagnetic emissions can be used as a form of physically unclonable function (PUF) [51], hence the leakage must have some device dependency. In [139], three micro-

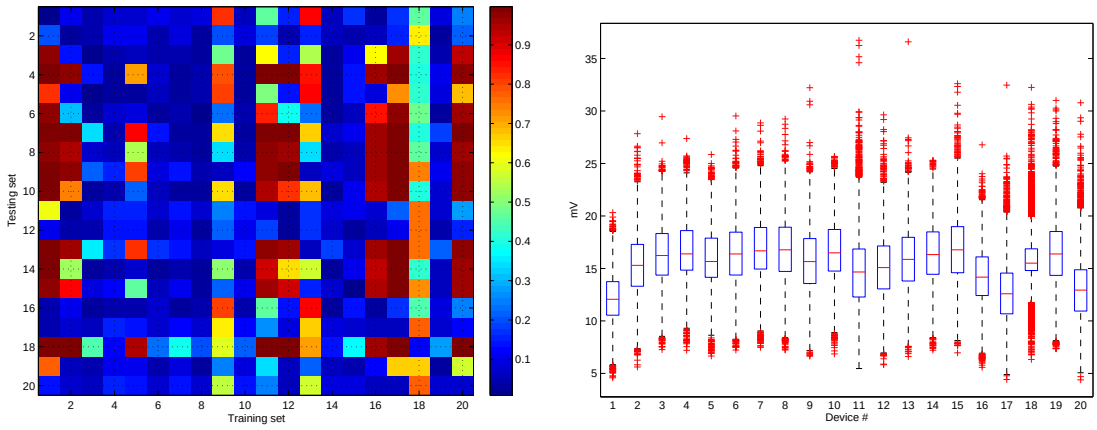
processors from each of $\{350, 130, 90\}$ nm manufacturing processes (*i.e.* nine in total) are used to evaluate the building of templates on one device and testing on another for various families.

Here 20 of the PIC smartcards, as used in the previous section, are used to examine power based templates, with the work in this section forming the basis of [91]. As mentioned, the clock supplied to the card is 4 MHz, and the sampling rate is 250 MSs^{-1}. For each card, 10 k power traces with random plaintexts and random keys are acquired. As no suitable trigger signal is available, the communication bus is used to trigger the oscilloscope leading to the traces being widely out of sync with each other. Each set is individually synchronised using cross-correlation after filtering, before reduction to the maximum point per clock cycle. Finally the *sets* of traces are aligned using euclidean distances between the *means* of each set.

While the Hamming weight model best fits the PIC smartcard as shown in Figure 3.12(a), the templates are built for the identity model as this allows for a more concise comparison based on the recovery of a key byte. Templates were first built with 9 k traces from each set individually, and these used to classify 1 k traces from every other set. To train the templates the z-scores of the data is first taken, then Fisher's linear discriminant is used to project the feature set into a reduced subspace retaining 20 features. When calculating the z-scores, it is important to note that the mean and standard deviation normalisation values for the testing data must be taken from the training set, *i.e.* it is not assumed that enough traces are present in the attack stage to accurately estimate the mean and standard deviation of the test set which is the approach taken in [154]. As an advantage of TA is that keys can be recovered with few or even a single trace, if many traces are present to accurately calculate the z-score, then a regular DPA attack can simply be performed.

The top left to bottom right diagonal in Figure 3.13(a) is the error rate when classifying the devices using templates generated by the same device. This can be viewed as a baseline 'best case' scenario (the green coloured bars in Figure 3.14 also display these values). It is clear from the image that classification is not even between all devices. For example, building templates with devices $\langle 1, 2, 11, 12, 16, 17, 20\rangle$ generally return a higher error rate regardless of what device is being tested, as can be seen by the "redder" colouring. On the contrary however, templates built with devices $\langle 6 - 8\rangle$ allow for a low error rate when attacking any device as indicated by the blue.

It is interesting to note that pairs of devices don't always have similar classification as might be expected. For example, when device 1 is being attacked, building templates with



(a) Single-device templates.

(b) Sample means.

Figure 3.13: Template attacks using different devices.

device 13 returns a high error rate, while building with device 16 gives a low error. It is reasonable to expect that a somewhat similar relationship will hold when attacking device 2, however in that case both devices 13 and 16 give a low error rate. This can be further explored by examining Figure 3.13(b) where the point in time at the output of the *S-Box* is manipulated, selected via the SOST method, is represented by a box-plot for each of the 20 PIC smartcards. This allows the difference in the mean values, as well as the spread of the points as a whole, to be easily compared. It can be seen that the mean of device 1 is below the average, and less than the mean of both devices 13 and 16, however much closer to device 16. The mean of device 2 however falls in between those two means. This gives a rough indication as to why, counter-intuitively, the classification of pairs of devices don't always follow each other as expected.

A more general way to generate the templates, is to use traces from many devices as suggested in [188]. Figure 3.14 shows error rates where 9 k randomly selected traces from across 19 devices are used to generate the templates, and used to classify 1 k traces from the other device. Only 9 k in total are selected rather than 9 k from each to allow for a fair comparison with the same training set size. For comparison, the average error rate of generating the templates with difference devices, and the error rate of generating the templates with the same device are also given. When generating the templates from multiple devices, there is a substantial reduction in the error across most devices, with set 18 the only set retaining a substantial error.

This section highlights that the presumption that templates are immediately applicable to any other 'identical' device is not entirely accurate. It must also be kept in mind

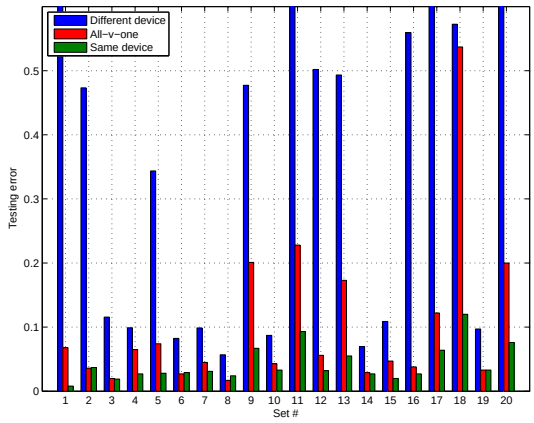


Figure 3.14: Multi-device templates.

that the PIC smartcard cards used in this work are inherently quite basic devices so are likely to more 'similar' than other devices such as the ARM7 microprocessor[3]. Overall performing TAs while using the same device to acquire both training and testing traces has merits however, especially for penetration or standardisation labs, as it can be viewed as worst case scenario testing, *i.e.* real world performance is likely to be somewhat less than projected. From an adversarial point of view however, if multiple identical devices can be used to attack a separate unseen device, then the attack performance can be improved.

3.6 Conclusion

TAs have been extensively examined in this chapter, with various training options explored with respect to how they effect the resultant classification error. The effect of increasing noise and different target platforms was also examined. Finally the validity of the underlying assumption that an *identical device* is available is explored using favourable devices.

It was found that taking the z-scores of the data should be the first step in any profiling attack as it helps reduce numerical errors by scaling the data. Using a pooled covariance matrix with LDA rather than the originally suggested QDA, improves both the classification performance and reduces the computational complexity as only a single matrix needs to be inverted. The use of a single covariance matrix to model the noise is justified as the noise is independent of the feature components of each class. Looking to directly recover

[3]The PIC smartcards were used for the comparison rather than a different device such as the ARM7 microprocessor as many cards could be purchased due to their cheap cost.

the byte component as opposed to the Hamming weight can also be advantageous when attacking a software target as there will be multiple points in time where a linear function of the byte will be computed on which can also leak side-channel information that can be utilised by the model. Note when targeting hardware designs this is unlikely to be the case due to the varying initial register state, and following the Hamming distance model will possibly give better performance. Although feature selection is a vital step of an attack, it was shown that simpler methods such as SOST can perform as well as transforming the data through Fisher's linear discriminant, when LDA is used as a classifier and a larger number of features are retained. The larger number of features in the original set doesn't necessarily contain more leakage information however, as the projected traces are linear combinations of the entire original data set.

While different classification methods are looked at such as quadratic and linear discriminant analysis, as well as reduced TAs, a notable exclusion from this chapter is alternatives to TAs such as SMs [195] or SVMs [103, 135]. These are dealt with, along with other machine learning methods in Chapter 6.

Chapter 4

Trace Only Template Attacks - AES

4.1 Introduction

In the TAs conducted in the previous chapter, it is assumed that either the plaintext or ciphertext is available to use in conjunction with the power traces to recover the key. This generally is true also of most non-profiling attacks attacks against symmetric algorithms. In this chapter, TAs that only require a power trace are presented, *i.e.* no plaintext or ciphertext information is required, the key is recovered entirely from the power trace. This is achieved in two different ways, firstly by attacking more than one intermediate value in the trace [92], and secondly by building templates on the key value directly.

In some respects, building templates on multiple intermediate values is closely related to algebraic side-channel attacks [189]. Algebraic attacks were first proposed as a powerful cryptanalysis technique against block ciphers [55]. They were subsequently extended to a powerful class of side-channel attacks firstly against the block cipher Present [189], and then AES [187]. Algebraic attacks make use of SAT solvers to solve an over-defined system of equations to recover key information from a single trace in an unknown plaintext or ciphertext adversarial model. Templates are used to provide inputs to the SAT solver based on the power model of the device, from multiple intermediate stages of the encryption. Recently, the attacks were extended to include probabilistic intermediate values [168] to allow for non-ideal template classification which allows for use in practical scenarios.

In the attacks presented here, the restriction on recovering the key from a single trace is relaxed in order to reduce the complexity of the attack which makes use of SAT solvers. This is felt to be a reasonable trade-off as while recovering the key from a single power

trace highlights the power of SCA, the overheads of having a different key for every block of data to be encrypted are substantial so it is likely to be applicable in specific, real-world scenarios such as challenge response protocols. Hence there will be many cryptographic devices which will encrypt multiple blocks per key. It must be noted too also, that depending on the power model, *i.e.* if there is a bijection between the target intermediate values based on a function of the key, and the noise of the device under attack, the key can also be recovered from a single trace in the simpler attacks presented here.

Comparing to classical non-profiled attacks, attacking multiple intermediate values or operations can be viewed as a form of higher-order attack [150]. However here multiple, multi-variate attacks are performed, while in non-profiled attacks a combination function is often used to combine the data prior to performing a univariate attack (for a more detailed discussion on non-profiled higher-order attacks see [208]). While higher-order attacks look to overcome masking type countermeasures, here we look to remove the need for plaintext or ciphertext. The effect of the attack when the masking countermeasure is used is also examined in §4.3.

While it may not be immediately obvious why key recovery is useful if the corresponding ciphertext isn't known, there are many scenarios in which this could be useful. A concrete example is when data is encrypted in *Counter Mode* [66], as shown in Figure 4.1. Here, even if the ciphertext is known, the input or output of the encryption block itself is not known, so in the context of a SCA the ciphertext is essentially unknown. The nonce and counter values must still be known or guessed for plaintext recovery however. Knowledge of the key could allow an adversary to verify any guess made at the counter values, or even possibly perform a DPA attack against the inputs. A non-profiled attack which doesn't require knowledge of the plaintext, ciphertext or initial counter value was presented in [105] based on the sequential nature of the counter provided the input to the block cipher. Another attack not requiring plaintext values was presented in [143] against the key schedule, where multiple scenarios are presented where no plaintext knowledge is available, such as smart card applications employed by banks, credit card companies, pay-tv broadcasters *etc.* While the precise input value might not be known due to randomness, the protocol used or error-correcting-codes for example, the underlying cipher used such as AES or triple-DES (3-DES) might be recognisable through the power trace patterns.

In this chapter, an outline of how the recover key bytes from AES without any knowledge of plaintext or ciphertext values with the use of TA is given in §4.2, with the expected number of traces required derived in §4.2.1. An empirical evaluation is carried out on two

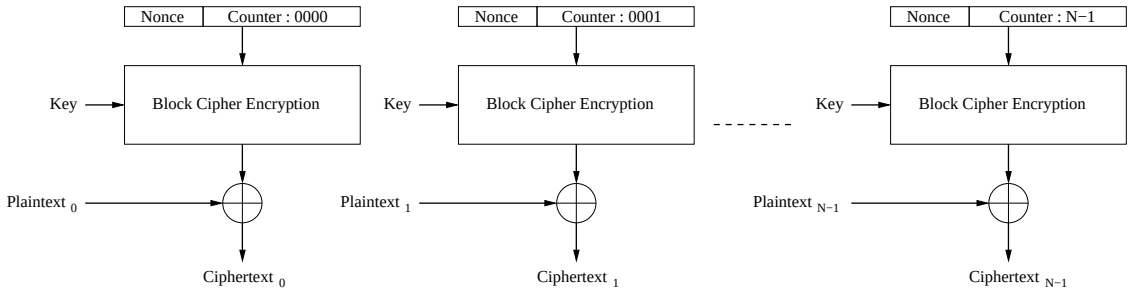


Figure 4.1: Block cipher counter mode of operation.

software targets in §4.2.2, with practical improvements to the attack suggested in §4.2.3. The effect of the masking countermeasure on the attack is discussed in §4.3, as well as strategies to overcome it. Finally an efficient TA on the key schedule is presented in §4.4.

4.2 Trace only Key Recovery

If we consider a straightforward implementation of a block cipher, a TA can be used to extract key information without using any known plaintext (or ciphertext) values by targeting two separate intermediate states in the encryption algorithm.

When attacking a block cipher a natural choice would be to build templates on y and z, where $z = S(y \oplus s)$ where S is a substitution table, and s is some portion of the secret key or sub-key. In the case of AES, y, z and s would each be eight bit values and $\langle y, z \rangle$ could be given by, but not limited to, the red or green pairs of lines in Figure 4.2.

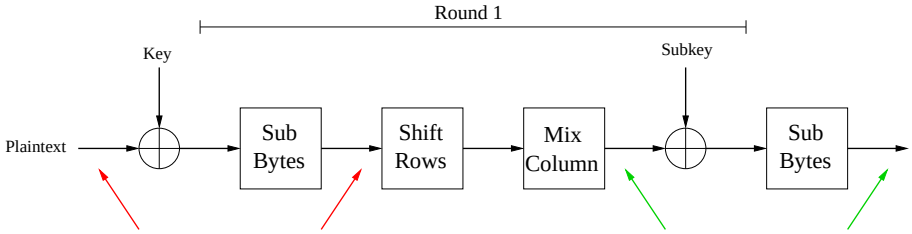


Figure 4.2: AES unknown plaintext attack target.

Where the templates are built for the intermediate value itself rather than the Hamming weight, and the intermediate targets are chosen such that there is a bijective function between them and the key value, the key can be recovered in a single trace. As shown in §3.3.5 however, in the specific case of AES, due to the linear properties of the *Mix-Columns* operation, the error rate is quite high when using the identity model. This can be expected to apply to other symmetric algorithms also as examined in [181]. Where the

templates are built following the Hamming weight model, an attacker would be interested in constructing templates to identify the Hamming weight of y and z and using this to derive s. If the second template is built on z, where z occurs before the substitution table (*i.e.* $z = y \oplus s$), the key cannot be extracted as there is a linear relationship between the two template sets and a unique value cannot be determined.

An advantage for an attacker is that the attack can be applied to any round of a block cipher, allowing countermeasures that are implemented on certain rounds, such as the initial or final rounds, such as suggested with masking in [6, 8], to be circumvented. An informal discussion is now given of the expected number of observations an attacker would be required to make to derive s if applying this attack using the Hamming weight model when targeting the *S-Box* function of AES is conducted. Of course, if the templates are built for the identity model then, given "clean" traces with a high SNR, the key bytes can be recovered from a single trace. However due to the greater separation required to distinguish between 256 rather than 9 templates, this will be somewhat less robust and more susceptible to noise for the *MixColumns* operation.

4.2.1 Computing the Number of Key Hypotheses for AES

Assuming templates are built for the Hamming weight of an intermediate value and that they are correctly classified each time, the amount of possible values that y and z can take can be computed from the Hamming weight h as $\binom{b}{h}$, where b is given by $\lceil \log_2(|\mathcal{K}|) \rceil$, where $\mathcal{K}$ is again the set y and z can be drawn from. Trivially, we can say that if either y or z have a Hamming weight of 0 or b, then the number of hypotheses returned for the key will be dictated by the Hamming weight of the other variable. For a given pair of observations, the number of valid key hypothesis returned can be computed by counting all the valid combinations when the inverse *S-Box* is applied to z. The number of valid key bytes for a pair of Hamming weight observations is given in Table 4.1.

The overall expectation can be computed from this by multiplying each table entry by the probability of it occurring, *i.e.* $\frac{\binom{b}{\mathrm{HW}(y)} \cdot \binom{b}{\mathrm{HW}(z)}}{2^{2\,b}}$, and computing the sum of the result. In the case of the $b = 8$ as is being examined, this produces an expected number of key hypotheses for a single observed pair, $\mathrm{HW}(y)$ and $\mathrm{HW}(z)$, to be 246.5. This means that one would expect to acquire $\log_2(256/246.5) = 0.0545$ bits of information from one observed pair, and to derive all eight bits would therefore be expected to require $8/0.0545 = 146.743$ observations. This is assuming that the Hamming weight is correctly identified every time, so this can be viewed as the expected lower bound on the number of traces required.

		HW(y)								
		0	1	2	3	4	5	6	7	8
HW(z)	0	1	8	28	56	70	56	28	8	1
	1	8	54	163	219	236	218	153	58	8
	2	28	146	251	256	256	256	249	153	28
	3	56	222	256	256	256	256	256	226	56
	4	70	244	256	256	256	256	256	246	70
	5	56	222	256	256	256	256	256	226	56
	6	28	146	251	256	256	256	249	153	28
	7	8	54	163	219	236	218	153	58	8
	8	1	8	28	56	70	56	28	8	1

Table 4.1: The number of key hypotheses.

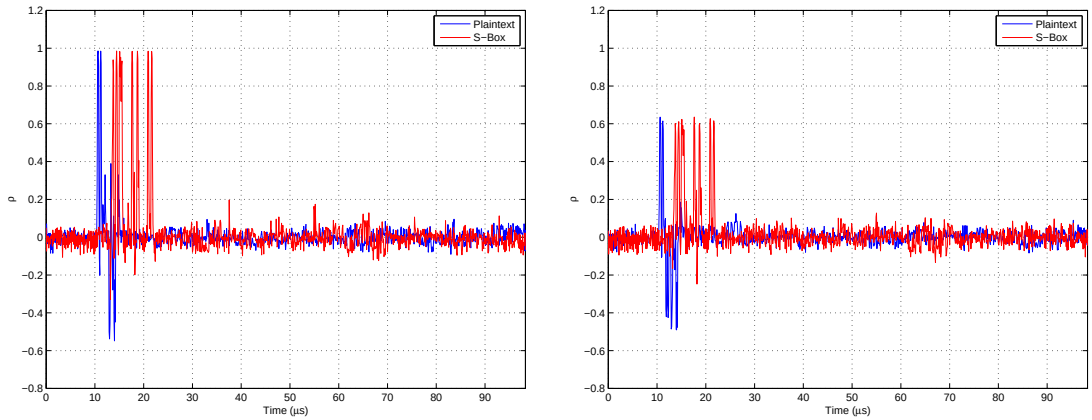
4.2.2 Experimental Results

To experimentally verify the attack, the Atmel 8051 microcontroller traces from §3.4 are used due as their leakage closely follows the Hamming weight model. Looking at the expected error rate for the *S-Box* as previously shown in Figure 3.12(b), it is clear that the identity model also leads to a very low error rate so the attack is conducted for both models. To re-iterate, the traces were acquired with a clock frequency of 11 MHz and a sampling rate of 250 MSs^{-1}. The mean of each trace was subtracted and it was then filtered with a low-pass filter with a cut-off frequency of 16 MHz.

Correlation Analysis

The correlation coefficient is useful sanity check to perform in order to determine the leakage from a given function and/or intermediate variable. The success or otherwise of a TA cannot however, be inferred by the strength of the correlation coefficient. Concentrating on the first possible attack target pair from Figure 4.2, *i.e.* the plaintext input and *S-Box* output, Pearson's correlation is calculated with the knowledge of the correct intermediate values in order to gauge the respective leakage of each value as shown in Figure 4.3.

There are two interesting points to take from Figure 4.3. Firstly, the correlation peaks for the Hamming weight model are close to one, and are higher than their respective identity model peaks which verifies that the power consumption closely follows the Hamming weight model, and secondly, there are many more peaks, in both models, for the *S-Box* operation



(a) Hamming weight model (b) Identity model

Figure 4.3: Correlation of intermediate target values → 1 k traces.

than for the plaintext where the *S-Box* output is re-used in the *MixColumns* function. Intuitively, one would then expect a higher error rate in the classification of the plaintext than the *S-Box* as fewer data-dependent points are available to generate the templates.

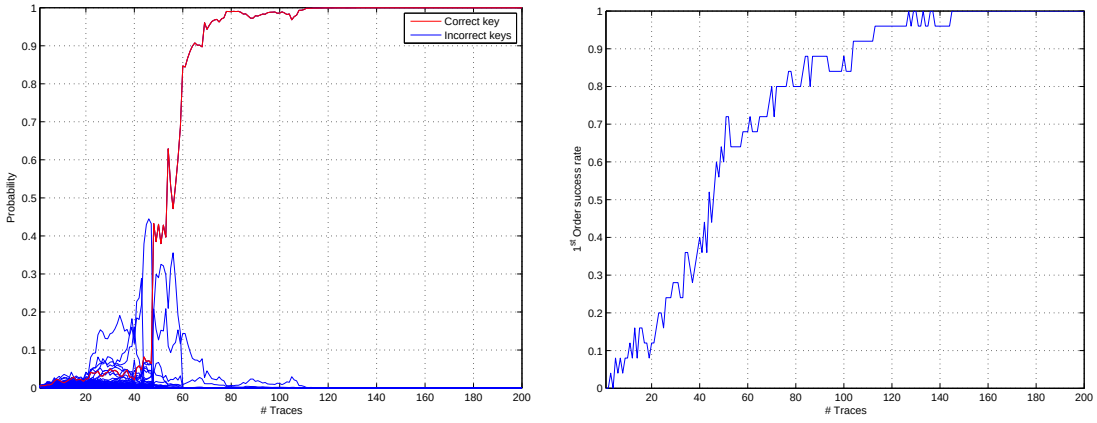
Template Attack

As only 10 k traces in total were available, the set was split into two sets of 5 k, one for training and one for testing. The SOST method of feature selection was used to select 30 points of interest to build the templates, with this value chosen from the learning curve in Figure 3.12(b). The z-scores of the trace set were taken prior to feature selection and the templates were built for both the Hamming weight and identity models. LDA with the pooled covariance matrix is used as a classifier.

The separate set of 5 k testing traces were then used to conduct the attack, split into 25 distinct sets of 200 randomly selected traces without replacement. This was to ensure that no short term acquisition effects could adversely effect the results. As the plaintext inputs are random uniformly distributed numbers, each key probability can be multiplied on a trace by trace basis for key recovery over a number of traces. In an ideal scenario when a key hypothesis is classified as incorrect, it has a probability of 0 so will subsequently be eliminated by multiplication. However, in a practical attack, the correct Hamming weight is not classified with a probability of 1, with incorrect values consequently 0. Therefore, keys are not ruled out on a trace by trace basis, but instead are multiplied by a smaller probability.

A single instant of an attack against the 8051 microcontroller when the Hamming weight

model is used is shown in Figure 4.4(a). After ≈ 60 traces the correct key byte has a significantly higher probability than the incorrect keys, and the probability is close to one after ≈ 100 traces. The success rate as averaged over 25 sets is given in Figure 4.4(b), and while 25 is a relatively small number to average over, the trend in the data is still clear. The empirically calculated expected number of traces required for a successful attack follows the theoretically calculated value of 146.7 quite well. This indicates that the Hamming weight for both the plaintext and *S-Box* is classified correctly the majority of the time. A closer inspection of the error rates for each individual Hamming weight shows that this is the case. In Table 4.2 the Hamming weight error between the actual and estimated values are given for both the plaintext and *S-Box* cases. When the attack was performed using the identity model, both target bytes were classified correctly nearly every time, with the error is given in Table 4.3. Hence the key was correctly recovered from only a single power trace over 99% of the time.



(a) Single attack instance

(b) Success rate over 25 sets

Figure 4.4: Unknown plaintext attack on 8051 with Hamming weight model.

Conducting the attack against the ARM7 microprocessor software implementation of AES the attack is not so successful however, with keys only recovered successfully sporadically when using the Hamming weight model, hence it is not plotted here (the success rate after 250 traces was ≤ 0.1). The same template generation parameters were used except 50 rather than 30 features were retained. Also, this time a larger trace set was available so 25 k training traces were used, and a separate set of 25 k traces were used for testing, split into 100 sets of 250 traces. Again, inspecting the Hamming weight error rates as shown in Table 4.2, it is clear that intermediate values are classified much less successfully than in the case of the 8051 microcontroller, especially so for the plaintext. Therefore the error in the cumulative probability of the keys prevents the correct one being identified.

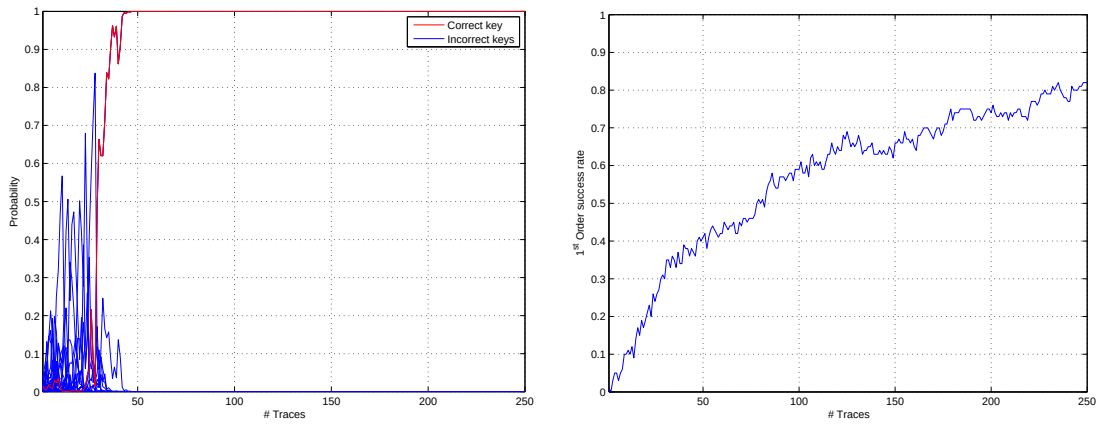
HW	8051		ARM7	
Error	Plaintext	S-Box	Plaintext	S-Box
0	0.9992	1.0000	0.2716	0.6155
1	0.0008	0.0000	0.4394	0.3628
2	0.0000	0.0000	0.2148	0.0211
3	0.0000	0.0000	0.0611	0.0005
4	0.0000	0.0000	0.0113	0.0001

Table 4.2: Comparison of Hamming weight errors.

	8051		ARM7	
	Plaintext	S-Box	Plaintext	S-Box
Error	0.0082	0.0082	0.9868	0.0320

Table 4.3: Comparison of byte errors.

Examination of the model error rates in the previous chapter such as in Figure 3.12(c), shows that this poor classification performance on the ARM7 microprocessor when using the Hamming weight model should not be too surprising. The same figure does show however that using the identity model gives a much lower error rate. The attack on the ARM7 microprocessor was re-run using the same parameters except using the identity model instead with the results given in Figure 4.5.



(a) Single attack instance.

(b) Success rate over 100 sets.

Figure 4.5: Unknown plaintext attack on ARM7 microprocessor with identity model.

The extremely poor classification of the plaintext values on the ARM7 microprocessor as shown in Table 4.3, requires that multiple traces can be expected to be required to recover the secret key, unlike the successful recovery with only a single trace for the 8051

microcontroller. This low classification rate is in line with the results from §3.3.5 where the classification of various target intermediate values were compared. On the one hand it is quite surprising that the attack works due to the fact that the plaintext is classified correctly so infrequently, however as noted previously, an incorrect classification does not mean that the correct plaintext value has been discarded, rather that it was not assigned the highest probability for that trace. Over a number of traces, eventually the correct value will be chosen with a greater frequency as shown in Figure 4.5(b), however the success of the attack cannot be guaranteed.

4.2.3 S-Box Only Attack

On the ARM7 microprocessor platform, given the discrepancy between the classification of the *S-Box* operation and the plaintext or *MixColumns* operations (Figure 3.8 shows that the classification performance of the *MixColumns* operation is similar to that of the plaintext), it makes sense to see can the key be recovered while solely targeting the *S-Box* operation.

Looking at the AES description in Appendix A, we can see that the *MixColumns* input to the *AddRoundKey* function, depends on 4 *S-Box* outputs. Therefore to recover a single sub-key byte, templates have to be built for 4 *S-Boxes* in round r and a single *S-Box* in round $r+1$ as illustrated in Figure 4.6. The four recovered bytes are then used to calculate the *MixColumns* operation output to allow for sub-key recovery. As the key expansion operation for AES is reversible, the recovery of a full round sub-key allows an attacker to learn the secret key.

This method of applying the attack introduces some restrictions however, as the Hamming weight model can no longer be used due to the fact that the combination of all possible permutation inputs to the *MixColumns* operation doesn't reduce the search space. Likewise, key probabilities from different traces can no longer be combined either, as combining the probabilities of each permutation of the 4 bytes doesn't allow for any one value to achieve a higher probability. In the previous attack strategy, incorrect keys were still retained and the cumulative probabilities removed it from contention. However, if targeting 4 bytes to generate the *MixColumns* output rather than the output itself, there are 2^{32} combinations to keep track of which is extremely computationally intensive[1]. A consequence of this is that there is no direct way to combine multiple traces for an amplified TA for key recovery

[1]While not computationally infeasible, the effort required would make it unlikely that this would be the most efficient method for key recovery.

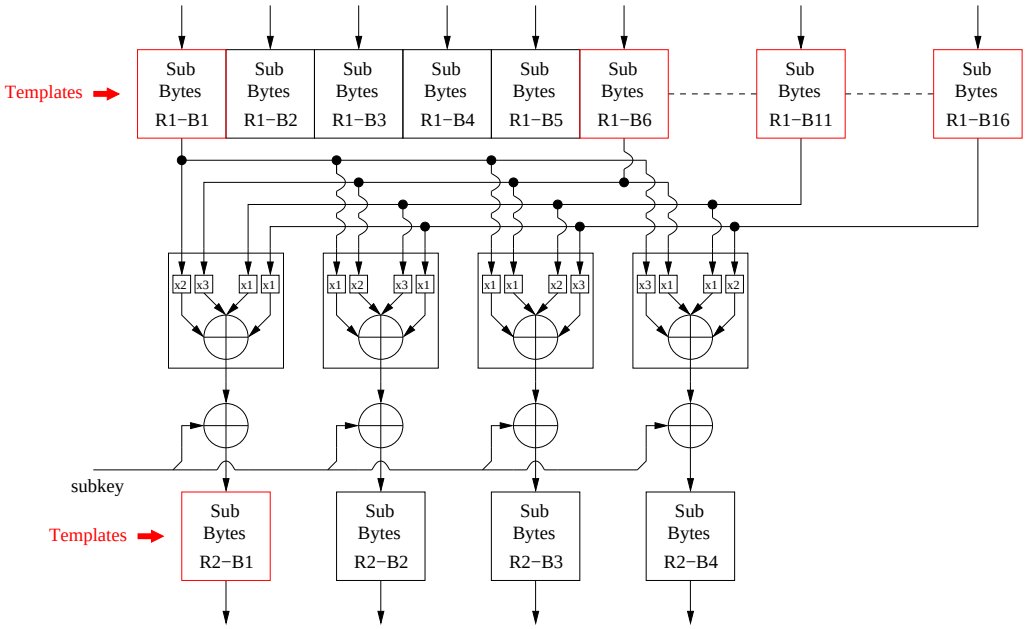


Figure 4.6: Target S-Boxes to extract the first sub-key byte.

bar majority voting, so where possible the keys should be extracted from a single trace.

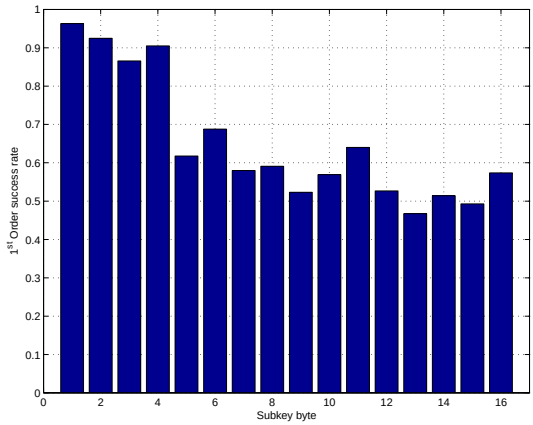


Figure 4.7: Success rate for each sub-key byte.

As *each* sub-key is dependent on four *S-Boxes* from round r and a single *S-Box* from round $r+1$, any time a single *S-Box* is classified incorrectly from round r, 4 sub-keys will be incorrectly guessed. Figure 4.7 gives the success rate for each of the 16 sub-key bytes calculated using the method described for round 1 of AES on the ARM7 microprocessor platform. It is interesting to note that the first 4 sub-keys are classified correctly significantly more often than the other sub-keys. A closer examination of the individual *S-Box* errors in Table 4.4 explains why, as *S-Boxes* $\langle 1, 6, 11, 16 \rangle$ from round 1 all have very low error rates, and due to the *ShiftRows* operation these are the 4 bytes that are used by

	S-Box							
Round	1	2	3	4	5	6	7	8
1	0.0087	0.1088	0.0753	0.0752	0.1271	0.0037	0.1555	0.1020
2	0.0104	0.0490	0.1103	0.0704	0.1068	0.0075	0.1632	0.1491

	S-Box							
Round	9	10	11	12	13	14	15	16
1	0.1170	0.0731	0.0050	0.1209	0.1279	0.1194	0.0797	0.0108
2	0.1911	0.1232	0.0124	0.1941	0.1931	0.1190	0.1467	0.0098

Table 4.4: Comparison of S-Box classification errors.

the *MixColumns* operation to calculate the input the the *AddRoundKey* operation for the first four bytes.

The overall success rate for a given sub-key byte in this experiment can be approximately estimated as $(1-0.0927)^5 = 0.6149$, where 0.0927 is the expected error rate averaged over all 32 *S-Boxes* from Table 4.4. This compares with the empirically calculated value of 0.6525 through direct calculation of the estimated *MixColumns* values. The discrepancy between the values, while small, is still relatively larger than might be expected given the sample size (10 k traces were used to estimate the error in this case), however this is due to the fact that each *S-Box* classification from round 1 affects 4 sub-key bytes, which was not taken into account.

4.3 Application to Masking

As outlined in [145], there are two main approaches to counteract power and electromagnetic attacks, *hiding* and *masking*. Hiding looks to obscure the amplitude of data-dependent leakage, for example with additive noise to decrease the SNR or custom logic styles [140] to equalise the power consumption. Alternatively traces can be desynchronised in the time domain, for example via shuffling of operations or randomising the clock. Each countermeasure has specific time, memory, area and power trade-offs depending on the user requirements or projected attacker model, and must be carefully selected to minimise any potential leakage. These type of countermeasures are generally applied in conjunction with some methods from the second class of countermeasures, masking [7, 41, 85, 150].

Masking, or secret sharing [161], are algorithmic level countermeasures which seek to break

the link between the intermediate value that is being processed and the power consumption by randomising the value for every encryption run. This prevents an adversary being able to hypothesise on intermediate values to perform a DPA attack. Of course such randomisation must be removed prior to the end of the algorithm to ensure correct operation. Therefore, there can be a significant performance and memory penalty incurred in keeping track of masks, and the number and updating of masks must be tailored to the device at hand. Common methods of masking are additive, affine, boolean, and multiplicative. It must be noted however, that while hiding and masking countermeasure combine to make an adversaries job considerably harder if implemented correctly, there is no technique that can guarantee perfect side-channel resistance against higher-order attacks [150] or more advanced power models such as those which take into account glitching [147]. A designer seeks to implement countermeasures such that the benefit of breaking the device is not worth the effort to do so.

Examining Boolean masking, anywhere an intermediate state is stored in memory, it is XORed with some random value that varies for each execution. This random value is chosen to be a convenient size for a given block cipher[2], *e.g.* for AES this would typically be an 8-bit value because of the *S-Box* function. The target equation would then become:

$$z \oplus \tau = S'(y \oplus s \oplus \tau) = S(y \oplus s) \oplus \tau \tag{4.1}$$

where τ is a random value generated for each execution of the implementation of AES. This is illustrated in Figure 4.8. Note this is not the only way to implement masking, for example in [7] the output of the masked *S-Box* function would be given as $S(y \oplus s) \oplus \tau'$, where τ' is a different mask. However for the purposes of the analysis this is the method we focus on. As previously mentioned, masking inner and outer rounds only as suggested in [6, 8] can be trivially overcome using the proposed attack, however the application of masking in this way has also been shown to be weak in the context of regular DPA [89] and is not recommended.

Using templates against masked implementations of AES was originally examined in [173, 178], as well as a different approach using *Gaussian Mixture Models* in [134]. In all these attacks the plaintext is required however. To attack a masked implementation of AES with unknown inputs where the target device follows the Hamming weight power model, the observations become $\mathrm{HW}(y \oplus \tau)$ and $\mathrm{HW}(z \oplus \tau)$. There will be some values of τ that will not be possible given a pair of observations (dictated by the structure of the function S), the

[2]This applies for other masking methods also.

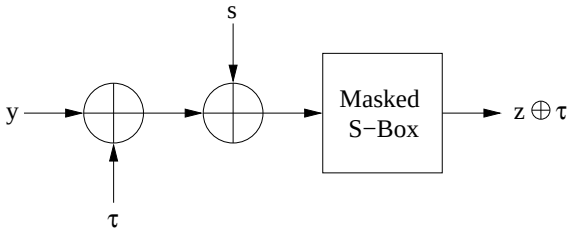


Figure 4.8: Boolean masking for AES.

simplest method of evaluating the expected number of hypotheses returned by evaluating a pair of observations is to again count all the possibilities as given in Table 4.5.

		HW$(y \oplus \tau)$								
		0	1	2	3	4	5	6	7	8
HW$(z \oplus \tau)$	0	163	254	256	256	256	256	256	254	163
	1	254	256	256	256	256	256	256	256	254
	2	256	256	256	256	256	256	256	256	256
	3	256	256	256	256	256	256	256	256	256
	4	256	256	256	256	256	256	256	256	256
	5	256	256	256	256	256	256	256	256	256
	6	256	256	256	256	256	256	256	256	256
	7	254	256	256	256	256	256	256	256	254
	8	163	254	256	256	256	256	256	254	163

Table 4.5: The number of masked key hypotheses.

As described in §4.2.1, the overall expectation can be computed from this by multiplying each table entry by the probability of it occurring, *i.e.* $\frac{\binom{b}{\mathrm{HW}(y\oplus\tau)}\cdot\binom{b}{\mathrm{HW}(z\oplus\tau)}}{2^{2\,b}}$, and computing the sum of the result. This produces an expected number of key hypotheses for an observed $H(y \oplus \tau)$ and $H(z \oplus \tau)$ to be 255.9. This means that one would expect to acquire $\log_2(256/255.992) = 4.299 \times 10^{-5}$ bits of information from one pair of observations, and to derive all 8 bits would therefore be expected to require $8/4.299 \times 10^{-5} \approx 186k$ samples, once again assuming the Hamming weight is correctly identified each time.

Examining this in the context of the identity model, each observation pair will return 163 valid keys, *i.e.* the same as when the Hamming weight observations in Table 4.5 are both either $\langle 0, 8 \rangle$. Following the analysis above, this gives an expected number of key hypothesis for an observed $y \oplus \tau$ and $z \oplus \tau$ pair to be $\frac{163 \times 256^2}{256^2} = 163$, giving $\log_2(256/163) = 0.6513$ bits of information. Assuming that each value is correctly classified each time, then

$(8/0.6513) \approx 12.3$ traces are required to successfully extract the correct key value.

4.3.1 Attacking Masked Implementations

Eliminating the Mask

To successfully attack a masked implementation while using the Hamming weight model, an attacker would be required to construct a third set of templates on τ. A known plaintext version of this attack is described in [173]. In the unknown plaintext scenario, the attack is restricted to the first round only, as one of the templates must be built on y before it is masked, not $y \oplus \tau$ which would allow any round to be targeted. In the case of building the templates for $y \oplus \tau$, removing the mask from both $y \oplus \tau$ and $z \oplus \tau$ by XORing both sets of values with the third template on the mask τ, limits the amount the valid key-space can be reduced when subsequently combining the values after the removal of the mask which prevents key extraction. On the other hand, by using templates built on τ to just remove the mask on z by XORing all possible combinations together, then the templates based on y can be used to extract the secret data s in $z = S(y \oplus s)$ as before. The attack reverts as the previously explained method without masking, at the cost of extra traces being required to deal with the extra unknown value τ. Hence, similar to the result presented in [173], Boolean masking does not theoretically prevent key extraction in the context of an unknown plaintext template attack, but the attack will be device dependent. However extra traces will be required regardless of the device under attack.

Extending this to the identity model, if templates could be built for τ, $y \oplus \tau$ (or simply y as in the Hamming weight case), and $z \oplus \tau$, then the key can be trivially extracted with a single trace if the values are successfully classified, as the mask can simply be removed. However previous experiments have shown that successful TAs against intermediate values other than the *S-Box* are non-trivial classification problems, and the extension of the attack to the *S-Box* only attack of §4.2.3 is not feasible as each byte will have a different mask value.

The validity of being able to build templates for the random mask values is also open to question. Ideally the masks would be generated internally on a secure device, so even though an attacker posses an identical device to generate the templates, knowledge of the mask value would be still unknown. However, in the context of worst case analysis it makes sense to give an attacker this capability. For example collision-based attacks [196] with known data could be used to estimate the mask values, or if there is the capability to program the identical device, then knowledge of the mask values could be read out

through some known function.

Predicting the Plaintext

It has also been shown in [105] that DPA is possible against an implementation of a block cipher where a counter is part of the plaintext, *e.g.* counter mode used for random number generation [66]. The initial value of the counter does not need to be known for an attacker to mount an attack against an implementation where no countermeasures are present. The same technique could be used to guess the values of a counter present in the plaintext to guess values of y such that information can be derived in a similar manner, by evaluating all the possible values of τ and $z \oplus \tau$, from templates constructed on $y \oplus \tau$ and $z \oplus \tau$. The advantage of this attack over eliminating the mask is that templates only need to be constructed on two points, however it is assumed the input can be accurately predicted somehow.

4.4 Building Templates On the Key

SCAs against the key schedule are somewhat under-represented in the literature relative to targeting intermediate values of an algorithm, as DPA requires randomly varying values to succeed. Also, where memory is not an issue, the key expansion might only be performed once prior to the encryption and stored in memory to encrypt many blocks of data. However in resource constrained devices where throughput is not a priority, such as smart-cards, the key expansion can often performed in line with the encryption.

Prior to the selection of *Rijndael* as the winner of the AES competition, a theoretical analysis of the key schedules for all AES candidates was carried out in [29], and a SPA attack against the key expansion of AES was presented in [143]. More recently, machine learning techniques were examined in [135] while targeting the key of a 3-DES implementation. Although the authors did not build their models on the key schedule *per se*, the labels were assigned directly on the key bits hence the classification is still determined by the key expansion. As the key schedule of DES consists of bit-shifts and permutations rather than an non-linear operations such as in AES, it is suited to this kind of analysis as each key bit will influence the power consumption across multiple rounds, as shown originally in [1].

The SPA in [143] can be viewed as a template attack as the author looks to identify

Hamming weights of intermediate values of the key schedule and use this knowledge to reduce the brute-force search space of the key. Averaging is suggested to reduce noise to allow distinguishing between values, as well as profiling on another device as suggested originally in [70]. This paper pre-dates TAs as presented in [42], which are ideally suited to this scenario. An attack is now presented that can be viewed as an extension to that presented in [143].

Templates were built for each key byte directly with a set of 25 k traces from the ARM7 microprocessor, each trace with a different uniformly random plaintexts and keys. As before the data was normalised by taking the z-scores of the data, the identity model was used, and SOST was used to select 50 features to build the templates with. The results of using LDA for each key byte on a separate set of 1 k traces are given in Figure 4.9. As a comparison the peak correlation value between the 1 k traces and corresponding key bytes is also given so show that the presence of a correlation peak does not necessarily imply a successful TA.

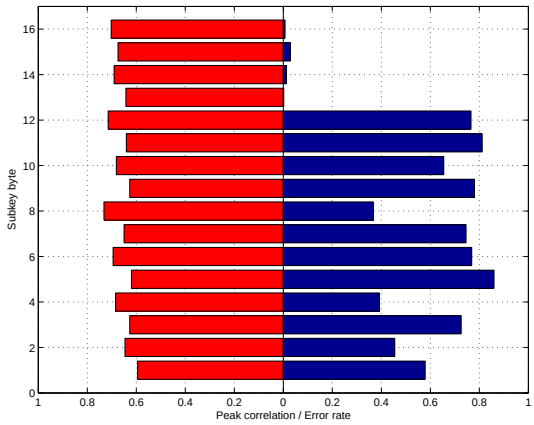


Figure 4.9: Error rate for each key byte → 1 k traces.

As has been shown previously, the classification of the output of the *S-Box* function far outperforms that of the other functions, which is again the case when building templates directly for the key as is clear from Figure 4.9. Even though all 16 sub-key bytes have a strong correlation with the power traces, only the final 4 key bytes have low error rates. Examining the key schedule from [163], it can be seen that these four key bytes are passed through the *S-Box* function.

This leads to the question of can all the key bytes be recovered by solely recovering the output of the *S-Boxes* in the key schedule which is now examined. The difference between the attack presented here and the one in [143], is that while in [143] the author assumes

the Hamming weight of arbitrary values in the key schedule can be recovered, here the attacker is assumed to be able to successfully recover the output byte value of the *S-Box* functions only. Note the attack in [143] can be seen as a precursor to Algebraic SCA [189]. Depending on underlying leakage properties of the target device, this assumption might hinder or assist the attacker.

While the analysis here is for AES with a 128-bit key, it is equally valid for 192 and 256-bit keys. Following the notation of [163], the entire secret key is split up into 32-bit words, with the entire sub-key consisting of 44 of these words. As mentioned previously, recovery of any single sub-key for any round (*i.e.* the consecutive words $w_{0,1,2,3}$ where $\langle 0, 1, 2, 3 \rangle \in 0 \ldots 43 \ (mod\ 4)$), allows an attacker to recover the actual key due to the reversible nature of the key schedule. The key expansion consists of *rotations*, XOR's and the *S-Box* function. The *S-Box* function, which is the attack target, is only used on every 4^{th} word, *i.e.* 4 bytes of each round sub-key.

Recovery of the four bytes of w_3 is possible by attacking the first use of the *S-Box*, where in the following description *S-Box* indicates the use of it across the 4 bytes of a word. Attacking the next use of it then allows an attacker to recover w_7. However as w_3 is also known, this allows w_6 to be recovered as when $i \neq 0 \ (mod\ Nk)$ (for AES-128, $Nk = 4$), then $w_i = w_{i-1} \otimes w_{i-Nk} \rightarrow w_6 = w_7 \otimes w_3$. Advancing to the next use of the *S-Box* gives an attacker w_{11}, with $w_{6,7}$ used to recover $w_{9,10}$. Finally attacking the 4^{th} use of the *S-Box* in the key expansion combined with $w_{9,10,11}$ gives an attacker the entire sub-key, $w_{12,13,14,15}$, which is the sub-key used for round 3 of the encryption. Note this is not taking advantage of all instances of the *S-Box* used in the key expansion, where a sub-key byte is classified with a small probability, further rounds can be looked at to increase the certainty of the classification. Figure 4.10 gives the success rate over 1 k sets for key recovery *of the entire* 128-*bit key*, where multiple traces are available to allow an amplified TA, with the product of the success rate for the individual key bytes taken to calculate this overall success rate.

4.5 Conclusion

This chapter is based on the work published in [92], and outlines TAs against AES in which it is assumed an adversary has no knowledge of the inputs or outputs of an AES encryption. A single plaintext/ciphertext pair is useful for an attacker *after* key recovery to test the key hypothesis, although by no means necessary. The expected number of traces required for a successful key recovery in the case of a target device following the Hamming weight power

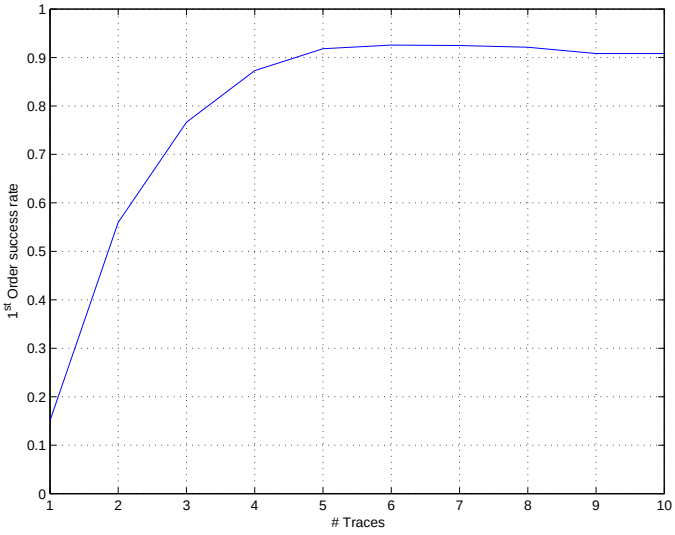


Figure 4.10: Success rate for the entire key $\rightarrow$ 1 k sets.

model was developed, and compared to empirical estimations from two different target devices, an 8051 microcontroller which closely follows the Hamming weight power model and an ARM7 microprocessor which does not.

The attack was subsequently modified to achieve better classification results while targeting the ARM7 microprocessor, by focusing only on the *S-Box* operations which have been both proved in the literature [181], and heuristically shown in the majority of DPA publications to be an suitable target for SCA. Two *S-Box* only attack methods were presented, one targeting the data path and another attacking the key schedule.

The application of the attacks in the context of masking was examined, however there is scope for much further work in this regard. As well as improvements in algorithmic or key recovery efficiencies, unsupervised or semi-supervised machine learning algorithms such as that in [134, 137] could be a quite interesting avenue to further explore. Unsupervised learning algorithms could be especially useful in the context of randomised countermeasures such as masking as mask knowledge might not be required in the profiling step [134]. Clustering has already been suggested as both a DPA type SCA distinguisher [20], as well as for key recovery in the context of non-profiled single trace attacks for public-key algorithms [100] which is the topic of the next chapter.

Chapter 5

Trace Only Template Attacks - Multiplication

5.1 Introduction

While the attacks in Chapter 4 looked to recover the secret key from ciphers by building templates on intermediate values in an algorithm, in this chapter a different approach is taken, again with the goal of key recovery from a single trace where possible. While asymmetric ciphers such as RSA [192] and ECC [123, 152] are targeted, the approach also can also work for some symmetric ciphers depending on their underlying primitive operations. The work in [93] forms the basis for this chapter. While attacks in the literature are often explained in the context of RSA or ECC, due to the duality between the *square & multiply* $\Leftrightarrow$ *double & add* methods of implementation, the attacks are often applicable to both.

To recover the secret key, the approach here is to try and distinguish between multiplication and squaring operations, even when the same software code or hardware unit is used to calculate the operation so as to prevent SPA type attacks. It was shown by Amiel *et al.* in [10] that the expected Hamming weight of the result of a multiplication is different to that of a squaring. This is utilised to build templates to distinguish between the operations as detailed below.

The construct of asymmetric algorithms is vastly different from that of symmetric ones due to the need for a back-door function to enable the public/private key pair to encrypt/decrypt. For example, the security of RSA is based on the assumed difficulty of

factoring large integers, while the security of ECC is based on the assumption that finding the discrete logarithm of a random elliptic curve element with respect to a known base point is computationally infeasible. While the required doubling and addition group operations for ECC are performed on an elliptic curve with a given set of parameters, the underlying addition, subtraction, multiplication and inversion field operations are generally performed in prime, $GF(p)$ or binary Galois $GF(2^m)$ fields. Integer multiplication and squarings are an essential operation in both algorithms, as well as many other asymmetric algorithms such as *ElGamal* [76] or *Paillier* [176], and it is these operations that are now targeted in this chapter. The attack is demonstrated by attacking a single Montgomery multiplication algorithm [153], with subsequent application to a secure RSA cryptosystem explained. The attack method is then extended to a practical demonstration against ECDSA power traces.

The ability to recover the secret key component from a single trace has many advantages, not least in the context of countermeasures such as outlined by Coren in [52] which might seek to restrict an adversary to a single side-channel trace, but also in the case of ECDSA where the scalar operand is different on every use [165] by design which inherently limits an adversary to just one sample.

A very brief overview of RSA and ECC is now given in §5.2, followed by a look at some of the proposed *single-trace* attacks against them in §5.3. The source of the leakage that is targeted in this chapter is explained in §5.4 and the building of templates with this leakage to attack a multi-precision multiplication is demonstrated in §5.5. How this affects secure RSA implementations is examined in §5.6, with the attack extended to a practical demonstration against an ECDSA ephemeral scalar multiplicand in §5.7. An attack on a hardware Montgomery multiplier is examined in §5.8 before a brief overview of where the attack could be applicable in a symmetric-key setting in §5.9. Various countermeasures are looked at in §5.10, before conclusions are drawn in §5.11.

5.2 Asymmetric Algorithms

While asymmetric cryptography incorporates encryption, key exchange and signature protocols, the attacks here focus on RSA exponentiation [192] and the ECC scalar multiplication [123, 152], which form the basis for these constructions. A brief overview of both is now given.

5.2.1 RSA Exponentiation

RSA is a widely used algorithm introduced by Rivest, Shamir and Adleman [192] which provides both encryption and signature capabilities. A public key accessible to anybody is used for encryption of data, with only the owner of the corresponding private key able to decrypt it. Its security is based on the difficulty of factoring the product of two large primes p and q. A modulus $N = p\,q$ is computed which is part of both the public and private key. A count of the relatively prime numbers to N is then calculated by $\phi(N) = \phi(p)\,\phi(q) = (p-1)\,(q-1)$, where ϕ is Euler's totient function. The second part of the public key e is then chosen such that $1 < e < \phi(N)$ and $\gcd(e, \phi(N)) = 1$. The corresponding private key part d is calculated as $d \equiv e^{-1} \mod \phi(N)$, which can be derived using the extended Euclidean algorithm. Encryption and decryption is simply modular exponentiation as shown in Equation 5.1.

$$\begin{aligned} c &= m^e \mod N \\ m &= c^d \mod N \end{aligned} \tag{5.1}$$

When implementing Equation 5.1, the bit-size of the private key d should be at the *very least* 2048-bits in size hence an efficient implementation such as that given in Algorithm 5.1 [149] will be required, however the public key e can be chosen to be much smaller. Here the bits of the exponent are stepped through one at a time, performing a squaring on each step and a multiplication where the exponent bit is 1, with all operations done modulo N. In practice RSA needs to be used in conjunction with some padding mechanism such as *PKCS #1 v2.2* [129] due to the homomorphic properties of the exponentiation. As embedded devices will generally have an $8, 16$ or 32-bit single-precision multiplication unit, the multiplication and squaring operations in each loop are still much too large to be implemented directly. Hence each is a *multi-precision* operation, consisting of many *single-precision* multiplications which use the hardware multiplier unit in the processor core.

5.2.2 ECC Scalar Multiplication

Elliptic curves have recently gained widespread acceptence due to their lower key sizes for equivalent security which has speed and storage advantages, particular for low power embedded designs. For the work here, an elliptic curve $\mathcal{E}$ over a prime finite field $\mathbb{F}_p$

Algorithm 5.1: Square and multiply algorithm.

Input: N, $x < N$, $e \geq 1$, b the binary length of e (*i.e.* $2^{b-1} \leq e < 2^b$)

Output: $A = x^e \mod N$

```
1 A ← 1 ;
2 for i = b downto 0 do
3 |   A ← A · A mod N ;
4 |   if e_i = 1 then A ← A · x mod N ;
5 end
6 return A ;
```

consists of points (x, y), with $\langle x, y \rangle \in \mathbb{F}_p$, that satisfy the short Weierstraß equation in Equation 5.2.

$$\mathcal{E} : y^2 = x^3 + a\,x + b \tag{5.2}$$

with $\langle a, b \rangle \in \mathbb{F}_p$ also, and the point at infinity denoted $\mathcal{O}$. The short Weierstraß form of the curve is used since the attack targeting ECC described in this thesis will be conducted against an instance of ECDSA using the P-192 elliptic curve, as described in the FIPS 186-3 [165]. The set $\mathcal{E}(\mathbb{F}_p)$ is defined in Equation 5.3.

$$\mathcal{E}(\mathbb{F}_q) = \{(x, y) \in \mathcal{E} \,|\, x, y \in \mathbb{F}_p\} \cup \{\mathcal{O}\} \tag{5.3}$$

where $\mathcal{E}(\mathbb{F}_p)$ forms an Abelian group under the chord-and-tangent rule and $\mathcal{O}$ is the identity element [198]. The addition of two points $\mathbf{P} = (x_1, y_1)$ and $\mathbf{Q} = (x_2, y_2)$ with $\mathbf{P} \neq -\mathbf{Q}$ is given by $\mathbf{P} + \mathbf{Q} = (x_3, y_3)$ where:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= (x_1 - x_3)\lambda - y_1 \end{aligned} \tag{5.4}$$

$$\text{with } \lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{if } \mathbf{P} \neq \mathbf{Q} \text{ [addition operation]} \\ \dfrac{3{x_1}^2 + a}{2y_1} & \text{if } \mathbf{P} = \mathbf{Q} \text{ [doubling operation]} \end{cases}.$$

The scalar multiplication of a given point is a fundamental operation in cryptographic algorithms that use elliptic curve arithmetic, *i.e.* $[k]\,\mathbf{P}$ for some integer $k < |\mathcal{E}|$. This

operation uses algorithms analogous to standard exponentiation algorithms such as in Algorithm 5.1, with squarings replaced by point doublings, and multiplications replaced by point additions.

5.3 Single-Trace Attacks

The original SPA attack [126], as previously mentioned in §2.3.2, showed that straightforward implementations of both RSA and ECC could be trivially broken through visual inspections of the power traces due to the different power consumption profile of the conditional operation. This can be easily prevented through the use of square & multiply always (or double & add always for ECC) [52], or through unified [35] or atomic [44] algorithms. A more through examination of countermeasures and how they affect the proposed attack is given in §5.6.

One of the first single trace attacks against asymmetric algorithms was the *Big Mac* attack, introduced by Walter in [219]. This is a non-profiled attack. which, like the work here, also tries to recover secret exponent bits by distinguishing between side-channel atomic [44] multiplication and squaring operations, as well as not requiring knowledge of the input plaintext values. A template trace is required, but can be computed from the first multiplication operation of the target trace which is known to be a squaring operation for RSA (see line 3 of Algorithm 5.1). It works against both *m-ary* and sliding windows exponentiation methods [149, Ch. 14], however cannot recover key bits from regular exponentiation algorithms which perform the same order of operations regardless of the key [110]. Average traces for the single-precision multiplications in each loop of a multi-precision integer multiplication are computed and compared to the known squaring multiplication at the beginning of the algorithm. As the power consumption of the multiplier is correlated to the number of bits that flip, and the number of bits that flip is also proportional to the Hamming weight of the input operands, squarings can be distinguished from multiplications by examing the Euclidean distance between the power consumption of the unknown operation under consideration and the initial known squaring operation.

Another single trace attack is *Horizontal Correlation Analysis* as introduced in [48]. This is a powerful attack to recover secret key exponents by correlating across the single-precision multiplications in a long-integer multiplication to determine if a given operand was used. If a correlation is present then the operation was a multiplication, otherwise it was a squaring. This method has the advantage of working against regular exponentiation

algorithms, unlike the *Big Mac* attack and the work presented here, but requires knowledge of the padded message or curve base point. While suggestions are given to overcome scenarios where the base exponent is blinded (hence unknown) [43], the effectiveness of the attack is reduced. Another interesting single-trace attack using clustering algorithms was introduced in [100]. Using this non-profiled machine learning method, the authors were able to recover the secret exponent from an elliptic curve scalar multiplication. However, multiple measurements from the same execution were required to improve the SNR to allow the attack to succeed.

Recent advances in *Collision attacks*, first introduced in [196] targeting DES, have also led to single-trace key recovery against asymmetric algorithms. The *doubling attack* [75] was the first collision attack against RSA, with furter attacks introduced in [102, 230, 233]. These attacks require two or more acquisitions however, hence can be protected against through randomising the exponent (or scalar multiplier). *Rosetta analyses* [47] can overcome this limitation and allows an attacker to distinguish between multiplications and squarings regardless of message or exponent blinding. The collision attacks presented in [90] expand this even further by detailing how to attack an exponentiation even when implemented as a regular algorithm. While collision attacks can be powerful, in reality all the attacks proposed to date require relatively *clean* traces to allow the collisions to be detected which is where templates have a significant advantage, at the cost of stronger adversarial model.

5.3.1 Template Attacks on Asymmetric Algorithms

Most TA to date have focused on symmetric algorithms, generally targeting AES, DES, and to a lesser extend RC4. One of the first TAs published against an asymmetric algorithm was against ECDSA [148]. The authors built templates for multiple intermediate values for the initial bits of the ephemeral ECDSA key. This was then either iteratively repeated (*i.e.* the templates for the entire key could not be built in advance), or else lattice-reduction based attacks used to recover the full key from a few known exponent bits as shown in [160]. While the attack only requires a single trace to work, the base point is required hence it is ineffective against multiplicative blinding of the base point. A related paper [97] extends this work to show how the masking of the point can be removed by building templates for the Hamming weight of intermediate partial products in a long integer multiplication, and subsequently filtering out invalid masks which cannot occur. The attack of [148] can then be applied to recover the ephemeral key. Another TA

is shown against an FPGA target [101] where the authors perform an attack against a decapsulated *Xilinx Spartan-3* FPGA running an elliptic curve scalar multiplication. This attack is somewhat different from the attacks in [97, 148], or the work presented here as they target leakage from the physical location of registers rather than intermediate values or operations. A TA against ECDSA was presented in [157], however the templates are built for the modular inversion required in ECDSA rather than attacking the elliptic curve operation itself.

There is a large body of publications on secure implementations and attacks for both the RSA and ECC cryptosystems. Informative surveys of attacks against ECC and subsequent countermeasures are available in [59, 71, 72], while a survey of regular exponentiation algorithms for both ECC and RSA is available in [110, 112]. The attack as presented here has advantages over many other single-trace attacks due to its applicability in the presence of many countermeasures, and the fact that its not as susceptible to noise as some other methods. This of course comes with the tradeoff that a similar device must be available to build templates.

5.4 Target Leakage

It was observed in [10] that the expected Hamming weight of the *result* of a multiplication is different than the *result* of a squaring operation. The difference in the expected Hamming weight can be observed by measuring these side channels as shown in [10]. As this observation applies to a single-precision operation, *i.e.* whenever the hardware multiplier unit in a microprocessor is used for example, this difference will be present in many time locations in any word-by-word multiplication algorithm. Non-profiling attacks based on distinguishing multiplications and squarings were also shown in [4, 223] on the basis that the number of internal bit-flips, hence power, in a hardware multiplier is linearly related to the Hamming weight of the input operands. However, as shown later, this is not the main source of leakage that is utilised here.

This observation on the leakage of multipliers assumes that the power consumption of the device under consideration is proportional to the Hamming weight of the values being manipulated. It was shown in Chapter 3 that for the ARM7 microprocessor the Hamming weight model is not wholly accurate. However it suffices for that attack here, particularly when coupled with that advantage that no intermediate values need to be known to conduct the attack which is advantageous when countermeasures are present as explained in

§5.6. Many devices also conform to the Hamming distance model, for example attacks on hardware such as FPGAs or ASICs rather than software implementations on microprocessors will often need knowledge of the previous register value in order to be successful. This complicates the attack somewhat as the power consumption is no longer solely dependent on the input operands. For the attacks presented in this chapter it is assumed that the target device corresponds to the Hamming weight model unless otherwise specified.

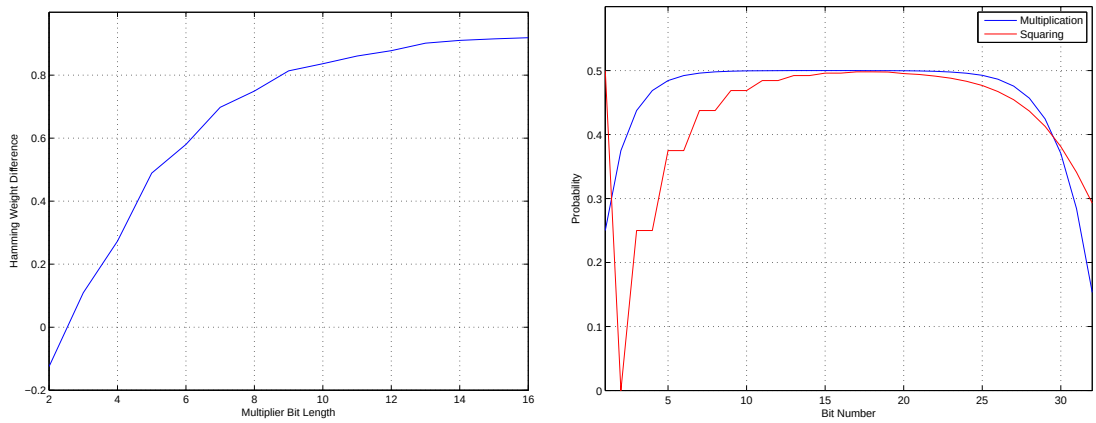
Assuming that a multiplication takes place between two random uniformly distributed b-bit values, the expected Hamming weight output can be calculated for smaller values of b using Equation 5.5 as shown in [10]. This essentially consists of taking the average of the Hamming weight of the product of all pairs of possible input values. Likewise, the expected Hamming weight value for a squaring can be calculated with Equation 5.6.

$$E(X \cdot Y) = \sum_{i=0}^{2^b-1} \sum_{j=0}^{2^b-1} HW(i \cdot j) \cdot Pr[X = i \wedge Y = j] = \frac{1}{2^{2b}} \sum_{i=0}^{2^b-1} \sum_{j=0}^{2^b-1} HW(i \cdot j) \tag{5.5}$$

$$E(X^2) = \sum_{i=0}^{2^b-1} HW(i^2) \cdot Pr[X = i] = \frac{1}{2^b} \sum_{i=0}^{2^b-1} HW(i^2) \tag{5.6}$$

Using these equations, the expected Hamming weight difference between multiplications and squarings for different operand bit lengths b were calculated as shown in Figure 5.1(a). It can be seen that for a 16-bit multiplication, the expected Hamming weight of the output of a multiplication is ~ 0.9 greater than that of a squaring. A closer examination of why this is so can be see in Figure 5.1(b) for 16-bit input operands, giving a 32-bit output. The probability of each bit output being equal to one is plotted for both operations. The restricted set of values that a squaring operation can take is reflected in the distribution of the lower bit probabilities. As this occurs every time a multiplication is calculated, this difference will be present in each of the single-precision multiplications that are required to compute a multi-precision multiplication. The plot in Figure 5.1 is only directly calculated for up to $b = 16$-bit operand lengths due to the computational complexity of greater bit-lengths, however the same leakage is present in larger bit-lengths also as shown in Appendix D where a subset of all possible values are selected.

Before looking at the practical effect of this on the ARM7 microprocessor device used previously, a short description of how the multiplier on this device operates is first outlined. The hardware long multiplication operation MULL on an ARM7 microprocessor is implemented such that it computes the multiplication of two 32-bit words in $2 \rightarrow 5$ clock cycles



(a) Hamming weight difference between operations.

(b) Probability that am output bit is 1.

Figure 5.1: Hamming weight analysis.

[138]. The number of clock cycles required depends on the bit length of one of the multiplicands, that is the multiplication will terminate early if a number of the most significant bytes of one multiplicand are set to zero. The precise algorithm of how the MULL op-code operation is defined in Algorithm 5.2 [87].

Algorithm 5.2: A functional description of ARM7 multiplication.

Input: The 32-bit integers x and y.

Output: The 64-bit result $A = x \cdot y$.

```
1 A ← 0;
2 for i = 0 to 3 do
3 |   A_step ← x · y_{7...0};
4 |   A ← A + (A_step ≪ 8i);
5 |   y ← y ≫ 8;
6 |   if y = 0 then return A;
7 |   ;
8 end
9 return A ;
```

Several attacks are described in [87] that demonstrate that naïvely using this multiplier to implement cryptographic algorithms will provide an attacker with an exploitable vulnerability. That is, the number of clock cycles taken by each multiplication can be seen in the power consumption and used to deduce information on the values being operated on.

This means that the multiplier has to be used such that it will perform a multiplication in a constant amount of time irrespective of the bit length of the multiplicands. An example of such an algorithm is shown in Algorithm 5.3 [87]. For the implementation of the attack

here, this algorithm was used to provide an implementation with a minimum level of security. That is, an implementation that is not vulnerable to attack by simply inspecting a small number of power traces.

Algorithm 5.3: A constant time algorithm to replace ARM7 multiplication.

Input: The 32-bit integers x and y.

Output: The 64-bit result $A = x \cdot y$.

1 $\gamma \leftarrow (y \wedge \texttt{00FFFFFF}_{(16)}) + \texttt{01000000}_{(16)}$;
2 $\tau \leftarrow (y \wedge \texttt{FF000000}_{(16)}) \gg 24$;
3 $A \leftarrow x \cdot \gamma$;
4 $A \leftarrow A + ((x \cdot \tau) \ll 24)$;
5 $A \leftarrow A - (x \ll 24)$;
6 **return** A ;

Algorithm 5.3 ensures a constant time implementation of the multiplication by replacing the most significant byte with $0x01$ hence always ensuring the full 5 clocks are required. The multiplication by the most significant byte is then performed separately as a 32×8 operation requiring 2 clock cycles.

Using Algorithm 5.3, the leakage of the multiplier output can now be examined. Figure 5.2(a) shows a portion of a power trace recorded while the ARM7 microprocessor was performing the constant-time multiplication as given in Algorithm 5.3. The trace is averaged over 10 k acquisitions with random uniformly distributed values. The sampling rate was 250 MSs^{-1}, and the clock frequency 7.3728 MHz, and for the example here no filtering nor reduction of the acquisitions was performed[1]. The 32-bit single-precision multiplication corresponding to line 3 of Algorithm 5.3 can be seen as a decrease in the power consumption at the start of the trace, while the shorter 32×8 multiplication of line 4 can be seen as the second shorter decrease just following it.

Following [10] where the power consumption difference between a multiplication and squaring is shown by looking at the difference of means of a large set of traces, Figure 5.2(b) shows the difference of means trace between the 10 k multiplication traces from Figure 5.2(a), and 10 k squaring traces. The largest peak occurs right at the end of the 32-bit single precision multiplication, just after 13.5 µs. There is a smaller peak towards the start of the multiplication, likely due to the effect of bit flips dependent on the inputs as explained in [4, 223]. The ARM7 microprocessor multiplier returns the 64-bit multiplicand as two 32-bit words [138], and as a sanity check, the correlation of the least significant

[1]The traces used in this example are a subset of the traces used in §5.5 prior to pre-processing.

word for the multiplication traces is given in Figure 5.2(c). The strongest correlation occurs at the same point in time as the DOM peak, which is expected from Figure 5.1(b). This confirms that the leakage at the output of the multiplier is what is causes the DOM peak. No DOM is present for the upper 32-bit word as examining line 1 of Algorithm 5.3, it can be seen that the most significant byte is replaced with $0x01$ therefore the upper bits where a difference is expected to occur cannot be set. While in [10] a DOM attack was suggested to utilise this leakage, templates are now built to allow distinguishing between the operations with a single trace.

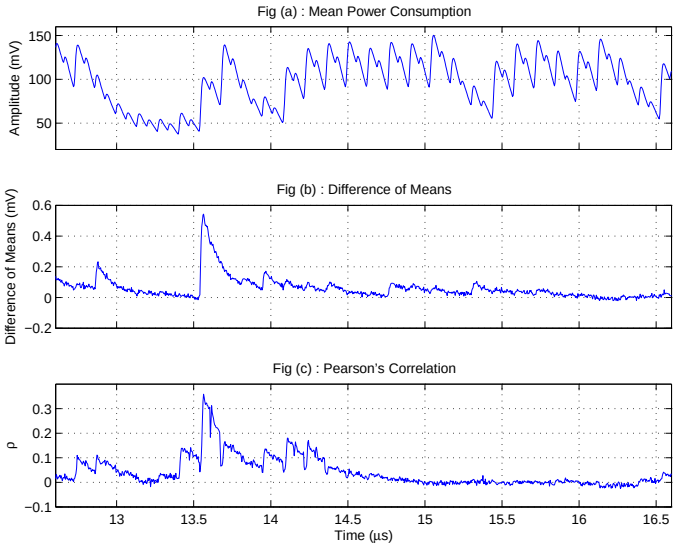


Figure 5.2: Multiplier target leakage.

5.5 Application of Templates

In this section an example TA exploiting the described leakage is conducted, targeting the Montgomery multiplication algorithm [153] as would be used in an RSA exponentiation, or the underlying field multiplications in an ECC scalar multiplication.

5.5.1 Montgomery Multiplication

When implementing modular exponentiation a commonly used multiplication algorithm is Montgomery multiplication [153], since the modular reduction is interleaved with the multiplication. The result of a modular multiplication using this algorithm is not $x\,y \mod N$, the algorithm actually returns $x\,y\,R^{-1} \mod N$, where $R^{-1} \mod N$ is introduced by the

algorithm as shown in Algorithm 5.4. $R = b^w$, where the modulus consists of w words of size b, with b generally chosen according the architecture size of the device being programmed for, *i.e.* $b = 2^{32}$ would generally be chosen for the ARM7 microprocessor under consideration as it has a 32-bit architecture. In order to use Montgomery multiplication, x and y need to be first converted to their Montgomery representation, *i.e.* $\widetilde{x} \leftarrow x\,R \mod N$ and $\widetilde{y} \leftarrow y\,R \mod N$. Then, when $\widetilde{x}$ and $\widetilde{y}$ are multiplied together using Montgomery multiplication, the result is $x\,y\,R \mod N$. This algorithm requires $2w\,(w+1)$ single-precision multiplications [149]. Where repeated multiplications are required, such as exponentiation or elliptic curve scalar multiplication, this has computational advantages as the division is simply shifting by a power of 2.

Algorithm 5.4: Montgomery product.

Input: $N = (N_{w-1}, \ldots, N_1, N_0)_b$, $x = (x_{w-1}, \ldots, x_1, x_0)_b$, $y = (y_{w-1}, \ldots, y_1, y_0)_b$ with $0 \leq x, y < N$, $R = b^w$, $\gcd(N, b) = 1$ and $N' = -N^{-1} \bmod b$.

Output: $A \leftarrow x\,y\,R^{-1} \bmod N$.

```
1 A ← 0 ;
2 for i = 0 to w − 1 do
3 |   u_i ← (a_0 + x_i y_0)N' mod b ;
4 |   A ← (A + x_i y + u_i N)/b ;
5 end
6 if A ≥ N then A ← A − N ;
7 return A ;
```

Algorithm 5.4 has been demonstrated to be vulnerable to side channel because of the final conditional subtraction that takes place if $A \geq N$ on line 6. This conditional subtraction affects the entire execution time of an exponentiation leading to an attack based on total time taken to compute an exponentiation [224]. It has also been shown that individual subtractions will be visible in the power consumption, or electromagnetic emanations, leading to efficient attacks [220, 221]. The simplest countermeasure would be to always conduct the subtraction and take the result as required. However, this approach is problematic since the bit that is used to make this choice may be visible in a side channel. Moreover, this could potentially be attacked by a fault attack, where a fault in a subtraction that does not affect the result of an exponentiation identifies a dummy subtraction [232], similar to the square and multiply always fault attack. More effective countermeasures involve increasing the number of iterations of the main loop so that the final subtraction becomes unnecessary [88, 218]. However, these attacks and countermeasures do not have any impact on the attack as described and can be considered beyond the scope of this

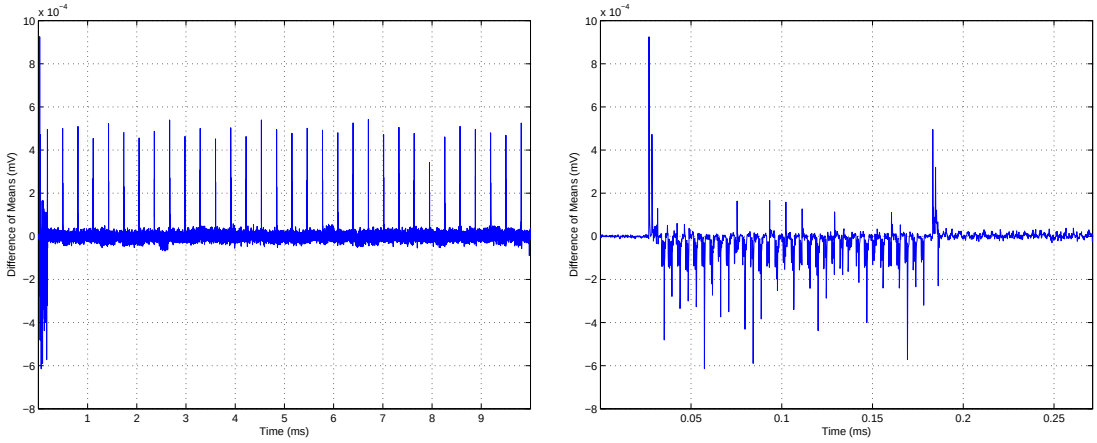
work. Note also that while the attack here is conducted against an implementation using the Montgomery multiplication, it is equally valid against many other multiple-precision multiplication algorithms such as long-integer or Comba.

5.5.2 Difference of Means

To put the attack into practice, a 1024-bit Montgomery multiplication algorithm was implemented on the ARM7 microprocessor and power traces of multiplication and squaring operations recorded. In all cases the inputs are uniformly randomly distributed. The acquisition parameters were the same as for the AES traces, *i.e.* a clock frequency of 7.3728 MHz, a sampling rate 250 MSs^{-1} with the 20 MHz bandwidth limiter of the scope turned on. Each trace was also pre-processed using the parameters as outlined previously in §2.10. In total, 10 k traces for each of multiplication and squaring operations were recorded, with 9 k of each used for training (*i.e.* $m = 18\,\text{k}$) with the remaining 1 k of each used for estimating classification accuracy. Each trace is randomly allocated to one of the two sets to remove any potential hidden bias present between traces that were recorded in close proximity time-wise. The operations are labelled such that y_0 represents a multiplication, and y_1 is a squaring.

The difference between the mean of the two classes of the training set is given in Figure 5.3(a). As the ARM7 microprocessor contains a 32-bit multiplier, to compute the 1024-bit multiplication $1024 \div 32 = 32$ iterations of the Montgomery multiplication loop are required. This can be seen in the 32-peaks of Figure 5.3(a), corresponding the the $x_i \cdot y$ operation on line 4 of Algorithm 5.4. This 32×1024-bit multiplication is performed as 32 separate single-precision multiplications. The power difference occurs when $x_i = y_j$ in the case of a squaring, as the equality isn't present (other than random chance) in the case of a multiplication.

A closer examination actually reveals 33 peaks, as well as a number of smaller peaks at the beginning of the trace. Figure 5.3(b) highlights the section which corresponds to the first loop of the Montgomery multiplication at the beginning of the trace in Figure 5.3(a). The initial large peak of Figure 5.3(b) at ≈ 0.03 ms is actually due line 3 of Algorithm 5.4 where $x_0 = y_0$ in the first round. The multiple smaller peaks can then be attributed to the re-use of u_0 on line 4 as $u_0 = x_0 \cdot y_0 \cdot N' \bmod b$.

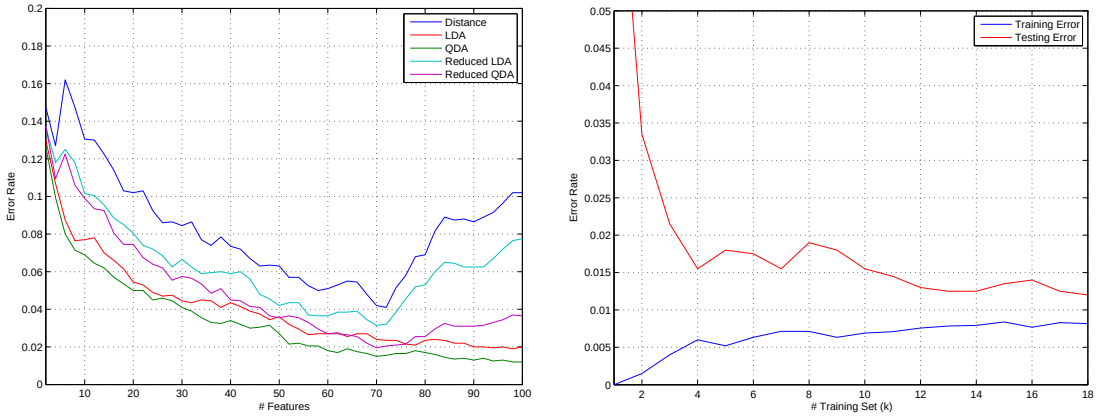


(a) All Montgomery multiplication loops. (b) Initial Montgomery multiplication loop.

Figure 5.3: Difference of means trace.

5.5.3 Template attack

To perform the template attack, the peaks from Figure 5.3(a) were used to build the templates. As there are only two class labels, *i.e.* multiplication and squaring, taking the absolute magnitude of the peaks is the same as the SOST method as outlined in §3.3.3. Prior to feature selection, the traces were normalised by taking the z-scores of the traces. Using the full set of $m = 18\,\text{k}$ training traces, the error rate of classifying the multiplication and squaring test traces is given in Figure 5.4(a).



(a) Error rate as a function of features. (b) Error rate as a function of training set size

Figure 5.4: Error rate for 1024-bit Montgomery multiplication.

It can be seen that very low error rates are achievable for the attack on this particular device. The five different classification methods from §3.3.6 are used, *distance* again simply denotes the Euclidean distance. The error rates are given as a function of the

number of features retained, with the features sorted according to the height of their absolute peaks. There are two peaks that can be utilised to build the templates for each single-precision multiplication on line 4 of Algorithm 5.4 which leaks information, along with the extra peaks from the initial round. It is interesting to note that for the Euclidean distance and reduced LDA methods, where no covariance matrix and a shared variance vector respectively are used, that the error increases with the number of features when the number of features is $\gtrsim 70$, which roughly corresponds to the two points per peak. These particular features have lower absolute difference peaks hence contribute less to the classification, or might not be leakage points at all. QDA gives the most accurate classification results as the large training set coupled with the small number of classes, $|\mathcal{K}| = 2$, allows the mean vector and covariance matrix to be accurately estimated for both classes. QDA outperforms LDA here as the covariance matrices are no longer equal due to the small peaks from the partial products in the initial Montgomery multiplication loop.

To check if the acquisition of a greater number of traces for template building would be beneficial, the training and testing error rates are plotted as a function of the training set size in Figure 5.4(b). The z-score normalisation was again re-calculated on each iteration only using the training trace sub-set, which themselves were randomly selected out of the full training set. QDA was used to classify the data while retaining 100 features. As can be seen both the training and testing error are quite low and stable, indicating extra training samples won't greatly improve the results.

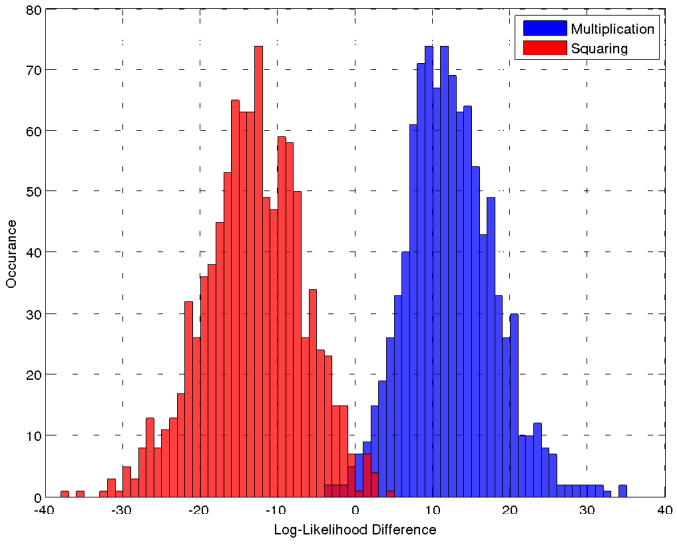


Figure 5.5: Log-likelihood comparison for multiplication and squaring operations.

As it is a binary classifier and amplified template attacks are not being considered (*i.e.*

the operation needs to be determined from a single trace), there is no need to compute the probability of each operation when classifying. Hence in the results of Figure 5.4 only the log-likelihood is considered, with the operation being assigned according to the template which returns the maximum log-likelihood. To visualise the overall classification performance, the *difference* between the log-likelihoods for both operations are given as a histogram in Figure 5.5, coloured according to their actual class. For each test trace, the log-likelihood of it being a squaring was subtracted from the log-likelihood of it being a multiplication. Hence, if the result is positive it is more likely to be a multiplication, and if negative then it is assigned to be a squaring. The histogram in Figure 5.5 gives the distribution of both operations, with 0 as the decision boundary. This plot also shows that the error (*i.e.* the blue occurances less than zero or the red greater) is evenly distributed between both operations, with no bias towards classifying one of them with a greater accuracy.

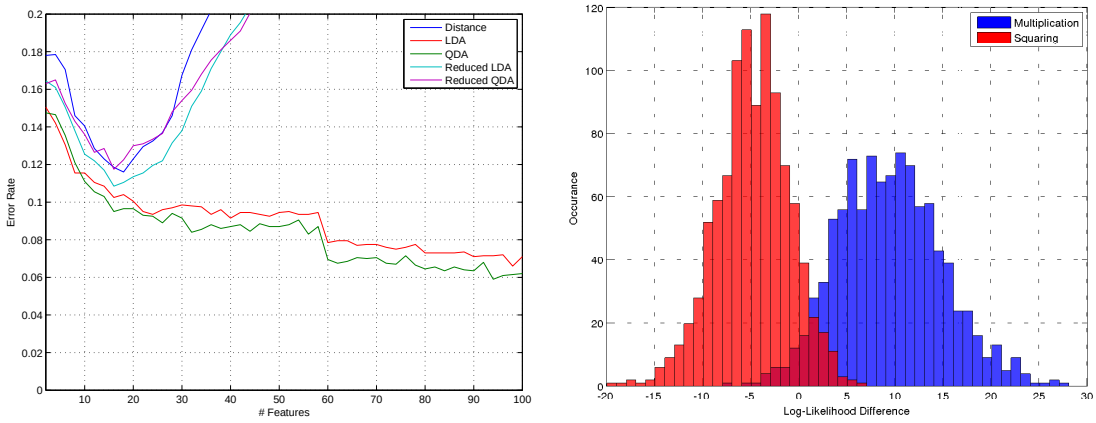
Effect of Multiplier Length

The low classification error rate of the testing traces in the previous section benefits from the large number of features that are available to build templates. For the 1024-bit Montgomery multiplication example, on the ARM7 microprocessor architecture with a 32-bit embedded multiplier, a total of 33 single-precision multiplications will leak information, as well as the multiple smaller peaks. A 1024-bit multiplier such as this would be used for the multiplication or squaring operation in the loop of Algorithm 5.1 in a typical RSA encryption with 1024-bit keys. For longer keys, the number of available features should be even greater, *e.g.* for a high security 4096-bit exponent, there will be 129 multiplications available to characterise a Montgomery multiplication on this platform. Therefore, as previously noted for the same reasons in [47, 220], long keys should counter-intuitively provide less security against this attack as there are a greater number of single-precision multiplications to model.

Conversely, this also means that for shorter exponent lengths that there is less information leakage hence they are harder to break. While RSA exponents should obviously never be set less than 1024-bits, and should be set to *at least* 2048-bits for any new implementations, ECC provides equivalent security for much smaller bit-lengths. For example a 1024-bit RSA key and a 160-bit ECC key, both have approximately the same security against known mathematical attacks, which is computationally equivalent to brute forcing 80-bits of a symmetric cipher key. While for ECC the Montgomery multiplication is used

to calculate the group operation rather than the main addition chain as in RSA, it is included here to see how the multiplier bit-length affects the attack, while a full attack on a scalar multiplication is subsequently given in §5.7. For an wide-ranging comparison on the required key lengths for various cryptographic algorithms see [83].

The attack is re-run against a 160-bit Montgomery multiplication using exactly the same acquisition settings, post-processing and template training parameters. There are now $w = 5$ words in Algorithm 5.4, leading to 6 single-precision multiplications that can be utilised to generate the templates with. As can be seen in Figure 5.6(a), the error rate when using QDA has now risen to just over 0.062% up from 0.012% from the longer multiplication before, which is also clear from the greater overlap in the histograms of the log-likelihood difference plot in Figure 5.6(b). As before there are two points per multiplication that are efficient to construct the templates with and, as can be seen in Figure 5.6(a), retaining $\gtrsim 20$ features leads to a higher error rate for classification methods which don't take into account the covariance matrix, *i.e.* least-squares, and reduced LDA/QDA. Taking into account the fewer single-precision multiplicatoins, this is similar to the 1024-bit case in Figure 5.4(a) where when retaining $\gtrsim 70$ features an increase occurs for the same classification methods.



(a) Error rate as a function of features.

(b) Log-Likelihood comparison for multiplication and squaring operations.

Figure 5.6: 160-bit Montgomery multiplication analysis.

An important point to note however is that although the error rate is higher for elliptic curve size multiplications, the elliptic curve doubling and addition group operations will contain many of these prime field multiplications. This is explored in more detail in §5.7.

Building Templates from a Single Multiplication

It was suggested in [93] that when performing TAs against asymmetric algorithms on an embedded device, an identical device might not be required for characterisation. If the public verification function executes using the same code, which is a reasonable assumption, then templates could be built using power traces from these operations instead. This is a significant weakening of the attacker capabilities both in terms of having an *open* identical device to program, as well as eliminating the effect of the power consumption variation between devices. Another possibility is that an attacker might be able to access the multiplier in a non-cryptographic setting and have the ability to build templates on just a single, single-precision multiplication. It is this scenario that is now examined.

Using the 1024-bit Montgomery multiplication traces from before, the 33 multiplications which leak information are extracted. While cross correlation using the difference of means peaks is used here to extract the required multiplications, in a *real-world* attack the power traces could just as easily be extracted without this information due to, either the regular nature of the algorithm, or correlating with the known input data of whatever function is being used to find the multiplication location. This time, rather than utilising all 33 single-precision multiplication leakages, just one is randomly selected from each of the $m = 18\,\text{k}$ training samples. The rest are discarded to allow comparison of the results with the previous experiments by keeping the same training set size. Once again $2\,\text{k}$ testing traces are used, with the single-precision multiplications extracted in the same manner. All 33 of these are retained however.

To build the templates, all 50 points of the reduced trace multiplication are retained and normalised by taking the z-score. The templates are then applied to each of the single-precision multiplications of the testing traces individually. The classification results of Figure 5.7 are for taking an increasing number of multiplications into account. Where only a single multiplication is used, the attack is barely better than randomly guessing the operation. However as the number of multiplications is increased comparable performance to building templates with the entire trace length is achieved, even though the same template has been used to classify every multiplication. The error rate from building templates on the entire trace from §5.5.3 is given in red for comparison.

As expected, the greater the number of multiplications utilised, the lower the error rate although there are some exceptions which adversely affect the error. When all the multiplications are used, the error rate is quite similar to the original results from using the full trace length giving credence to the hypothesis that, where possible, non-cryptographic

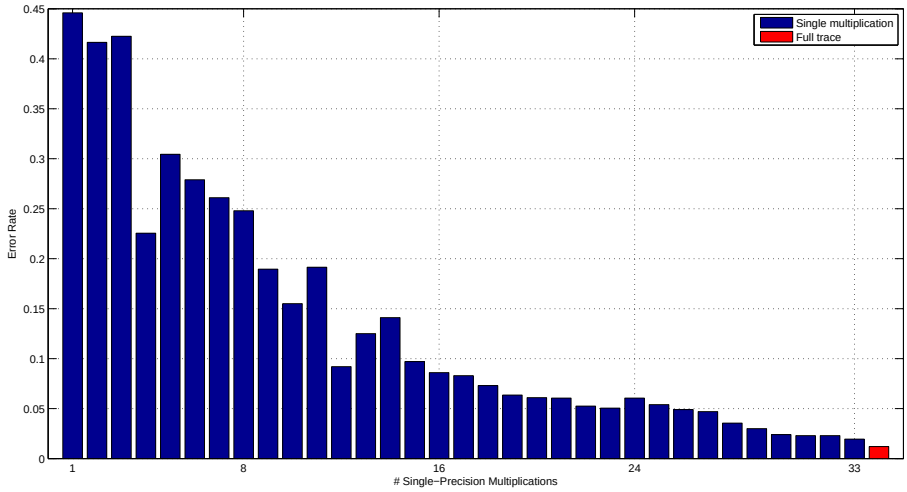


Figure 5.7: Classification using a single multiplication.

functionality of a device could be used to build templates. Although this is a somewhat contrived experiment, it gives a useful insight into the portability of the attack, as well as providing a nice visualisation as to how a longer key helps rather than hinders the attack as explained previously, as well as providing the basis for an interesting avenue of future research.

5.6 Application to Secure RSA Implementations

The attacks to date show how to distinguish between a multiplication and a squaring operation, with various examples for a single Montgomery multiplication of different word sizes. The implications of this for secure RSA implementations is now discussed.

As mentioned previously, if an RSA exponentiation is straightforwardly implemented using the left-to-right binary exponent algorithm as given in Algorithm 5.1, the key can be trivially read out from a power or electromagnetic emanation trace due to the different power profile of the multiplication and squaring operations. The most straightforward method to counteract this is through the use of the square & multiply *always* algorithm, where the multiplication is always performed regardless of the bit value, and discarded if not required. This obviously has a performance disadvantage as on average half the performed multiplications are redundant. However it is also susceptible to *Safe Error* [232] type attacks where an attacker can introduce faults during a multiplication operation. Given a correct/faulty exponentiation pair, it can be determined if the operation where the fault was injected was redundant allowing the key bit to be determined. Hence

regular algorithms such as the Montgomery powering ladder [113] where the order of operations is the same regardless of exponent, are preferred. Greater information on regular exponentiation algorithms can be found in [110, 112]

Another option is to remove the difference between a squaring operation and a multiplication. This means that, to square a value x, the calculation of $x^2 \mod N$ is replaced with $x \cdot x \mod N$ rather than using a dedicated squaring algorithm which is often the case in resource constrained environments. This idea is put forward in [44], where two instructions are defined as side channel equivalent if they are indistinguishable through side channel analysis, and algorithms are termed side channel atomic if the algorithm can be broken down into indistinguishable blocks. This principle, applied to the square and multiply exponentiation in Algorithm 5.1, is demonstrated in Algorithm 5.5. This algorithm would prevent an attacker from being able to distinguish a multiplication from a squaring operation by simply observing the difference in a side channel, since an optimized squaring operation is not used.

Algorithm 5.5: Side channel atomic square and multiply algorithm.

Input: N, $x < N$, $e \geq 1$, b the binary length of e (*i.e.* $2^{b-1} \leq e < 2^b$)
Output: $A_0 = x^e \mod N$

1 $A_0 \leftarrow 1$; $A_1 \leftarrow x$; $i \leftarrow b-2$; $k \leftarrow 0$;
2 **while** $i \geq 0$ **do**
3 $\quad A_0 \leftarrow A_0 \cdot A_k \mod N$;
4 $\quad k \leftarrow k \oplus \text{bit}(e, i)$;
5 $\quad i \leftarrow i - \neg k$;
6 **end**
7 **return** A_0

If RSA is implemented in a side-channel atomic manner following Algorithm 5.5, typically using Algorithm 5.4 or a variant to implement to multiplication step, recovering the secret exponent is equivalent to distinguishing between the multiplication and squaring operations hence side-channel resistance cannot be assumed. However, when implementing a cryptographic algorithm on a device that is potentially vulnerable to side channel analysis, one would typically include other specific countermeasures, in conjunction with side channel atomicity, to prevent an attacker being able to derive information on cryptographic keys.

For RSA this would typically mean implementing the blinding countermeasures, such that the input message and exponent are randomised on every execution, following the blind

signature scheme [43]. These countermeasures modify the exponentiation algorithm such that the values being operated on at a given point in time cannot be predicted by an attacker. Given that the Hamming weight leakage model as utilised above is based on the number of bits being manipulated at a given point in time, these countermeasures randomize the bitwise representation of all the variables being manipulated. For instance, to implement a function to compute a RSA signature $\varsigma = \varphi(m)^d \mod N$, where m is the message, d the secret exponent, and φ an appropriate padding scheme such as PKCS #1 v2.2 [129], each variable would be modified with a random value as shown in Algorithm 5.6. This is referred to as blinding since an attacker is unable to determine the values being manipulated at a given point in time, and different intermediate values would be computed for two instantiations of the same signature.

Algorithm 5.6: Blinded exponentiation.

Input: m, d, N, φ a padding function, ϕ Euler's Totient function and non-zero random values r_i, for $i \in \{1, 2, 3\}$.

Output: $\varsigma = \varphi(x)^d \mod N$.

1 $m' \leftarrow \varphi(m) + r_1 N$;
2 $N' \leftarrow r_2 N$;
3 $d' \leftarrow r_3 \phi(N) + d$;
4 $\varsigma' \leftarrow \mu(m)^{d'} \mod N'$
5 **return** $\varsigma' \bmod N$

The addition of $r_1 N$ to $\varphi(m)$ provides a redundant representation of $\varphi(m)$ modulo N and is referred to a message blinding. Given that $\varphi(m) + r_1 N \equiv \varphi(m) \mod N$, and this will remain true for all the intermediate states during the execution of the algorithm, the computation should not take place in the set of invertible elements $(\mathbb{Z}/N\mathbb{Z})^*$ (where $\mathbb{Z}/N\mathbb{Z} = \{0, 1, \ldots, N-1\}$) since the blinding would be removed. The bit length of the modulus is therefore increased by multiplying it by a random value so that the computation takes place in $(\mathbb{Z}/N'\mathbb{Z})^* = (\mathbb{Z}/r_2 N\mathbb{Z})^*$ instead. The values held in memory at a given point during the computation of the exponentiation cannot be predicted without knowing these random values. However, the values held in memory at a given point in time would represent the same value in $(\mathbb{Z}/N\mathbb{Z})^*$ for a fixed exponent.

The bitwise representation of the exponent is also modified by replacing the exponent d with $r_3 \phi(N) + d$, where ϕ is Euler's Totient function, and is referred to as exponent blinding. Any value raised to a multiple of $\phi(N)$ will be equal to itself in $(\mathbb{Z}/N\mathbb{Z})^*$, i.e. it is an identity function. Adding a multiple of $\phi(N)$ to the exponent changes the bitwise representation of the exponent without changing the result of its use in $(\mathbb{Z}/N\mathbb{Z})^*$.

One would typically choose the values of r_i, for $i \in \{1, 2, 3\}$, to be at least 32 bits. As noted in [199], one would also want the bit length of r_2 to be longer than the longest run of zeros in the bitwise representation of $\phi(N)$ so that all the bits of d are blinded. Furthermore, some side channel attacks could potentially derive an exponent if the bit length of these random values is too small [48].

When implementing a TA as described on the output of the multiplication operation, the use of message blinding aids an attacker as it randomises the bitwise representation of the values being operated on by the exponent. This helps an attacker since the values being operated on will be random and uniformly distributed and any values set to a constant will be randomized, *e.g.* the PKCS padding scheme sets parts of its output bytes to fixed values. The increase in the bit length of the modulus will also mean that the Montgomery multiplication will require at least one more iteration of the main loop, providing more single-precision multiplications that can be analysed by an attacker. As illustrated in Figure 5.7, the more single-precision multiplications available to build templates with, the higher the success rate of the attack.

The use of exponent blinding randomizes the bit wise representation of the exponent and therefore randomizes the sequence of multiplication and squaring operations that are computed. An attacker would be unable to take several acquisitions in an amplified TA type attack to improve the quality of classification since the operation being computed at a given point in time during the execution of the algorithm will vary from one instantiation to another. This means that an attacker has to determine whether an operation is a multiplication or a squaring operation from one single acquisition. From a practical aspect, an attacker also needs to acquire a single trace that includes the power consumption during an entire exponentiation to attempt to derive the exponent used, and therefore recover a value equivalent to the exponent d. Due to the computation time of RSA, it is a non-trivial issue to record a power trace of this length while retaining a high enough sampling rate. Alternatively some synchronous sampling method such as the low-cost proposal in [167] could be used to reduce the memory and storage requirements.

The TA results from the previous section can distinguish between 1024-bit multiplications and squarings with an expected error rate of only 0.012, so for a given acquisition of Algorithm 5.6 on an ARM7 microprocessor, an attacker could expect to correctly identify $\approx 99\%$ of the bits of an exponent. The incorrect classification of operations can be corrected to a certain extent, since the Hamming weight of the entire exponent can be computed by observing the total number of operations, and that a multiplication will always be preceded,

and followed by, a squaring operation in a side-channel atomic implementation such as Algorithm 5.5. One approach to conducting this analysis is through the use of hidden Markov models (HMMs) as suggested in [86, 116, 172]. Overall the attack highlights the importance of using regular exponentiation algorithms such as the Montgomery Ladder [113] which perform the same sequence of operations regardless of the input hence, distinguishing between a multiplication and squaring operation does not leak key information. Further countermeasures are examined in §5.10.

5.7 Attack on ECDSA

The attack is now extended to recover the secret multiplicand from an ECC scalar multiplication. Unlike previously where the training and testing traces were of the same format, *i.e.* the both consisted of a single Montgomery multiplication, here templates are built for point doubling and addition operations, while the testing traces are a full ECDSA power trace. The individual point operations must first be extracted, and the same templates then used to determine each operation. The ECDSA algorithm is given in Appendix B, however it suffices to say here that the scalar operand k must be random for each signature so that only a single trace is ever available to recover the secret, similar to exponent blinding in the case of RSA. Where k is not always random, it leads to a complete break of the system which is what happened in the Sony $PS3$ console attack [38]. Hence only the ECC scalar multiplication component of ECDSA is of interest here.

In practice a combination of countermeasures is used to increase the level of side-channel resistance. To protect the ARM7 microprocessor implementation against side-channel attacks, a number of countermeasures are also implemented. Randomised projective coordinates are used where the base point is blinded by replacing the Z-coordinate with a 192-bit random value and modifying the x and y-coordinates as required on every execution. Projective coordinates [229] are an efficient implementation method to avoid the need for the costly point inversion operation when computing point doubling and addition by changing the point representation from (x, y) to (X, Y, Z). The introduction of Z means that the points are no longer unique hence any Z value can be chosen to calculate $x = \frac{X}{Z}$ and $y = \frac{Y}{Z}$. A side channel atomic left to right binary multiplication algorithm is used following Algorithm 5.5 where squarings are replaced by doublings, and multiplications by additions. Unified group operations [35] as described for Edward's Curves in [24] and given in Algorithm 5.7 are used. Blinding of the base point $\mathbf{P}$ as suggested in [52] is not implemented, however as no knowledge is assumed of $\mathbf{P}$ this does not affect the attack.

Likewise, randomisation of the scalar multiplicand [52] similar to the exponent blinding for RSA is unnecessary as the scalar is only used once regardless.

Algorithm 5.7: Edwards curve → unified addition formula.

Input: Input projective coordinate points $\{X_1, Y_1, Z_1\}$,$\{X_2, Y_2, Z_2\}$

Output: Output projective coordinate point $\{X_3, Y_3, Z_3\}$

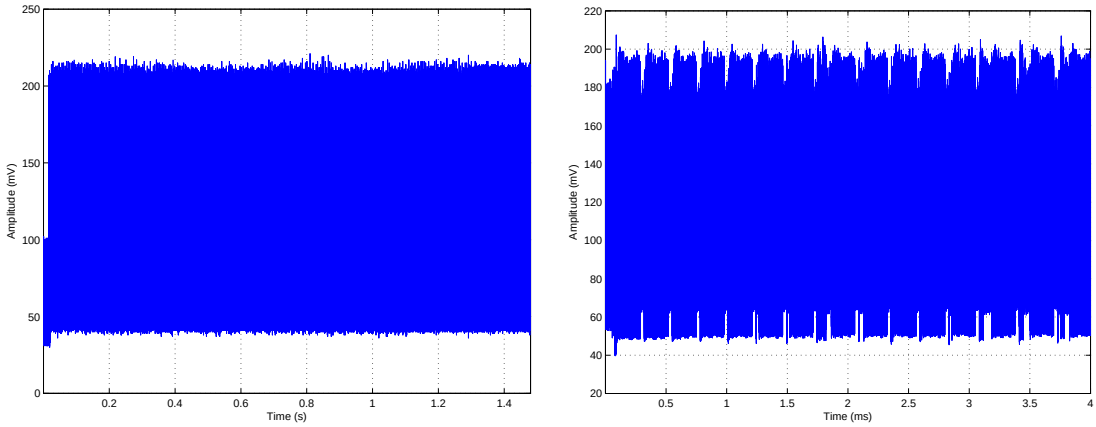
1 $A \leftarrow Z_1 \times Z_2$;
2 $B \leftarrow A^2$;
3 $C \leftarrow X_1 \times X_2$;
4 $D \leftarrow Y_1 \times Y_2$;
5 $E \leftarrow D \times C \times D$;
6 $F \leftarrow B - E$;
7 $G \leftarrow B + E$;
8 $X_3 \leftarrow A \times F \times ((X_1 + Y_1) \times (X_2 + Y_2) - C - D)$;
9 $Y_3 \leftarrow A \times G \times (D - C)$;
10 $Z_3 \leftarrow C \times F \times G$;
11 **return** $\{X_3, Y_3, Z_3\}$

This TA is considerably different to that presented against ECDSA previously in [97, 148] as no knowledge of intermediate values is required, hence the use of point blinding or randomised projective coordinates has no bearing on the outcome of the attack as a hypothesis is not being made on intermediate values. Similarly, the previously mentioned TA against ECDSA by De Mulder *et al.* [157] targets the modular inversion, and the attack by Fouque *et al.* presented in [74] looks to defeat exponent randomisation by attacking the randomisation steps, rather than attacking the elliptic curve operation itself.

The acquisition setup for these traces was slightly different, with the traces recorded at the University of Bristol by Dr. Mike Tunstall as due to the execution time of the ECDSA operation the memory of the oscilloscopes available were too small. The same model of ARM7 microprocessor board as used previously was targeted again, however this time the sampling rate was reduced to $125\,\mathrm{MSs}^{-1}$, with the resultant traces containing $\approx 185\,\mathrm{M}$ points. In total 100 traces were recorded for testing the classification error. For building the templates, 4.5 k training traces of each of the unified point doubling and addition operations were recorded (*i.e.* $m = 9\,\mathrm{k}$), with these traces being $\approx 640\,\mathrm{k}$ points long. The traces were filtered as before, however to compress the traces the individual clock cycles had to be initially extracted using the inverse fast Fourier transform (FFT) as suggested in [65] before taking the maximum point to prevent de-synchronisation of the traces as explained in §2.10.3.

5.7.1 Attack Procedure

A sample trace of the entire ECDSA operation is given in Figure 5.8(a), while in Figure 5.8(b) the mean of the unified group operations in the training set is plotted. Due to the scale of the ECDSA trace it is difficult to discern any information from Figure 5.8(a) and it is simply provided for reference. It is worth pointing out that the end of the trace around $\sim 1.3\,\text{s}$ has a distinctly different power profile than the rest of the trace. This is where the hash computation and multiplication/inversion steps as given in Appendix B are performed. A quick sanity check can be performed by checking that the length of time the point operation takes roughly corresponds to the length of time for the ECDSA algorithm, *i.e.* $192 \times 4\,\text{ms} \times 1.5 = 1.15\,\text{s} \approx 1.3\,\text{s}$ where 192 is the bit length b, 4 ms is approximately the length of time of the unified point operation (as can be seen on the x-axis in Figure 5.8(b)), 1.5 assumes that half the bits are 1 (hence the group operation is performed twice), and 1.3s is approximately the length of time for the scalar multiplication section of the trace in Figure 5.8(a). This is only a rough approximation, however suffices for checking that the traces represent what is expected.



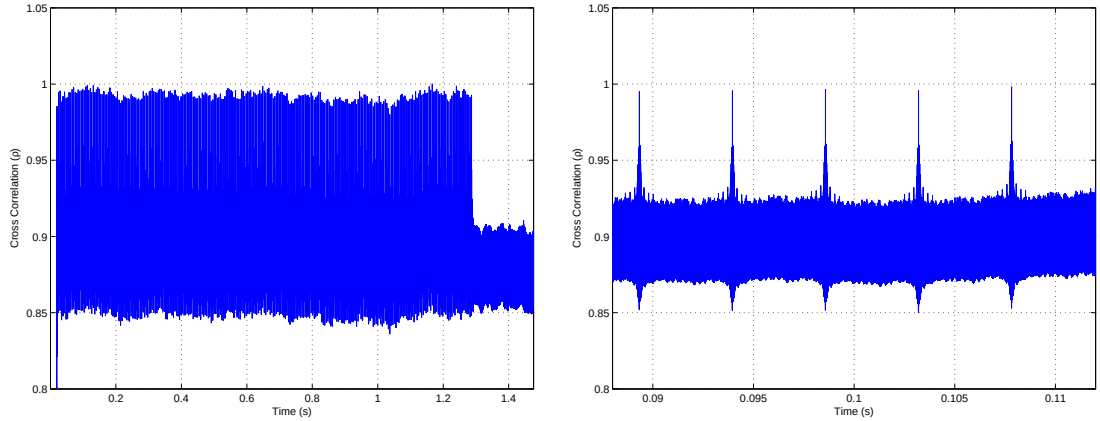
(a) Full ECDSA power trace.

(b) Mean unified elliptic curve point operation.

Figure 5.8: ECDSA target & training power traces.

To extract the group operations from the trace, the cross correlation between the ECDSA traces and the mean unified group operation as shown in Figure 5.8(b) was calculated for each of the 100 testing traces individually. A randomly selected cross correlation plot is shown in Figure 5.9(a). Note the cross correlation values were scaled such that the largest value is equal to 1. For the particular trace the correlation plot is shown for, the 192-bit key has a Hamming weight of 87 which, ignoring the MSB, gives 275 group operations, each of which corresponds to a single peak. A zoomed section of the trace is shown in

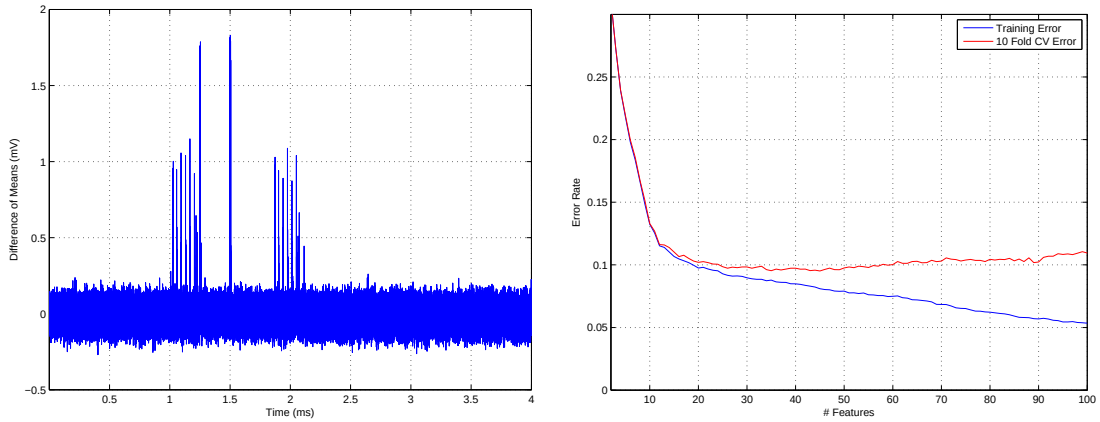
Figure 5.9(b) for clarity. No peaks occur after $\sim 1.3\,\mathrm{s}$ where the power consumption in Figure 5.8(a) is distinctly different, again validating what was expected. On a practical note, the group operations can be automatically extracted more reliably when the cross correlation is performed prior to trace reduction at the cost of a longer running time and greater memory requirements, hence extraction of the group operations from the ECDSA trace is recommended prior to trace compression. This likely holds true when extracting operations for other attacks also.



(a) Cross correlation trace.

(b) Zoomed cross correlation trace.

Figure 5.9: Unified group operation extraction.



(a) Unified group operation difference of means trace.

(b) Training & testing error rates as a function of retained features.

Figure 5.10: Template building of unified group operations.

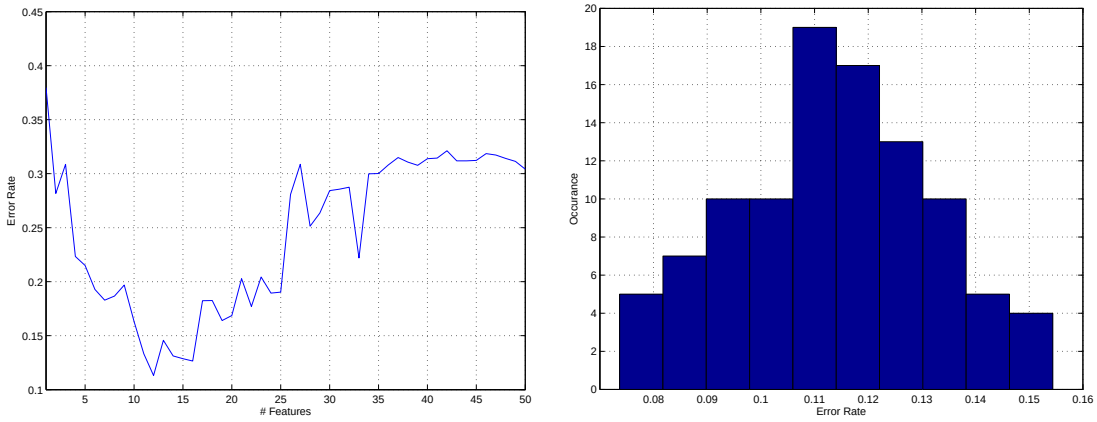
To build the templates, features were extracted as before by examining the difference of means trace of the unified operation which is plotted in Figure 5.10(a). The peaks again indicate the points in time where the Hamming weight of the output of a single-precision multiplication differs depending on if the inputs are equal. While the training set traces

represent point addition and point doubling traces, the underlying leakage is still that of field multiplications or field squarings. A different approach to selecting the number of features to build the templates with is used here, as the training and testing traces no longer directly correspond with each other like previously. Rather than estimating the error on the test set for an increasing number of features, 10-fold cross validation is used on the training set to select the number of features to use. The templates are then built across the entire training set for this number of features. This cross validation error is shown in Figure 5.10(b). The traces were normalized by taking the z-scores of the *individual* cross validation training sets *separately*, and QDA was used to classify the examples. It is clear from Figure 5.10(b) that when too many features are retained that the training error continues to decrease while the cross validation error increases hence the templates are over fitting the training data. An attacker would expect that retaining $30 \rightarrow 40$ features should give the best classification for these particular traces as this is what minimises the cross validation error.

When classifying the extracted group operations from the test ECDSA traces however, the mean error rate was found to be significantly higher at ≈ 0.3. This mean error rate was calculated by classifying all group operations from all 100 test traces individually and averaging, hence it is an operation error rather than a bit error. A plot of the error rate as a function of the number of features retained is given in Figure 5.11(a) where it is clear that using in the region of 12 features gives considerably better results than when using the 35 predicted by cross validation. A closer examination of this result reveals that the 12 features are all in two large peaks of Figure 5.10(a), where each peak is actually 6 distinct peaks. Using only these 12 features results in a mean error rate of 0.113 which is closely aligned to the expected cross validation test error for this number of features. The distribution of the mean error for each of the 100 individual ECDSA traces is given in Figure 5.11 to estimate how accurately an adversary could recover the entire scalar operand. For a 192-bit scalar multiplication, an attacker would expect to incorrectly classify $192 \times 1.5 \times 0.113 = 32.54$ point operations per trace. Hence to successfully forge an ECDSA signature, these mis-classified bits at unknown locations must be corrected somehow.

5.7.2 Hand-tuning the Results

The use of the test error in Figure 5.11(a) to select the number of features to use can be viewed as somewhat cheating as post attack information is required, *i.e.* an attacker



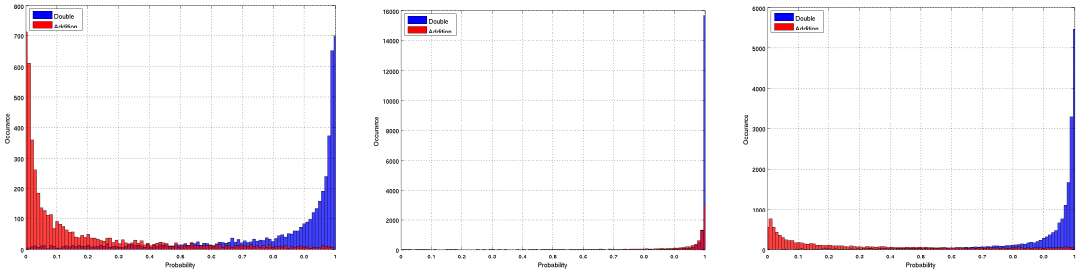
(a) Testing error as a function of retained features.

(b) Distribution of minimum test error.

Figure 5.11: Analysis of testing error rates.

won't have the correct secret to verify the results hence allowing the choice of the best parameters. However the knowledge that the ratio of point doublings to additions will be approximately 2 : 1 can be used to manually adapt the selection of training parameters according to the test probabilities. As an example, in Figure 5.12(a) the training traces were classified using the templates they built and a histogram of the probability of each trace being a doubling operation is shown. As it is a binary classification system, the closer to zero a probability is, the higher the chance of it being a point addition. As the number of point doubling and addition training traces are evenly distributed in the training set, there is a symmetry to the graph as the operations are largely classified correctly. In contrast, Figure 5.12(b) shows the classification distribution of the operations extracted from all the ECDSA trace group operations when 35 features are retained following the cross validation results. Here there is a very clear bias towards classifying doubling operations which indicates the templates are a poor fit and don't generaise well. In contrast Figure 5.12(c), where 12 features are used which is known to provide the lowest error rate, the distribution, although still biased towards classifying point doublings, is a much closer to the expected ratio of 2 : 1. It must be acknowledged that this is a somewhat *ad-hoc* method of refining the template parameters, however is likely to be applicable to many scenarios where the classifier can be hand tuned.

A closer examination of the error rates in Table 5.1 looks at the misclassification differences between point doublings and additions for both the training and testing traces, where 12 features are retained. Its interesting to note that while the errors are evening distributed for the training set, there is a large bias towards classifying the test operations as point doublings, hence reducing the point doubling error rate compared to the training set and



(a) Training distribution. (b) Testing distribution - 35 features. (c) Testing distribution - 12 features.

Figure 5.12: Distribution of point doubling & addition classification probabilities.

	Training	Testing
Doubling	0.1156	0.0473
Addition	0.1153	0.2441

Table 5.1: Comparison of point operation errors.

increasing the point addition error rate. This is a somewhat trickier problem for an attacker as while the traces under examination here have this effect, there is no reason to assume that a set from a different target platform (or even a set on the same target platform using a different group operation formula) will have a similar effect. Setting the decision boundary larger than 0.5 will rebalance the error rates, but without prior knowledge an attacker will simply be making random choices as to how to proceed. One scenario where this might apply could be where an attacker is looking to break many ECDSA signatures, each with a different ephemeral scalar operand. After performing an attack on the first trace where considerable effort might be required to recover the secret, this information could be used to further refine the key recovery stage. Iteratively updating the template model in a reinforcement type learning manner should lead to greater accuracy hence a faster correction stage, *i.e.* the greater the number of elliptic curve multiplicands recovered, the faster it is to recover them.

5.7.3 Correcting Scaler Multiplicand Bit Errors

As mentioned previously, the error rate can be corrected somewhat due to the fact that two consecutive addition operations cannot appear. HMMs [63, Ch. 3] were used to perform this correction as suggested in [86, 116], however a slightly different approach is taken here. Rather than the hidden states representing key bits as suggested in the previous work, the hidden states here represent doubling and addition operations directly. In [116] emissions

correspond to double or double-add, which is impractical for this analysis as it only allows correction of errors between those operations. A more realistic approach is presented in [86] where an extra variable is introduced to keep track of the number of operations analysed allowing for distinguishing errors between point doubling and point additions directly. This approach can be simplified here however due to the extra information available due to our template training data set. By classifying the training set itself, the emission probabilities of the HMM can be estimated, hence removing the extra complexity of the tracking variable required in [86].

To generate the emission probabilities of state 0, *i.e.* the doubling state, a template is first built for the doubling operation using the available traces. These doubling traces are then classified using that template, with each trace being assigned a probability that it is a doubling. The mean, and one minus the mean, of these probabilities are then used as the emission probabilities for that state. The same procedure is repeated for the emission probabilities of the addition state using the addition traces. Note that ideally, the traces used to generate the emission probabilities would be separate to those used to build the templates. A diagram of the HMM for this particular attack is given in Figure 5.13. These HMM parameters can be viewed as extra parameters to train in the profiling stage, with the emission probabilities dependent on the data under consideration. This slightly reduces the overall error rate from 0.1131 to 0.1084, with a similar distribution to Figure 5.11(b). Note that this method still does not make full use of the available information. The classification probabilities of the predicted test operations go unused and the integration of this information into the HMM could be an interesting topic to explore.

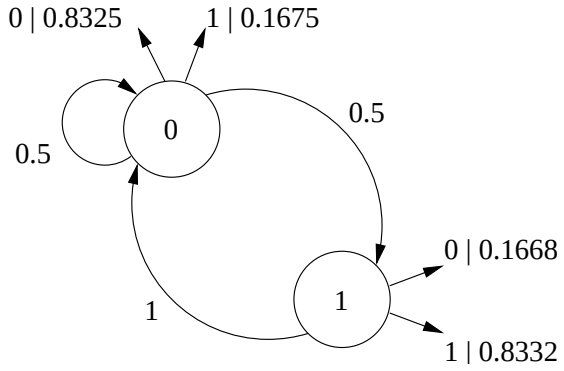


Figure 5.13: Hidden Markov model parameters.

To correct remaining key bits after the template attack, a few options are available. Where the base point and scalar multiplication output are known, a brute force search can be attempted. Note the output template probabilities give a natural search order to the

key bits. A lattice based attack was also proposed in [193] where if a few bits of the ephemeral nonce from the elliptic curve scalar multiplicands from several signatures was available, the secret key can be recovered. Lattice based attacks require correct nonce bits to work however. The work in [157] takes into account the probabilistic nature of templates using *Bleichenbacher's Solution* [31] when correcting ECDSA ephemeral scalar multiplicand bits.

5.8 Attack on Hardware Montgomery Multiplier

The attacks to date have been all been against software algorithms running on the ARM7 microprocessor. This is a suitable target device as it leaks, like many other microprocessors, a Hamming weight approximation of the values being manipulated. Hence it is worth exploring how the attack would fare against a hardware platform such as an FPGA where the leakage would approximately follow the Hamming distance model.

A 1024-bit version of the coarsely integrated operand scanning (CIOS) Montgomery multiplier [124] algorithm as given in Algorithm 5.8, was implemented with the architecture as given in Figure 5.14. The target device was the SASEBO-G evaluation board with a Xilinx Virtex-II FPGA, and 10 k each of the multiplication and squaring traces were taken from the circuit while running at 2 MHz, while recording the power consumption using a *PicoScope 3400* oscilloscope with a sampling rate of 250 MSs^{-1}. Algorithm 5.8 as implemented requires $w\,(2\,w+6)+2$ clock cycles to complete. The radix b is set according the to size of the single-precision multiplier, which is set to 32-bits in this case, and $w = \lceil 1024/b \rceil = 32$. While the choice of 32-bits is to allow for a more direct comparison with the software results from the ARM7 microprocessor, it must be noted that this is not an ideal choice for the underlying FPGA device. A dedicated 18-bit signed embedded multiplier block is available on the Virtex-II device [231], so for the choice of 32-bits cascaded multiplier blocks must be used. As no pipe-lining is present between the embedded multipliers this is a suboptimal use of resources, however it is adequate for the evaluation purposes here. The CIOS architecture was chosen from the available options in [124] as the additions could be scheduled to allow the use of a shift register rather than random access memory (RAM) as shown in Figure 5.14.

The traces were filtered using a 100-point bandpass FIR filter with a Blackman window and cut-off frequencies of 100 kHz and 9 MHz, and the traces were reduced to a single point per clock cycle by selecting the maximum point per clock. A sample multiplication

Algorithm 5.8: CIOS Montgomery product.

Input: $m = (m_{n-1}, \ldots, m_1, m_0)_b$, $x = (x_{n-1}, \ldots, x_1, x_0)_b$, $y = (y_{n-1}, \ldots, y_1, y_0)_b$ with $0 \leq x, y < m$, $R = b^n$, $\gcd(m, b) = 1$ and $m' = -m^{-1} \bmod b$.

Output: $A \leftarrow x\, y\, R^{-1} \bmod m$.

```
1  for i = 0 to n − 1 do
2      C ← 0 ;
3      for j = 0 to n − 1 do
4          (C, S) ← a_j + x_j y_i + C ;
5          a_j ← S ;
6      end
7      (C, S) ← a_n + C ;
8      a_n ← S ;
9      a_{n+1} ← C ;
10     C ← 0 ;
11     t ← a_0 m' mod R ;
12     (C, S) ← a_0 + t m_0 ;
13     for j = 1 to n − 1 do
14         (C, S) ← a_j + t m_j + C ;
15         a_{j−1} ← S ;
16     end
17     (C, S) ← a_n + C ;
18     a_{n−1} ← S ;
19     a_n ← a_{n+1} + C ;
20 end
21 return A
```

trace is given in Figure 5.15(a) for reference, and the difference of means for all traces is given in Figure 5.15(b).

Three distinct peaks are visible in the difference of means trace in Figure 5.15(b), two at the beginning and one at the very end. A delay was implemented between reading and writing data from the serial port, and execution of the multiplication to ensure that any leakage is from the multiplication itself rather than the communications which can have a significant effect on the power due to driving the loads of the pin-outs. Using the Xilinx simulation tools to determine exactly where the peaks occur, the first two peaks can be attributed to line 4 of Algorithm 5.8 when $i \equiv j \equiv 0$ and $i \equiv j \equiv 1$ respectively. The third peak can be attributed to the last loop of line 19 where the output of the second multiplication loop is input to the shift register. While this leakage point seems to be at

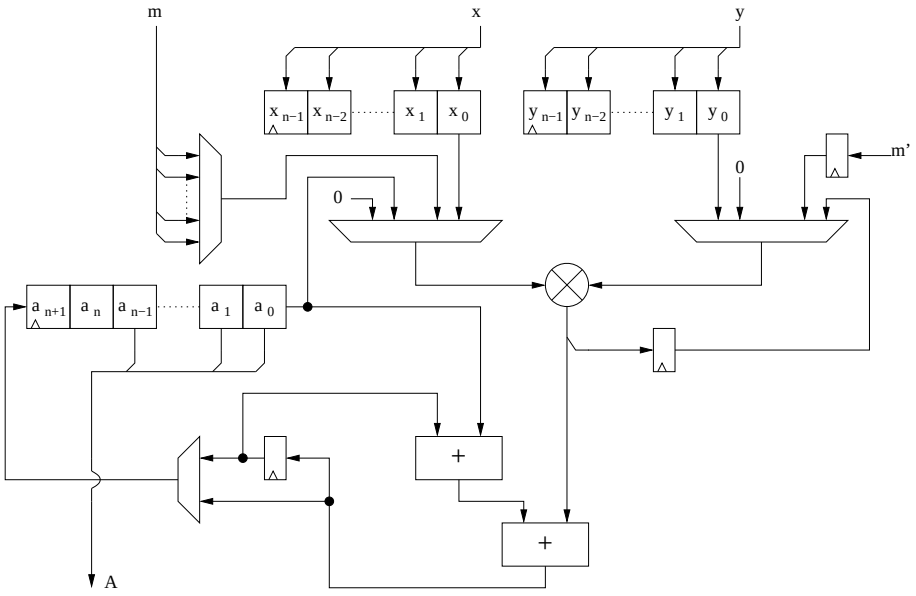
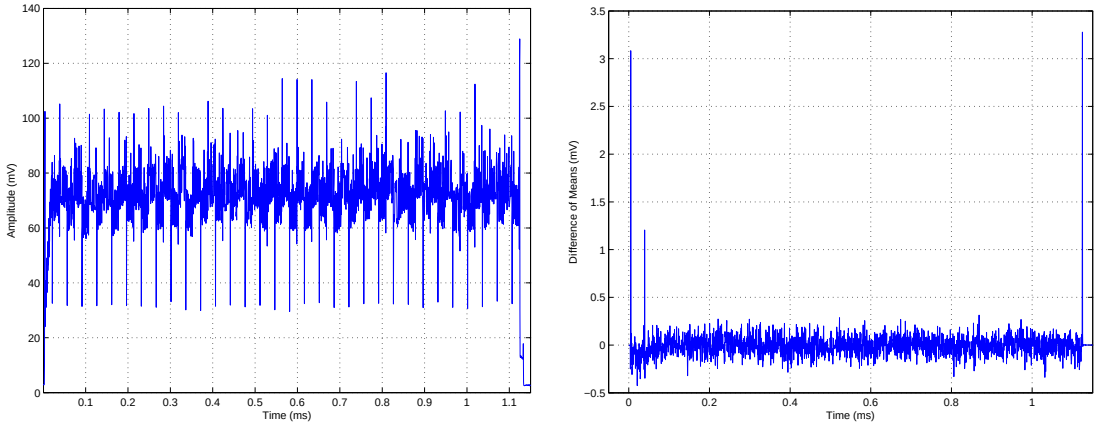


Figure 5.14: CIOS Montgomery product architecture.

odds with what is expected, closer examination reveals that the control signal for shift register a also controls shift register y, and this final multiplication step results in x_0, y_0 once again being routed though the embedded multiplier blocks although the output is not used. Unlike when targeting a software implementation, a peak is not present for for every time $i = j$ however. This is likely due to the fact that the output register a has to be cleared after each full multiplication for the correct accumulation of the single-precision multiplication outputs hence it is only at the start that the Hamming weight is leaked. While this does not fully explain the leakage of the final peak, it is likely to be something similar due to the resetting of the circuit states back to their original values. It is worth pointing out that the leakage is indeed due to the output Hamming weight of the multiplier and not the bit transitions *within* the multiplier unit as described in [4, 223], as if the latter was the reason for the leakage peaks would occur on line 4 of every loop iteration when $i = j$.

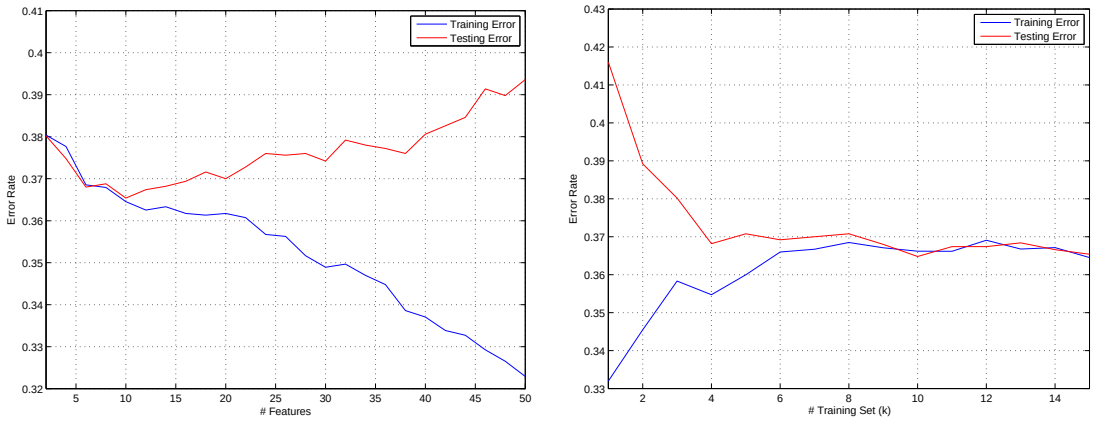
Templates were built with 15 k traces randomly selected for the training set out of the set of out of the 20 k, with the remaining 5 k allocated for testing. As before, the templates were built by taking the z-scores of the traces, using the difference of means to select points of interest. Cross validation is not used here as with the ECDSA attack, as the training and testing traces are drawn from the same set. The expected classification error when using QDA for this particular setup is given in Figure 5.16(a) as a function of the number of features retained. When compared to the results from the ARM7 microprocessor, the



(a) CIOS Montgomery multiplication trace. (b) Difference of means trace.

Figure 5.15: CIOS Montgomery multiplication architecture trace examples.

error rate is much higher at ≈ 0.37, and the number of features that minimises the error is much less due to the fewer number of peaks available to build the model. In Figure 5.16(b) the error is plotted as a function of the training set size to see if a larger training set would reduce the error rate however both the training and test errors are similar for a training set size greater than approximately 10 k, hence a larger training set will unlikely decrease the error.



(a) Error rate as a function of retained features. (b) Error rate as a function of training set size.

Figure 5.16: Error rate for CIOS Montgomery multiplication architecture traces.

Despite the relatively few leakage points due to the Hamming weight difference under consideration, templates can still be built to distinguish between multiplication and squaring operations, albeit with a larger error rate than when attacking software implementations. Hence in the context of key recover from an elliptic curve cryptosystem the attacker might require multiple acquisitions with the same scalar multiplicand to compensate. Other

options to improve the error rate could be to adopt a different strategy for feature extraction such as examining non-linear combinations of the data. The most effective method however would be to take a more targeted approach to building templates. Currently no knowledge of the underlying circuit architecture is assumed and the general approach used is favourable to attacks against software implementations, however if an adversary had access to source code or a bit file a more detailed template model could be built based on bit transitions or a custom power model.

5.9 Application to Symmetric Key Algorithms

While the nature of asymmetric algorithms such as RSA and ECC encryption is suited to being attacked by distinguishing multiplications and squarings, it can also be applied against symmetric algorithms where multiplications are required. While it might not be the most efficient method to recover a key for a given number of traces, there could be scenarios in which building templates based on intermediate values is not feasible, for example due to applied countermeasures.

Taking AES as an example, the *MixColumns* operation is in fact polynomial multiplication modulo an irreducible polynomial. This involves byte-wise multiplications, with the required operands being 2 and 3. If AES is implemented in this manner then the templates could be built to determine when the input to the *MixColumns* is one of these values which would allow key extraction in a known plaintext (or ciphertext) scenario. In an 8-bit device it is likely look-up tables would be used, however in [25] an efficient method to implement the *MixColumns* on 32-bit platforms using the multiplier is presented. As this involves 32-bit words a chosen plaintext adversarial model is likely more appropriate.

Likewise the block cipher International Data Encryption Algorithm (IDEA) [130] also requires multiplications as shown in Figure 5.17. The algorithm has a 64-bit state with 128-bit keys, where each data path of Figure 5.17 is 16-bits wide. A chosen plaintext attack where a squaring can be distinguished from a multiplication would trivially return sub-keys k_1, k_4 with at most 2^{16} inputs. The sub-keys k_3, k_5 can be jointly recovered by varying the input to the modular addition with k_3 while holding the other inputs constant. Once the value that causes a squaring with k_5 has been found a search of the $2^{16} \times 2^{16}$ possible sub-keys returns the valid pairs. Sub-keys k_2, k_6 can be recovered in a similar manner. Finally sub-keys k_7, k_8, which are used in the following round, can be trivially found once the other sub-keys are known.

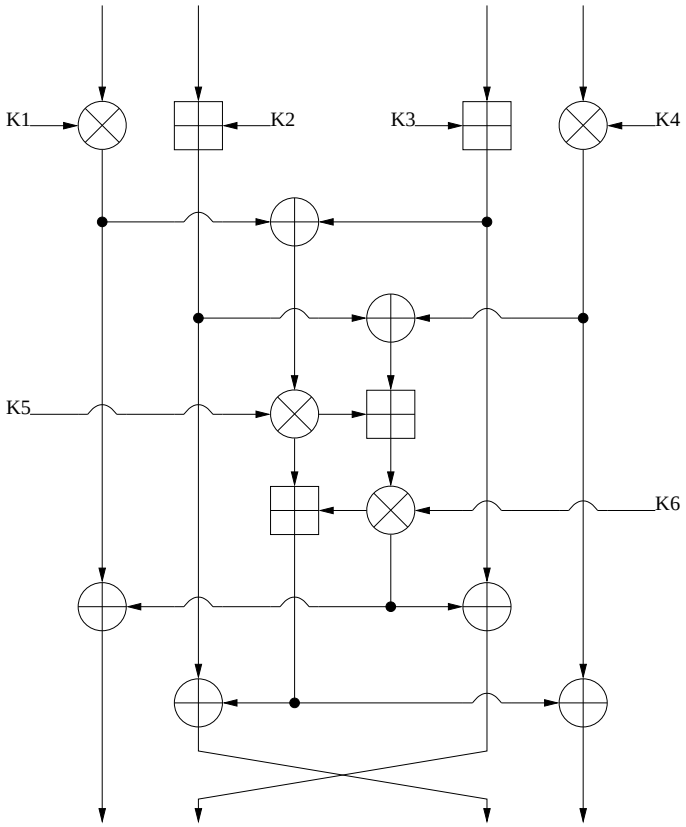


Figure 5.17: IDEA block cipher round function.

5.10 Countermeasures

The presented attack highlights the importance of using regular exponentiation algorithms such as the Montgomery Ladder [113], square only exponentiation for RSA [49] or add only scalar multiplication for ECC [110]. An in-depth look at regular exponentiation algorithms can be found in [110, 112]. It has been shown here that commonly used countermeasures for RSA or ECC cryptosystems such as randomisation of the message/base point, randomisation of the exponent/scalar multiplicand, side-channel atomic/unified algorithms or the restriction of exponent/scalar use to a singe case, can be easily defeated if templates can be built to distinguish between single-precision multiplication and squaring operations.

The authors in [4] also attack ECC by distinguishing between squarings and multiplications. While they propose a method to recover the scalar operand when the Montgomery ladder algorithm is used, the attack requires the use of a special elliptic curve point and the attack does not generalise. Many attacks exist against regular algorithms however, and all regular exponentiation algorithms were shown to be potentially vulnerable to collision

attacks in [90]. This highlights the difficulty of protecting devices against SCAs due to the varied adversarial threats.

A countermeasure proposed in [48] was to randomise the internal order of the single-precision multiplications in a long integer multiplication. For the Montgomery multiplication outlined in Algorithm 5.4, this would involve randomising the order that $x_i\, y$ was computed on line 4. This should have a relatively low computational overhead, however will make an attackers job considerably harder as he now has to determine if one single-precision multiplication per loop is a squaring. For the ARM7 microprocessor, this should provide a strong level of security as looking at Figure 5.7 the classification rate of a single multiplication isn't much better than a random guess.

5.11 Conclusion

In this chapter it is demonstrated that the principle of side channel atomicity is not valid simply because the same code is executed for a given function for all possible inputs. This was done by characterizing the difference in expected Hamming weight of the result of a multiplication and squaring operation given random uniformly distributed inputs to generate templates, based on previous descriptions of this difference in [10]. These templates are generated by considering single-precision multiplications rather than multiplications between multi-precision values.

These templates can be used to characterize multiplications and squaring operations to determine a private exponent when an RSA signature is generated using Algorithm 5.5. Previous work in this area [97, 148] has concentrated on building templates on intermediate values (*e.g.* such as observing where the input value is reused in a left-to-right exponentiation algorithm) rather than the expected distribution of the result of a given function.

An advantage of this work over previous work based on the template model originally proposed in [42] is that an attacker does not need an open identical device to conduct the attack as demonstrated in §5.5.3. That is, an attacker does not need a device where all the input and cryptographic keys can be changed to arbitrary values. An attacker can use an identical device with a known key, or a verification functions that uses the same operations. Furthermore, the values being operated on do not need to be known, an attacker just needs to know that the values are random.

The work also applies to elliptic curve scalar multiplication as shown in §5.7. If strongly

unified formulae such as those defined for Edward's [24] or Weierstraß curves [35] are used, then an algorithm equivalent to Algorithm 5.5 can be defined. While the bit length of the prime field using in ECC is significantly smaller than the bit length of an RSA modulus there are numerous operations in the prime field for each operation, *i.e.* the addition and doubling operation, that could be exploited.

The attack is similar to the *Big-Mac* attack [219] in that recovery of the secret is due to determining if an operation is a multiplication or squaring, however it trades a stronger adversarial model, *i.e.* an attacker can generate templates, for a more robust attack. *Horizontal CPA* [48] also looks to determine if an operation is a multiplication or squaring, however intermediate values are used to determine the path taken by the algorithm. The TA described here will also, counter-intuitively, become more effective with longer key lengths due to the extra single-precision multiplications required, as previously noted in [220] for similar reasons. This is also true of many of the other related attacks mentioned in this chapter. Overall this work gives a strong argument for using regular exponentiation algorithms such as found in [112], rather than, or in conjunction with, side channel atomic algorithms.

Chapter 6

Machine Learning Methods for Side-Channel Attacks

6.1 Introduction

While TAs are the main method of choice for profiled SCA, they are not the only one. SMs were introduced by Schindler *et al.* [195] as an alternative method to conduct profiling attacks based on linear regression. However both TAs (or discriminant analysis) and SMs assume a Gaussian distribution of the variables. While in the context of SCA this assumption is often valid, there can be scenarios where it is not so, or where it would be beneficial for an attacker not to be constrained by such assumptions. Hence supervised machine learning methods seem a suitable topic for application to profiled attacks.

There is a large body of research in the computer science domain on machine learning techniques which can be adapted for attacks, much of which pre-dates the topic of SCA itself. Machine learning as a topic is becoming ever more important in the age of *big data* to extract value from the slew of information being constantly generated.

One of the first references of machine learning in the context of cryptography was by Rivest [191], who explored how advances in cryptanalysis contributed to advances in machine learning, and vice versa. It was many years then before one of the first uses of supervised machine learning, as opposed to discriminant analysis, in the context of SCA was used in an acoustic attack against dot matrix printers [14][1]. Recently there has been a number of papers investigating the use of machine learning in the context of "classical" SCA,

[1] Cluster analysis was used prior to that in an unsupervised machine learning attack by Batina *et al.* [20]

such as SVMs [19, 96, 99, 103, 104, 135], random forests [135, 136] and self-organising maps [135]. Naïve Bayes [63] is a widely used probabilistic classifier which assumes multiple independent features. It is essentially equivalent to the *reduced* TA of [145, Ch. 5] where the off-diagonal elements of the covariance matrix are set to zero, thereby removing any correlation between the features. Another profiling attack was presented by Sugawara *et al.* [209] based on multivariate regression analysis. This is somewhat similar to the SM as it uses linear regression to model expected power consumption values, however in the classification step CPA (or any other non-profiling distinguisher) is used for key recovery. This was also suggested by Whitnall *et al.* in [226]. Gaussian mixture models (GMM) as previously used in [134] is another method for which closer examination could lead to interesting results. There has also been a number of papers investigating unsupervised methods for non-profiling attacks based on clustering [20, 100], as well as investigating clustering as a means to improve classical TA [137], however these unsupervised methods are outside the scope of this thesis which focuses on profiled attack methods. Although profiling attacks were originally suggested at the 1999 CHES workshop [70], the same year Kocher *et al.* first presented power analysis in the CRYPTO conference, it is only recently that the large body of literature on widely used machine learning algorithms have begun to be applied. Hence any general purpose book on statistical learning such as [30, 56, 63, 95, 106] will have relevant algorithms and techniques to improve classification performance.

To compare classification methods, the error rate of unseen testing data is used which is essentially the inverse of the success rate as outlined in §2.8. Although all algorithms are used such that they return probabilistic outputs allowing for amplified TAs, key recovery from a single trace is preferred hence the error rate is the main metric used to compare algorithms. A paper by Standaert *et al.* comparing TAs and SM [205], uses an entropy reduction matrix to compare the efficiency of the profiling stage, with the mutual information [80] directly derived from it. This idea was further expanded in [188] with the notion of *perceived* information which can be viewed as the mutual information where a different device is used to build templates, as it accounts for the difference in power consumption between devices. Perceived information is upper bounded by the mutual information as identical leakage between devices can be seen as the best case scenario for an attacker.

Two trace sets are used in the following, the first are the 160-bit multiplication traces from §5.5.3 implemented using a constant time 32-bit single precision multiplier. This is a binary classification problem where the goal is to distinguish between a multiplication and a squaring operation. A total of 10 k traces are allocated as training traces, and the error

rate is calculated over a separate set of 5 k testing traces. These 15 k traces are randomly selected for each experiment from a larger set of 40 k. The training traces are normalised by taking their z-scores, with the empirical mean and standard deviation then applied to the testing traces. Feature selection is always performed using the SOST method as outlined in §3.3.3, as due to the binary classification problem Fisher's linear discriminant can only return a single feature. Note also that the use of Fisher's linear discriminant will bias the features prior to non-parametric methods such as SVMs or NNs. Where the error curve is a function of the training set size, the z-score and feature extraction calculations are performed using the current number of training samples.

The second set of traces used are the AES traces from the ARM7 microprocessor that have been used extensively through this thesis. The target is always the value of first byte of the *S-Box* output in round one, giving a $\mathcal{K} = 256$ multi-class classification problem. Here 25 k traces are used for the training set, chosen at random from a set of 50 k for each experiment, with 5 k samples again used for error calculation, chosen from a different set of 50 k traces from the same device. Feature selection is also performed using the SOST method to ensure consistency with the binary classification results.

The first four sections of this chapter deal with *Stochastic Methods* §6.2, *Logistic Regression* §6.3, *Support Vector Machines* §6.4, and *Neural Networks* §6.5. An overview of each is given, with attacks against the two trace sets as described above evaluated for various parameter selections. A comparison between the classifiers is given in §6.6 using the parameters that minimise the error for each model. An overview of the importance of feature selection is given in §6.7, with conclusions drawn in §6.8.

6.2 Stochastic Methods

SM were first introduced by Schindler *et al.* [195] as an alternative profiling attack to TA. The stated goal of the approach aims to *achieve the efficiency of the template attacks in the key extraction phase but requiring far less measurements in the profiling phase* [195]. While works such as [81, 205, 226] have already discussed the relative merits of SM compared to templates, here different methods of building SM are explored, before comparing them to different machine learning techniques in §6.6.

SM allow an adversary to take advantage of any knowledge he may have about the target device during the profiling stage, however they can equally be generalised such that an adversary is assumed to have no prior device specific information. Following the description

in [205], given a set of leakage traces $x^{(i)}$ for $1 \leq i \leq m$, there will a some point in time where $x_t^{(i)}$ which corresponds to the target leakage function $L_t(\mathcal{F}(p^{(i)}, s))$ where $p^{(i)}$ is a known input corresponding to $x^{(i)}$, s is the secret we wish to recover and $L_t(\mathcal{F}())$ some leakage at a specific time corresponding to a function of both inputs, such as the *S-Box* function of AES. For illustrative purposes it is be assumed that recovering $y^{(i)} = \mathcal{F}(p^{(i)}, s)$ is equivalent to recovering s, hence the error is calculated for $y^{(i)} \in \mathcal{K}$. While this will not necessarily always be the case, the model can be easily adapted when it is not so.

The SM assumes that the leakage function L_t can be written as a deterministic and a random part, *i.e.* $L_t(y^{(i)}) = \delta_t(y^{(i)}) + \rho_t$. Both parts are then profiled separately to build a model for key recovery. The training samples are split into two (not necessarily equal) sets $\{\dot{x}, \ddot{x}\}$ such that $\dot{x} \cap \ddot{x}$ is empty, with m_1 samples in $\dot{x}$, m_2 samples in $\ddot{x}$, and $m = m_1 + m_2$.

6.2.1 Approximation of the Deterministic Part

Assuming that the deterministic part of the model can be computed as a linear sum, $\delta_t(y^{(i)}) = \sum_{j=0}^{u-1} \beta_{j,t} \cdot g_{j,t}(y^{(i)})$ for some base functions $g_{j,t}$, the coefficients β can be approximated using linear regression. The problem is then reduced to finding u base functions $g_{j,t}$ that allow an accurate approximation of δ_t.

In practice, the chosen base function is simply the bit values of the chosen target function. For example, if the target when attacking AES was the output byte of an *S-Box*, this would give $g_{j,t}(y^{(i)}) = \text{bit}(\ S(p^{(i)} \otimes s), j\ )$ for $1 \leq j \leq 8$. A bias term is required for the approximation of the β coefficients via linear regression therefore $g_{0,t}(y^{(i)}) = 1\ \forall\ i$, hence $u = 9$ in this particular scenario. This choice of base function, while simple, is generally quite effective as it has been shown that different bits will contribute varying amounts to the power consumption [5].

The matrix A is constructed as in Equation 6.1 for each of the m_1 training samples, allowing the β weighting values to be then calculated according to Equation 6.2. As $y^{(i)}$ can only take on $|\mathcal{K}|$ finite values, an approximation of δ_t for each possible value can be precomputed for use in both the approximation of the random part, as well as the classification stage using Equation 6.3, where G is similar to the matrix A, except constructed for the $|\mathcal{K}|$ distinct values of $y^{(i)}$.

$$A = \begin{pmatrix} g_{0,t}(y^{(1)}) & g_{1,t}(y^{(1)}) & \cdots & g_{u-1,t}(y^{(1)}) \\ g_{0,t}(y^{(2)}) & g_{1,t}(y^{(2)}) & \cdots & g_{u-1,t}(y^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ g_{0,t}(y^{(m_1)}) & g_{1,t}(y^{(m_1)}) & \cdots & g_{u-1,t}(y^{(m_1)}) \end{pmatrix} \tag{6.1}$$

$$\mathbf{b}_t = [\beta_{0,t}, \beta_{1,t}, \ldots, \beta_{u-1,t}] = (A' \times A)^{-1} \times A' \times \dot{x} \tag{6.2}$$

$$\hat{\Delta}_t = [\hat{\delta}_{0,t}, \hat{\delta}_{1,t}, \ldots, \hat{\delta}_{|\mathcal{K}|-1,t}] = G \times \mathbf{b}_t \tag{6.3}$$

6.2.2 Approximation of the Random Part

As outlined in [195], approximation of the random part of the traces is not strictly necessary, however in practice a lower classification error is achieved when it is. For each of the m_2 traces, the $\hat{\delta}_{i,t}$ corresponding to $y^{(i)}$ is subtracted to leave a noise vector $r^{(i)} = \ddot{x}^{(i)} - \hat{\delta}_{i,t}$. An empirical covariance matrix $\hat{\Sigma}_t$, can then be calculated using these noise vectors. This is similar to the calculation of the single noise covariance matrix calculation for LDA, §3.3.6 (as opposed to the classical QDA which has a noise covariance matrix for each class). Rather than subtracting the empirical mean vector $\hat{\mu}^{(i)}$, the deterministic section of a trace $\hat{\delta}_{i,t}$ is used instead. It is worth pointing out that using a separate set of traces to model the noise component would likely be beneficial for regular TA also. However in the case of QDA a substantial penalty on the number of training traces would be required.

There are no set rules as to the ratio to split the training set m into m_1 and m_2. In previous work, a split [195, 81, 134, 205] of 0.5 is commonly used. The optimum split is wholly dependent on the target device and acquisition setup, with particularly low noise devices maybe benefiting from a larger number of traces allocated towards estimating the deterministic trace parts. In the experiments that follow, all traces are allocated evenly and at random between m_1 and m_2, and any plot indices representing the number of traces used for training represent the the number of traces used for *both* m_1 and m_2. The random allocation of traces from across m rather than just the first or second half is also important to prevent any potential biases adversely affecting the results.

6.2.3 Classification using Stochastic Models

The classification stage differs depending on if the random part is modelled or not. Where it is not available, the attack is known as *Min*, and the minimum distance between the attack trace $x^{(i)}$ and the various $\hat{\delta}_{i,t}$ indicates the most likely intermediate value. It was shown in the original paper [195] that this is not as efficient as the *Max* method which includes the noise covariance estimation, however for completeness is is included here.

The likelihood of an attack trace given a set of learned stochastic models, including the noise covariance, can be calculated using Equation 6.4, which again is quite similar to classical TAs. Indeed from an implementation perspective, the main difference between a TA and using the SM is in the generation of the mean vectors $\hat{\mu}^{(i)}$ and the deterministic trace parts $\hat{\delta}_{i,t}$. Once the likelihoods for all models is computed the maximum value is taken as most likely, or if probabilities are required then Bayes' theorem, Equation 3.4, can be used. Likewise for an amplified attack where more than one trace is available for key extraction, Bayes' can be applied iteratively as in Equation 3.6. It is also worth noting that in the case of a *Min* attack, likelihoods can be calculated by taking the identity matrix, I, in place of the noise covariance matrix, $\hat{\Sigma}_t$, allowing the same equation to be used.

$$\mathcal{L}\left(x^{(i)} \mid \hat{\delta}_{i,t}, \hat{\Sigma}_t\right) = \frac{1}{\sqrt{(2\pi)^n \left|\hat{\Sigma}_t\right|}} e^{-\frac{1}{2}\left(x^{(i)}-\hat{\delta}_{i,t}\right)\hat{\Sigma}_t^{-1}\left(x^{(i)}-\hat{\delta}_{i,t}\right)^\top} \tag{6.4}$$

6.2.4 One-Hot Encoding

The choice of the base function $g_{j,t}()$ greatly affects the outcome of a Stochastic based attack. While the selection of intermediate bit values as explained previously is widely used, it is not the only method. It was also suggested in [195] that a Hamming weight based model could be used by appending the Hamming weight of the intermediate value to the bias term as shown in Equation 6.5 to give a two-dimensional basis. From here it is a straightforward step to simply append the intermediate value itself rather than the Hamming weight as in Equation 6.6.

$$A = \begin{pmatrix} 1 & HW(y^{(1)}) \\ 1 & HW(y^{(2)}) \\ \dots & \dots \\ 1 & HW(y^{(m)}) \end{pmatrix} \quad (6.5) \qquad A = \begin{pmatrix} 1 & y^{(1)} \\ 1 & y^{(2)} \\ \dots & \dots \\ 1 & y^{(n)} \end{pmatrix} \quad (6.6)$$

Another option, which has not been previously examined in the literature, is to encode the intermediate target value using *one-hot* encoding, for example as in Equation 6.7, where only a single column of each row besides the bias term contains a 1. An advantage of this is that each β parameter now represents a different intermediate value as opposed to a part of a weight sum. This allows a more accurate representation, as power consumption due to bijections of the target value (such as the *S-Box* input in the case of targeting the AES *S-Box* output) are now also considered. A disadvantage of this encoding however, is the extra β parameters that need to be estimated, which is now of the order of the number of unique classes, rather than $\log_2$ of the number, hence a greater number of traces is required for the profiling stage.

$$A = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 0 & \cdots & 1 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & 0 \\ 1 & 0 & 1 & \cdots & 0 & 0 \\ 1 & 1 & 0 & \cdots & 0 & 0 \end{pmatrix} \quad (6.7)$$

Using the AES traces acquired from the ARM7 microprocessor as described in the introduction, a profiling attack using the SM was conducted for various base vectors. Note Hamming weight based models were not considered as the goal is to recover the key with only a single trace.

Examining Figure 6.1, training errors are given by the dashed lines, with the testing error given by the solid lines. The plotting of the training error in learning curves is useful as the relationship between the error curves can help to distinguish bias or variance in the model. It is clear from the graph that using one-hot encoding provides the lowest error rate, keeping in mind that a large number of training samples were used ($m = 25\,\text{k}$ as specified in §6.1). It is interesting to note however, that when only using a small number of features, the *Min* and *Max* attacks give somewhat similar results, likely due to the fact that all 25 k traces are used for estimating β in the *Min* attack while in the *Max* attack

they are split between estimating β and Σ. Simply appending the target values in the case of the identity model gives poor results as is expected.

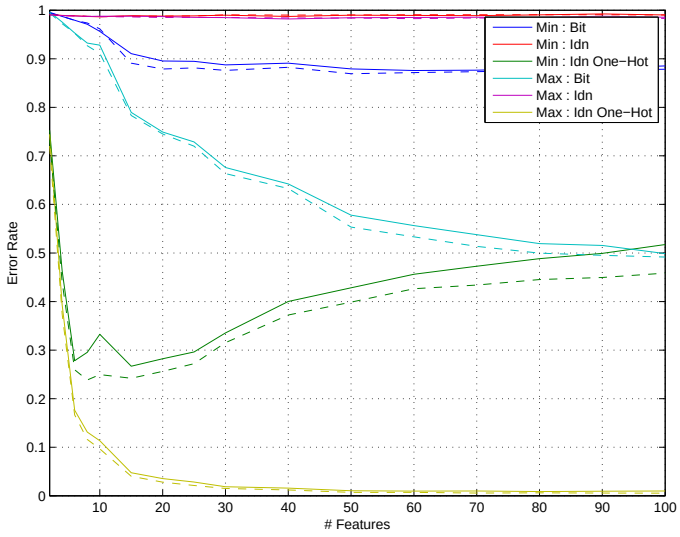


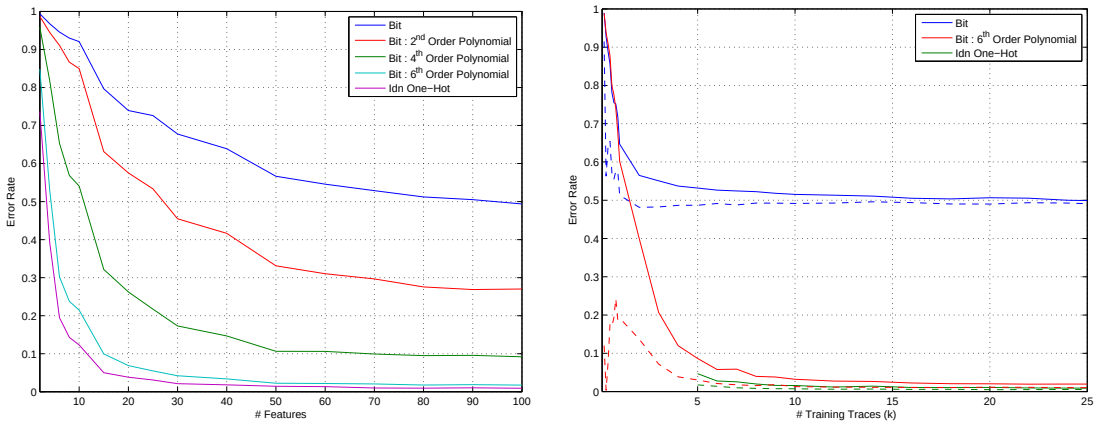
Figure 6.1: Error rate as a function of retained features.

6.2.5 Polynomial Basis Functions

Another option for the basis function is to explore polynomial combinations of the bits, as suggested by Whitnall and Oswald [226]. A related, lower order approach was also examined by Gierlichs *et al.* in [81] where they utilised a 17-point basis function to include *S-Box* input leakage by incorporating the bits of the input and the output. Here, every combination of bits is calculated hence contributions due to the square or cube of bits are calculated, unlike in [226]. This leads to larger computational complexities for higher orders however as the expansion has $u = \binom{b+\ell}{\ell}$ terms where ℓ is the expansion order and b the bit-length of the target value. For example for a $\ell = 6^{th}$ order polynomial, there are $u = \binom{8+6}{6} = 3003$ base function components. Practically, this matrix needs to be inverted, and the exponential nature of the expansion means that larger values become infeasible.

Looking at Figure 6.2(a), it is clear that by increasing the polynomial order that the error rate can be significantly reduced compared to the base function based on the bit values, and is comparable to the identity model using the one-hot encoding. Only the *Max* method using the bit and one-hot basis functions are retained from Figure 6.1, as they are they original, and give the lowest error rate respectively. The training error is omitted in Figure 6.2(a) for clarity. Fixing the number of retained features to 100 to minimise the testing error (note in a *real-world* attack cross-validation would need to be used to

determine the number of features to retain), the effect of the training set size is examined in Figure 6.2(b), with the training error again represented by the dashed lines. The use of a 6^{th} order polynomial gives comparable performance to that of the one-hot method, however with the advantage that profiling is still possible for a low number of training samples, unlike the one-hot method which requires at least 4 traces per intermediate value (2 each for the deterministic and random parts).



(a) Polynomial expansion of basis function.

(b) Error rate as a function of training samples.

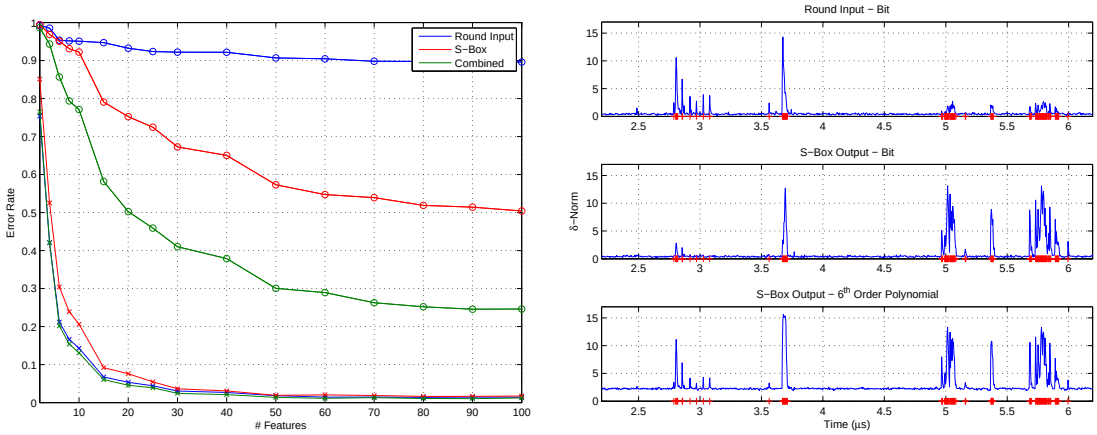
Figure 6.2: Error rate of polynomial expansion.

6.2.6 Higher-Order Stochastic Attack

To see exactly why polynomial expansion of the basis function gives improved results, higher-order attacks are examined. Figure 6.3(a) gives the error rate when attacking the both the *S-Box* input and output, as well as combining the probabilities of both profiling attacks to give a higher-order attack. The lines with circles here represent the original base function based on bit values, while the lines with the crosses are where a 6^{th} order polynomial is used. It can be seen that the higher order attack greatly reduces the error rate for the *Bit* model, while it only slightly improves the polynomial expansion results.

As shown in [117], plotting the β characteristics can reveal interesting information about where the trace leakage occurs. Here, instead of the β values, a plot of the sum across $\hat{\Delta}$ is provided instead in Figure 6.3(b). No feature reduction method is used here prior to model building to give an idea of the β weighting values in the time dimension. This then gives the expected trace representation as modelled by the SM. For reference, the red crosses along the base of the graphs represent the first 100 features as would have been selected by using the SOST method. The top two graphs of Figure 6.3(b) have different

peaks referring the input data (upper) and the *MixColumns* operations (middle). They share a common peak around where the *S-Box* operation occurs, but closer examination shows that the large peaks are slightly offset as they represent the *S-Box* input and output respectively. In contrast in the lower plot from the high-order polynomial basis, all the peaks are present, including a wider peak about the *S-Box*. Hence the error rate is lower due to the greater utilisation of the leakage data.



(a) Error rate as a function of retained features.

(b) Weighting values with sost selected points.

Figure 6.3: Error rate for higher order stochastic attack.

Overall, if conducting a SM attack and a large training set is available, then building the basis functions using the *one-hot* method will give the best results, however as a larger training set is required this negates the main advantage of the SM, *i.e.* the small profiling base. For smaller set sizes, setting the basis functions as polynomial expansions of the bit components of an intermediate value should give a significant improvement in the classification over simply taking the bit-values.

6.3 Logistic Regression

LR is a widely used binary classification algorithm [95, 158] that can also be use to conduct side-channel attacks. It is similar in ways to the linear regression based SM attack as presented in the previous section. In a stochastic attack the weighting parameters, β, are used to determine an empirically modelled trace, $\hat{\delta}$, for some hypothetical intermediate value in an attack. This is then subtracted from the actual attack trace to leave a noise vector, and the multivariate Gaussian distribution is used to determine the likelihood of that intermediate value occuring, given the target trace. In contrast, in LR the weighting

values, θ, are applied directly to the selected features of the attack trace to give a hypothesis of between 0 and 1, with 0.5 generally being the decision boundary between the two classes.

To perform classification using LR, it is required to learn weights for each feature (*i.e.* selected trace points) such that $0 \leq h_\theta(x) \leq 1$. This can be achieved with the use of the *Sigmoid* function as shown in Equation 6.8.

$$h_\theta(\mathbf{x}) = g(\theta^\top x) = \frac{1}{1 + e^{\theta^\top x}} \tag{6.8}$$

A graphical representation of the sigmoid function is given in Figure 6.4, and it shows that the larger the negative (resp. postive) input is, then the asymptotically closer to 0 (resp. 1) the function outputs. Therefore if we train our θ values such that negative numbers represent class 0, and postive numbers class 1 then we have a bounded output which also gives us the probability that the input trace x belongs to class 1.

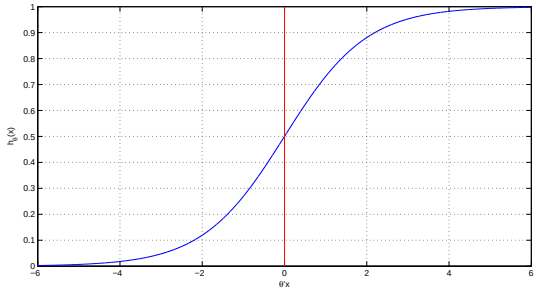


Figure 6.4: Sigmoid function.

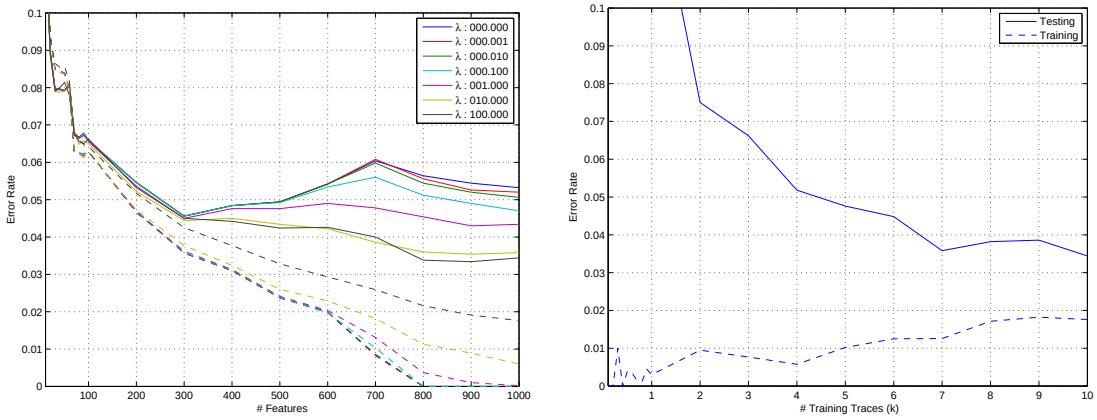
Following the tutorial as provided in [158], to learn the θ parameters, the cost function as given in Equation 6.9 must be minimised. This is a convex minimisation problem and can be solved using normal equations, similar to the SM in Equation 6.2, or through methods such as gradient descent. The use of normal equations becomes problematic for larger training sets due to the requirement of a matrix inversion of the order of the number of features. However, packages such as MATLAB and GNU OCTAVE contain efficient numerical minimisation algorithms that are often faster than gradient descent, especially for training sets with a large number of samples and/or features. In the work here, an slightly modified version of the gradient descent minimisation algorithm provided by the tutorials in [158] is used. Plotting the cost as function of the number of iterations in the minimisation algorithm gives an efficient way of viewing the convergence of the algorithm.

$$J(\theta) = -\left[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2 \tag{6.9}$$

The number of training samples is represented by m, $x^{(i)}$ is a training feature vector, and $y^{(i)}$ the corresponding class label. A regularisation parameter λ is also included in the cost function to prevent over-fitting of the model, as otherwise the minimisation algorithm could lead to a result that perfectly fits the training samples but generalises poorly to the population sample as a whole.

6.3.1 Attack on Multiplication

As LR is a binary classifier, it is first applied to the 160-bit multiplication & squaring traces from §5.5.3, using 10 k training traces with feature selection as outlined in the introduction. Both the training (dashed line) and testing (solid line) error rate for an increasing number of features is given in Figure 6.5(a). It can be seen that larger values of λ not only lead to a lower testing error, but also increase the training error hence preventing the selection of θ values which cause over-fitting of the model.



(a) LR lambda comparison.

(b) LR training set size comparison.

Figure 6.5: LR attack on 160-bit multiplications.

The error rate as a function of the number of training samples is also given in Figure 6.5(b), with 1 k features and $\lambda = 100$ used to train the classifier. It can be seen that the testing error is still falling as the training set size is increased, hence the error could likely be further decreased with the addition of extra training traces. The increase in the training error again shows that the model is beginning to fit the model as a whole rather than the subset of the training samples.

To analyse *why* the LR based attack works, the training stage is again run using 10 k training traces, except this time without any feature selection pre-processing step. The z-scores of the traces are still taken however, as this allows a faster convergence of the gradient descent algorithm. The learned θ values are shown in the lower plot of Figure 6.6, with the SOST selection trace given for comparison in the upper plot. Note the largest peak is truncated for clarity in the SOST plot. It can be seen that LR largely assigns weights according to the SOST peaks, except for the initial smaller peaks which are due to the re-use of y_0 in line 3 of Algorithm 3. When classifying using LR, the weighted sum of θ with the target trace should only be greater than 0 prior to the application of the sigmoid function when the target class is multiplication operation. This is because a multiplication is expected on average to consume more power that a squaring, hence the negative weights at the points of interest will cause the sum of a squaring trace to add up to less than 0.

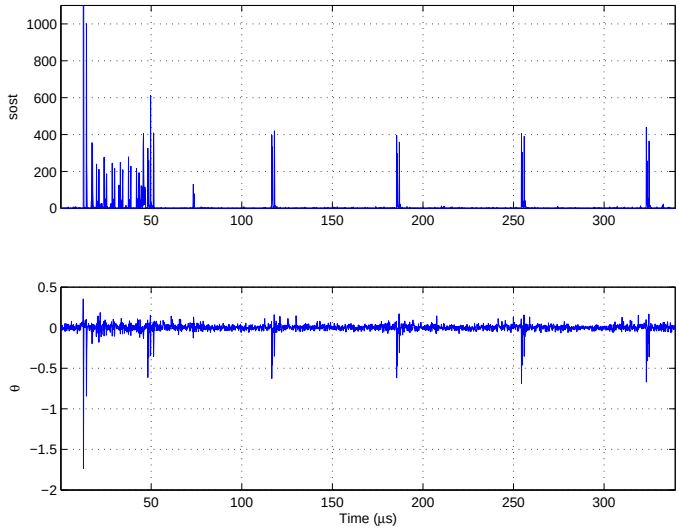


Figure 6.6: LR weighting values.

6.3.2 Attack on AES

To utilise LR classifier or any other binary classifier, such as SVMs, to recover a key byte from AES, for example, an extension to a multi-class classifier is required. Two widely used methods utilised in machine learning literature are *One-v-All* and *One-v-One* [106]. These are used here along with a bit-wise model as now explained in the context of attacking a byte-sized intermediate value:

Bit: Each target key byte can be simply decomposed into its binary format, and models

built individually for these bits. The probability of each byte is subsequently calculated through the product of its individual bits. This gives a total of 8 models to be trained for byte recovery, and all of the available traces are used for each model.

One-v-All: In this method, a model is trained for each byte individually, with each trace belonging to that byte assigned to one class and all other training traces assigned to the second class. All traces are again used for the training of each model, and 256 models in total need to be trained. It is worth keeping in mind that the class sizes are no longer symmetric and, assuming uniformly random plaintext inputs, the ratio of traces in each class is expected to be $\frac{1}{256}$. Where the training set is large enough to accurately model the θ parameters for the smaller class this is not an issue, however for smaller training set sizes it could introduce a skew to the classifier. The correct byte is then simply chosen as the one that gives the largest probability.

One-v-One: Depending on the data set size and number of classes, this can be quite a computationally intensive method to train as it requires training a classifier for all pairs of classes. In the case of classifying an *S-Box* output, $\binom{256}{2} = 32640$ separate models are required. However, as only two bytes are considered for each model, only $\frac{2}{256}$ of the training traces are used for each model. To perform the classification the target trace is classified using every model, with the byte that is assigned most likely assumed to be the correct one.

It must be noted that these are not the only binary to multi-class options available, and the nature of power analysis is such that algorithms can be tailored for the problem at hand. For example a binary tree based method could utilise a divide-and-conquer approach for key recovery. This is the approach followed in [19], however it requires a strict order in terms of the power consumption of each class such as the Hamming weight model.

The three methods listed above were used to classify both the AES traces, with the error rate as a function of the number of retained features given in Figure 6.7. For this initial analysis, 25 k training traces are used and $\lambda = 0$. It can be seen that the computationally intensive *One-v-One* method outperforms the other methods with regards to byte-wise classification, however the *One-v-All* method also performs relatively well. It is interesting to see that the *Bit* method, while the most straightforward and quickest to implement, performs significantly worse. This is somewhat expected as only the leakage of a bit is being utilised for training a classifier rather than the leakage of a byte. Each bit also leaks different amounts, this is shown in greater detail in Appendix C.

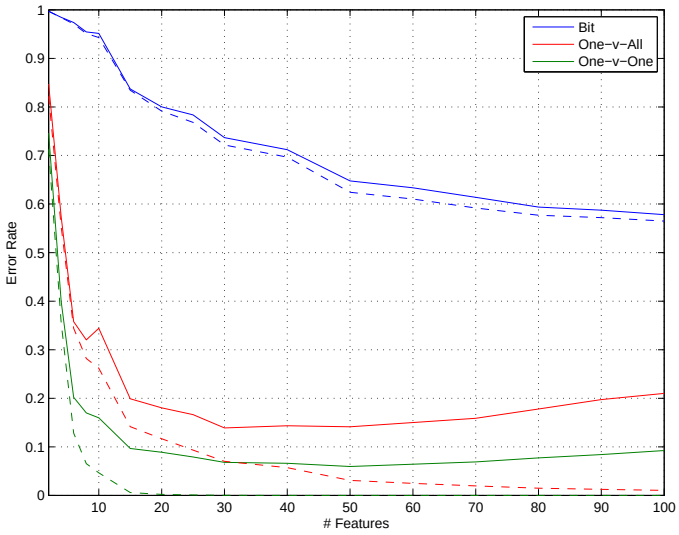
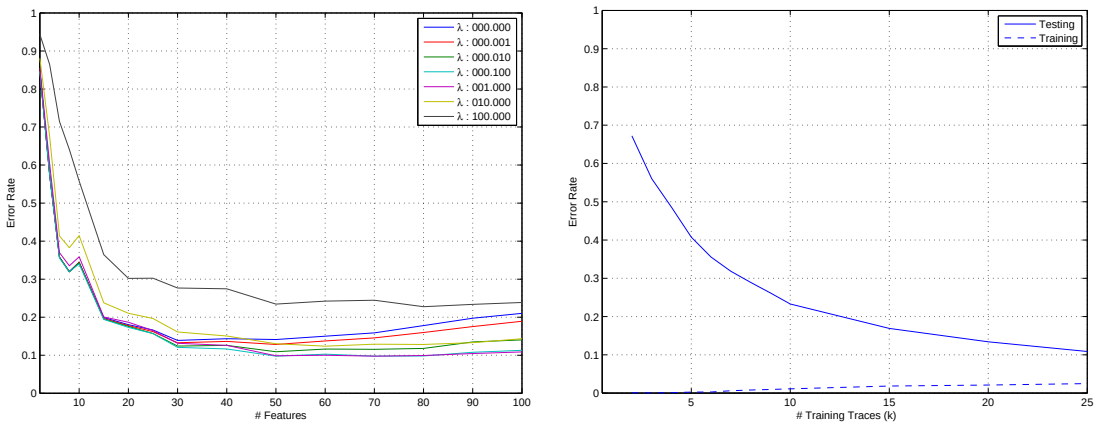


Figure 6.7: LR multi-class comparison.

Due to the computational overhead for the *One-v-One* multi-class method for a $\mathcal{K} = 256$ problem, the *One-v-All* method is used in future experiments where applicable. In Figure 6.8(a), the effect of λ is examined and in contrast to the binary classification with the 160-bit multiplication traces, larger values of λ give a higher error. Taking $\lambda = 1$ and 100 features, the effect of the training set size is explored in Figure 6.8(b). The error rate is asymptotically decreasing as the number of training samples is increased with a larger set size likely to further decrease the error rate, albeit by a small amount as the curve is flattening out.



(a) LR lambda comparison.

(b) LR training set size comparison.

Figure 6.8: LR attack on AES.

When presented with a set of traces to perform supervised learning on, applying LR as a first test is worthwhile, particulary if it is a large dataset with many training samples and features, as it is quick to run and can give results comparable to more advanced algorithms. Even where the results aren't as accurate, it can give an indication as to what to try next. The disadvantage of course is the lack of a native multiclass classification method, with the extension choice directly affecting the error rate.

6.4 Support Vector Machines

As previously mentioned SVMs [54], have recently been proposed for use in profiling attacks [104, 135], with further subsequent studies in [19, 96, 99, 103]. SVMs offer an interesting alternative to TAs due to their non-parametric approach, thereby removing the assumption of Gaussian distributed data. While power traces from many different devices have been shown to be inherently Gaussian in nature, this could allow for some interesting artificial feature constructions, such as that in [96] where the authors include the plaintext bits when attacking an XOR operation. Work to date has largely focused on the Hamming weight leakage of software AES implementations running on basic microcontrollers. An exception to this is the work by Lerman *et al.* [135] which looks at an FPGA implementation of DES. They build models on the key-bits directly, the fact that the key expansion in DES is simply bit-shifting means that key leakage occurs across the full length of the encryption trace. Previous work [19, 99] has also shown that SVM based profiling attacks can allow model building with a smaller number of training traces. As before, we are interested in the error rate for a key byte given only a single trace, hence models based on the Hamming weight leakage are not examined here.

Like LR, SVMs are a binary classification algorithm without any natural extension to multi-class classification. Two assumptions underpinning the algorithm are that transforming data into a higher dimensional feature space can convert complex decision boundaries to simpler, linear boundaries, and that the training samples closest to the decision boundaries provide the most useful information for classification.

Following the explanation as given in [212], the SVM training process looks the maximise the margin between the two classes as illustrated in Figure 6.9, where $\mathbf{w}$ is the normal to the hyperplane $\mathcal{H}$ given by the solid line, $\frac{b}{||\mathbf{w}||}$ is the perpendicular distance to the origin, and $\frac{2}{||\mathbf{w}||}$ is the distance between the hyperplanes $\mathcal{H}_{-1}$ and $\mathcal{H}_1$, which are represented by the dashed lines. To ensure the model generalises well, the hyperplane is chosen such that

it maximally separates the two classes. The vector $x^{(i)}$ that represents $\mathcal{H}$ is that which sets Equation 6.10 to 0, with the hyperplanes $\mathcal{H}_{-1}$ and $\mathcal{H}_1$ given by $f(x^{(i)}) = \{-1, 1\}$ respectively.

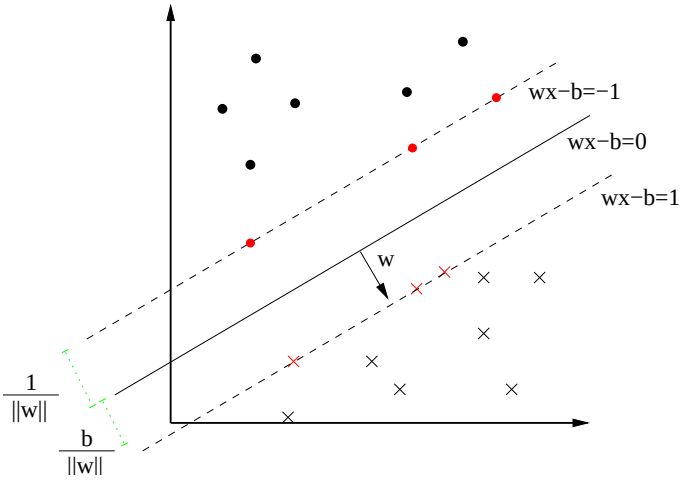


Figure 6.9: SVM decision boundary.

$$f(x^{(i)}) = \mathbf{w} \cdot x^{(i)} - b \tag{6.10}$$

Given a set of m training samples, in order for them to be correctly classified, each trace must satisfy Equation 6.11, however as we are also seeking to maximise the distance between $\mathcal{H}_{-1}$ and $\mathcal{H}_1$, therefore $\frac{2}{||\mathbf{w}||}$ must be as large as possible. Hence, finding $\mathcal{H}$ can be seen as the problem of minimising $\frac{1}{2}||\mathbf{w}||^2$ subject to Equation 6.11. This original *primal* optimisation problem and can be solved using the Lagrange function given in Equation 6.12, where the Lagrange multipliers satisfy $\alpha_i \geq 0$ for all i.

$$y^{(i)}\, f(x^{(i)}) \geq 1, \quad \forall i \tag{6.11}$$

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{||\mathbf{w}||^2}{2} - \sum_{i=1}^{m} \alpha_i \left[y^{(i)}\, f(x^{(i)}) - 1 \right] \tag{6.12}$$

Consequently, this leads to $\sum_{i=1}^{m} \alpha_i\, y^{(i)} = 0$, and $\mathbf{w} = \sum_{i=1}^{m} \alpha_i\, y^{(i)}\, x^{(i)}$, which can subsequently be substituted back into Equation 6.12 leading to the *dual optimisation* problem as given in Equation 6.13. This is a convex quadratic programming (QP) problem for which there are efficient solutions. A through explanation of the SVM algorithm is given in textbooks such as [56, 95]. Note the use of a dot product $x^{(i)} \cdot x^{(j)}$ in Equation 6.13 which is important for the use of *kernels* as explained below.

$$\text{maximize} \quad \sum_{i=1}^{m} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \, \alpha_i \, y^{(i)} \, y^{(j)} \, (x^{(i)} \cdot x^{(j)})$$
$$\text{s.t.} \begin{cases} \displaystyle\sum_{i=1}^{m} \alpha_i \, y^{(i)} = 0 \\ \alpha_i \geq 0 \quad \forall i \end{cases} \tag{6.13}$$

When the weights α_i and bias b are calculated, classification is performed by taking the sign of Equation 6.14 when applied to a test vector, where $m_s \leq m$ is the number of support vectors. Note that the SVM *only* returns what class a sample belongs to, it doesn't return any confidence measure of how correct it is, however $f(x^{(i)})$ does return the distance to the hyperplane which can be used to determine a probabilistic output as explained below. Note that the two classes are given by $y^{(i)} \in \{-1, 1\}$ as opposed to the more usual $\{0, 1\}$, and that only the samples denoted support vectors, as indicted in red in Figure 6.9, are required for the classification. Where a large percentage of the input samples are support vectors, it can be a sign of over-fitting to the training set, or alternatively that the classes are poorly separable.

$$f(x^{(i)}) = \sum_{j=1}^{m_s} \alpha_j \, y^{(j)} \, (x^{(i)} \cdot x^{(j)}) - b \tag{6.14}$$

Soft Margins: Due to noise on the training traces, a separating hyperplane is unlikely to exist, or where a kernel as described below is used, the hyperplane is unlikely to generalise well and over-fit the training data if it is found. Hence *slack variables*, ξ_i, are introduced for each training sample to allow its misclassification if it increases the overall accuracy of the model. This modifies Equation 6.11 to that in Equation 6.15. Trivial solutions exist where the ξ_i are made arbitrarily large, hence a regularisation parameter C is introduced such that there is a *Cost* associated with each slack variable. Note this performs a similar function as λ in LR, except it is now inverted such that as $C \to \infty$ errors are not allowed hence the SVM becomes a *hard margin* classifier with the slack variables set to zero. As C decreases a greater number of errors are allowed and it is known as a *soft margin* classifier. Another way of looking at this is that a large C gives a low bias, high variance model, while a small C gives a higher bias, low variance model [158]. The primal optimisation problem now looks to minimise $\frac{1}{2} ||\mathbf{w}||^2 + C \sum_{i=1}^{m} \xi_i$ subject to Equation 6.15.

$$y^{(i)} f\,(x^{(i)}) \geq 1 - \xi_i \quad \text{where} \quad \xi_i \geq 0 \quad \forall i \tag{6.15}$$

Kernel Usage: The use of kernels allows the classification of non-linearly separable data through mapping to a higher dimensional feature space where it is linearly separable. The dot product in Equation 6.13 and Equation 6.14 is replaced by a non-linear mapping Φ such that $\langle x^{(i)} \cdot x^{(j)} \rangle \rightarrow \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle$. There is no need to know this mapping explicitly, only the kernel $K(x^{(i)}, x^{(j)})$ which is known as the *kernel trick*. Commonly used kernel functions which are also used here given in Equation 6.16.

$$\begin{aligned} \textit{Linear} \quad & x^{(i)} \cdot x^{(j)} \\ \textit{Polynomial} \quad & \left(x^{(i)} \cdot x^{(j)}\right)^d \\ \textit{Gaussian} \quad & e^{-\left(\frac{|x^{(i)} - x^{(j)}|^2}{2\,\sigma^2}\right)} \end{aligned} \tag{6.16}$$

The linear kernel is equivalent to not using a kernel, and is only suitable in cases where the data is linearly separable. As shown in [19], for low noise target devices this is enough to allow the classification of Hamming weights. The Gaussian kernel is also commonly referred to as the radial basis function (RBF) kernel and the terms are used interchangeably here. Good visualisations of how the different kernels affect the decision boundary in the two-dimensional case are given in [103, 104]. These are not the only kernel functions however, and while arbitrary functions cannot be specified as a kernel (functions need to satisfy Mercer's Theorem [56]), there are many other options such as perceptrons, string, chi-square, histograms, *etc.* One disadvantage of using kernels is the extra parameter that now must be chosen, *e.g.* d and σ for polynomial and Gaussian kernels respectively, as well as the cost C. For the Gaussian kernel, increasing σ has the effect of smoothing the decision boundary thereby giving a lower variance model and vice-versa. This is the opposite to the cost C, however the relationship between them is non-linear. The classification algorithm is now modified by taking the sign of Equation 6.17.

$$f(x^{(i)}) = \sum_{j=1}^{m_s} \alpha_j \, y^{(j)} \, K\left(x^{(i)}, x^{(j)}\right) - b \tag{6.17}$$

Probabilistic Output: The SVM classification is given by the sign of Equation 6.17, and as previously mentioned this does not provide any probabilistic information on the decision. Due to the inherent noise in a SCA, probabilistic values are useful to determine the key by analysing multiple traces, or to enable an enhanced brute search correction of an estimated key guess. It was suggested by Platt in [180] to fit to Sigmoid function, as previously shown in Figure 6.4, to the returned distances

$f(x^{(i)})$ as shown in Equation 6.18, where the parameters $\{P_A, P_B\}$ are found by minimising the negative log-likelihood of the training data. This method of generating a probabilistic output was also used in [19], and a full explanation is given in [180].

$$Pr\left(y^{(i)} = 1 | x^{(i)}\right) = \frac{1}{1 + e^{P_A f\left(x^{(i)}\right) + P_B}} \tag{6.18}$$

The work in [103, 104] use a different variant of SVMs as just described, which is known as least squares support vector machines (LS-SVM) [210]. Here the model is formulated such that a solution can be found by solving a set of linear equations rather than a QP problem. Rather than building support vectors based only on a subset of samples, all samples are assigned some weighting value hence every sample is a support vector. Larger weights indicate a greater contribution to the training sample to the decision boundary. In practice, experiments indicate little difference in classification performance with the data sets under consideration. While LS-SVM can be faster for smaller data sets, a matrix inversion of the order of the number of training samples is required which can become prohibitive so the original SVM construction is used here.

The attacks presented here use the SVM$^{\text{light}}$ library [108] with the Matlab wrapper from Anton Schwaighofer available at [107]. A secondary wrapper as used in [212] is used to implement a grid search for the SVM cost parameter, as well as the extra parameters depending on the kernel if required. The grid search alternates holding one parameter constant while searching across a range of values for the other, using 10-fold cross validation to minimise the error, hence both parameters are selected with respect to each other. While this removes to need to select the SVM parameters, it does so at the expense of a considerably longer training time.

6.4.1 Attack on Multiplication

Using the same multiplication traces as in the LR based attack previously, classification is performed using SVMs to determine if a trace is from a multiplication or squaring operation. The choice of 10 k training traces for this data set across all machine learning methods, is actually due to the length of time it takes to train the SVM parameters. Larger trace sets incur memory problems, as well as a considerably longer training time due to the cross validation required to optimise parameter selection.

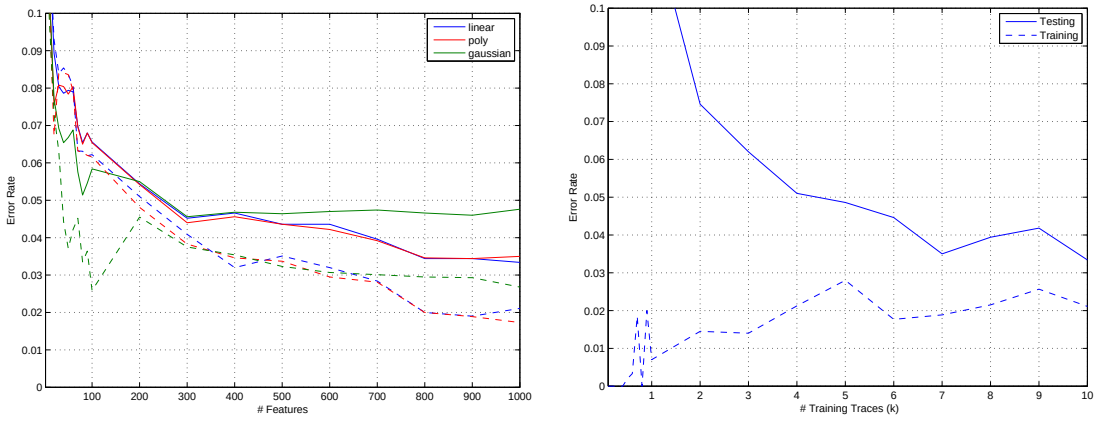
The error rate for the three kernels mentioned previously is given in Figure 6.10(a), where the dashed lines indicate the training error and the solid lines the test error. It can be

		Linear	Polynomial	Gaussian
Cost		0.002	0.005	0.2
Kernel Param.		–	1 (d)	8.41×10^{-4} (σ)
Platt	P_A	4.5310	3.9410	8.5171
	P_B	0.0876	0.0766	0.0907

Table 6.1: Learned SVM parameters.

seen that the linear and polynomial kernels outperform the Gaussian one, indicating that the data sets are somewhat linearly separable. The parameters learned using 1 k features for the three kernels are given in Table 6.1. The estimates for the *Platt parameters*, P_A and P_B to return a probabilistic value are also given.

The SVM algorithm was rerun, except this time setting the number of features to 1 k and varying the number of training traces, while using the linear kernel. As seen in Figure 6.10(b), the error rate is still falling as the number of training traces is increased, hence it is likely that further training samples would further decrease it, as was the case with the LR classification method. However the error rate is already quite low for smaller training set sizes.



(a) SVM kernel comparison.

(b) SVM training set size comparison.

Figure 6.10: SVM attack on 160-bit multiplications.

6.4.2 Attack on AES

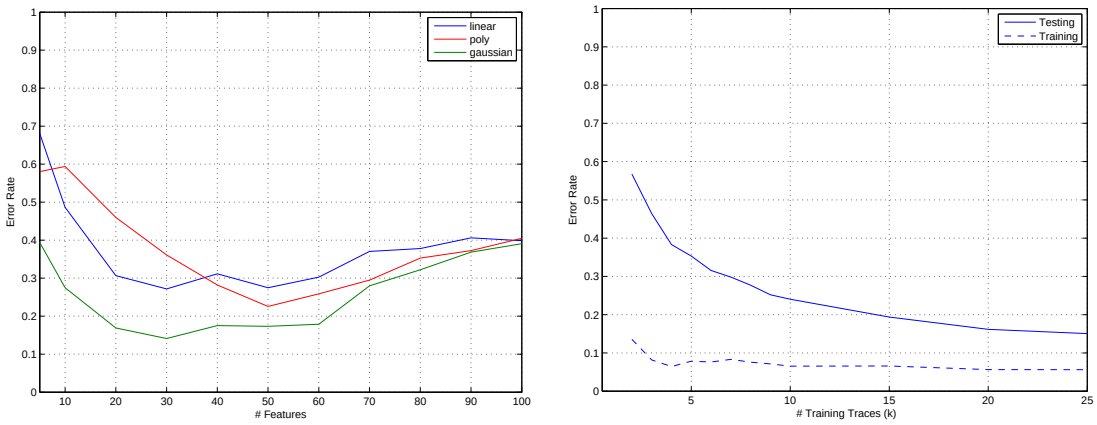
To extend SVMs for use in multi-class problems to determine the output of the AES *S-Box*, the same methods as before can be utilised again, that is *bitwise, one-v-all, one-v-one*. An empirical study of multi-class options for SVMs was undertaken in [62], and

found a modified version of the *one-v-one* method incorporating Platt's probabilities [180] was most efficient. However for the data set here, the $\binom{8}{2}$ number of combinations when using *one-v-one* leads to an unrealistic training time despite the fact only a portion of the training set is used each time. Likewise, due to the poor performance of the *bitwise* model previously in §6.3.2, this method is also discounted. Hence only the *one-v-all* method is used, however the large training set size again causes memory issues. To work around these memory problems, for each of the 256 classes, the class under consideration is labelled 1, and the -1 class is randomly selected from the remaining samples such that it is 10 times larger. This arbitrary value of 10 is chosen such that the resultant subset of the training set that is fed to the SVM has $\approx 1\,\text{k}$ samples, which through experimental analysis, allows the training of the SVM in a reasonable length of time without losing too much accuracy.

When training unbalanced data sets, the use of cross-validation to select parameter values that minimises the error rate can be problematic as biasing the model towards always correctly classifying the larger of the two classes will artificially lower the error rate and generalise poorly to unseen samples. The SVM construction allows for separate cost parameters, C, for both classes leading to asymmetric soft margins. However in this scenario, the penalisation of one class over another isn't desirable as both classes have an equal a priori probability of occuring. Hence rather than looking to minimise the overall error, the mean of the individual *scaled* class errors is minimised.

As can be seen in Figure 6.11(a), the Gaussian kernel gives the best results using 30 features. Also, unlike in the previous experiment distinguishing multiplication and squaring operations, as the number of features is increased the error rate also increases as features which do not contribute to the target leakage skew the model towards lowering the training error. Looking at the effect of the training set size in Figure 6.11(b), the error rate is once again asymptotically decreasing. For smaller training set sizes, at least two samples per class must be present to allow for training hence at least 512 samples are required, but as the samples are randomly selected from a larger set, in reality closer to 1 k are required before all classes have at least two samples. Finally it must be remembered that all samples are not used for training every multi-class SVM, hence even for smaller training set sizes the actual fraction of samples used is expected to be only $\approx \frac{11}{256}$.

For both attacks, the SVM performs similarly to LR which is also a binary classifier, despite only a subset of the training samples being selected because of computational issues due to the grid search in the AES case. Compared to regular TA, for the experiments here SVMs are not as effective, and required a considerably longer training time. While inefficient for



(a) SVM kernel comparison.

(b) SVM training set size comparison.

Figure 6.11: SVM attack on AES.

this particular experiment, there is many scenarios where they are advantageous however, and are interesting in a hardware context where the noise does not necessarily follow a Gaussian distribution.

6.5 Neural Networks

NNs were originally envisioned so as to mimic the operation of the brain, with multiple interconnected layers of neurons allowing the learning complex decision boundaries. Partly due to their computational overhead, and partly due to the fact SVMs were totally different to what had come before, they fell out of favour somewhat with the emergence of SVMs in the 90s. However with the advent of *deep-learning* methods and more powerful multi-core computers, they are once again considered state of the art [158].

A brief overview is given here, however for a through description of the topic one of the many books available such as [30, 63, 95] provide a good reference. A sample NN diagram is given in Figure 6.12. It consists of the input layer where each *unit*[2] represents an input feature, multiple hidden layers with differing number of hidden units, and an output layer where each unit represents a class. The model in Figure 6.12 has 3 features, 2 hidden layers with 4 and 3 units respectively, and is a binary classifier. A nice property of a NN over an SVM is that it naturally extends to the multi-class case with the addition of extra output units. Each layer except the output also has an extra *bias* unit which is fed into

[2]The units are alternatively known as *neurons* after as they were originally envisioned as representing neurons of the brain.

the following layer to account for the intercept term.

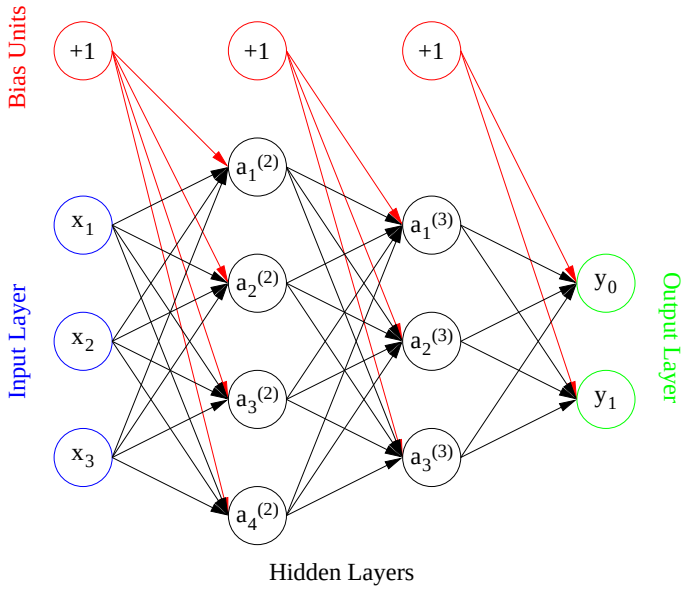


Figure 6.12: Artificial NN nodes.

Following the description in [158], every hidden unit and those in the output layer, has an activation function $a_j^{(i)}$ where j is the unit in a given layer, and i is the layer itself. While this activation unit is often the Sigmoid function as previously shown in Figure 6.4, other functions such as Gaussian (RBF) or step models are also possible. The initial activation units are assigned the input feature values as shown in Equation 6.19, with subsequent units calculated via some function $\mathcal{G}$, of the weighted sums of the previous layer, where $\mathcal{G}$ is the Sigmoid function in this work, and the weights are given by Θ. For clarity the unit subscripts in Equation 6.19 are omitted, and each $a^{(i)}$ term is a vector. As each element in the vector $a^{(i)}$ has a different set of weights for each unit of the previous layer $\Theta^{(i)}$ is therefore a matrix, *e.g.* in the illustration of Figure 6.12, $\Theta^{(1)}, \Theta^{(2)}$ and $\Theta^{(3)}$ are of size 5×4, 4×5 and 2×4 respectively, taking into account the bias units. The decision of a NN model given a input vector is given by $h_\Theta(x^{(i)}) = a^{(l)}$, where l is the number of layers in the model. Each unit of the feed-forward NN[3] outputs a non-linear function of its inputs, which themselves have been mapped non-linearly, allowing complex decision boundaries to be learned [63, ch. 6].

$$a^{(1)} = x^{(i)}, \quad a^{(i)} = \mathcal{G}(\Theta^{(i-1)}\, a^{(i-1)}) \tag{6.19}$$

[3]There are also *recurrent* NN that allow feedback within the network which essentially leads to a memory capability within the model.

Training a NN model requires the learning of the weighting parameters Θ for each layer to minimise the cost function as given in Equation 6.20 [158]. The most common method to determine these weights is through the use of the *backpropagation* algorithm. Random initial weights for Θ are assigned, and for each training sample the output $h_\Theta(x^{(i)})$ is calculated. The error is then calculated by subtracting the actual class label $y^{(i)}$ (note that the one-hot encoding is used to encode the labels), and the errors for each training sample are propagated back through the network to calculate the error at each layer. Minimisation of Equation 6.20 entails modifying the weights Θ through an iterative algorithm like gradient descent such that error is reduced. The number of classes in Equation 6.20 is denoted by $\mathcal{K}$, the number of samples by m, the number of layers by l, and the number of units in a given layer by u_i. A regularisation parameter λ is again present as in LR, to prevent over-fitting of the training data through the selection of arbitrarily large Θ values.

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{\mathcal{K}} y_j^{(i)} \log(h_\Theta(x^{(i)}))_j + (1 - y_j^{(i)}) \log((1 - h_\Theta(x^{(i)}))_j)\right] + \frac{\lambda}{2m}\sum_{i=1}^{l-1}\sum_{j=1}^{u_i}\sum_{k=1}^{u_{i+1}}\left(\Theta_{kj}^{(i)}\right)^2 \tag{6.20}$$

There are no set rules with regards the selection of parameters for NN, with the number of hidden units/layers greatly affecting the classification performance. Due to the computational complexity of training a NN, a cross validated grid search as with the SVMs is inefficient to implement, and neither it is clear how to optimally trade off hidden units and layers while searching. There are a few *rules of thumb* to select parameters however. It must also be pointed out that if the data is linearly separable, then no hidden layers are required, and the NN is similar to LR. It is suggested in [158] that the number of hidden layers should initially be set to 1, and the number of hidden units set to approximately the number of features, subsequently extra hidden layers can be added in an attempt to improve performance. Where a large number of features are available however, this requires a significant amount of time to train. It is suggested in [30] that multiple layers often add little in the way of performance improvement except in specialised cases, and that choosing the number of units to be the mean of the input and output units gives a good starting point. Regardless of selection advice followed, the optimal parameters are going to be determined by the data.

The NN was implemented following the tutorial in [158], with the same gradient descent based minimisation function as used for LR once again used here. The weighted inputs

for the activation units for each layer can be efficiently calculated en bloc using matrix multiplication, with subsequent unrolling allowing a vectorised input to the minimisation function. The number of gradient descent iterations was restricted to a maximum of 250, this number was heuristically chosen by examining the cost as a function of the number of iterations such as in Figure 6.13. Here, as was the case with many other experiments, minimisation occurred within $100 \rightarrow 150$ iterations.

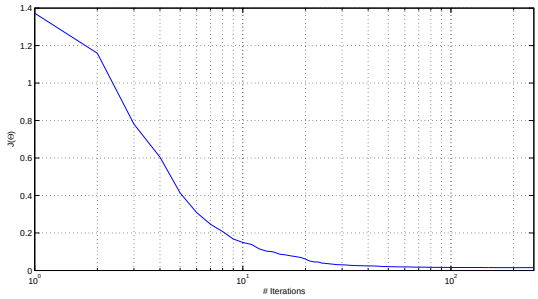


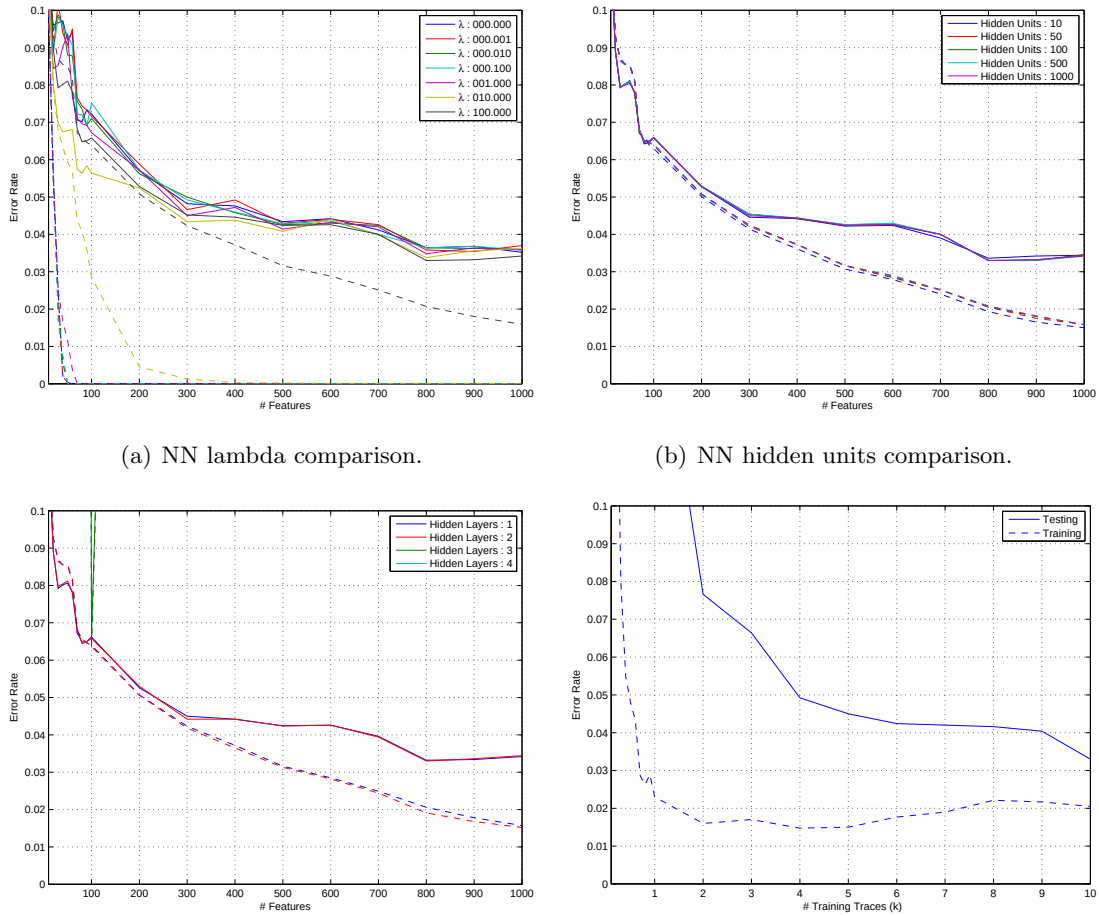
Figure 6.13: NN cost function.

6.5.1 Attack on Multiplication

Using the 160-bit multiplication and squaring traces as before, a NN model with a single layer was trained for different values of λ. The number of hidden units was set to be equal to number of input features, with the error rate as a function of the number of features given in Figure 6.14(a). As can be seen, in this binary classification problem the λ parameter has little effect, with $\lambda = 100$ giving slightly better results similar to LR.

Taking this value of λ, the number of hidden units is then varied independently of the number of input features. This makes little difference to the error rate as shown in Figure 6.14(b), however smaller values for the number of hidden units are considerably faster to train hence 100 units is chosen to examine the effect of multiple hidden layers in Figure 6.14(c). Again, the addition of extra layers has little effect on the error indicating that, for this dataset, the rule of thumb as given in [30] is appropriate. The NN model appears somewhat robust to parameter selection in this scenario, hence the parameters can be picked with regards to efficiency of training.

Finally the error as a function of training set size for a single hidden layer NN model with 100 units and $\lambda = 100$ is plotted in Figure 6.14(d). Like the LR and SVM based attacks, the error rate is still falling hence it is reasonable to expect that a larger training set will further decrease the error.



(a) NN lambda comparison.

(b) NN hidden units comparison.

(c) NN hidden layers comparison.

(d) NN training set size comparison.

Figure 6.14: NN attack on 160-bit multiplication.

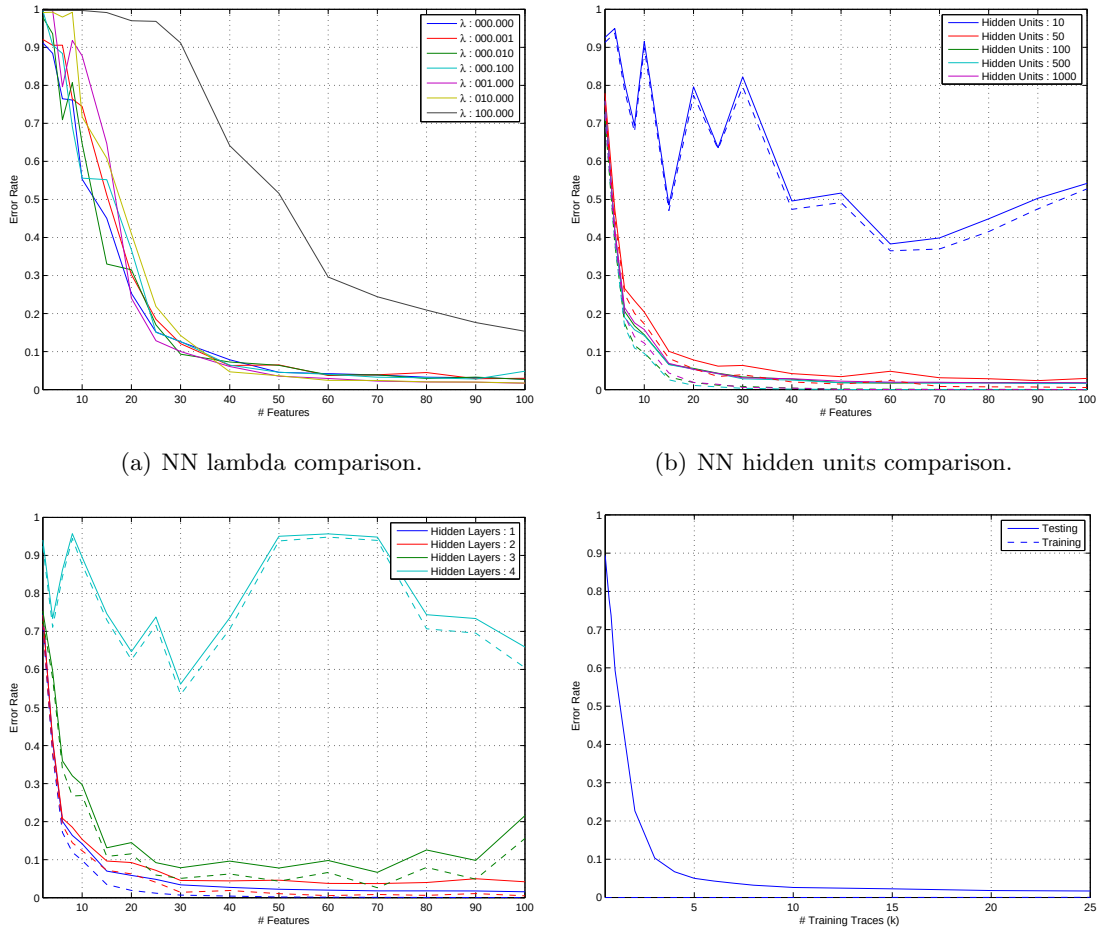
6.5.2 Attack on AES

Following the same method as with the multiplication traces, NN models were trained for the ARM7 microprocessor AES traces for various values of λ, using a single hidden layer with the number of hidden units of the order of the number of features. Figure 6.15(a) shows that large values of λ significantly affect the results for the worse, with $\lambda = \langle 1, 10 \rangle$ giving the lowest error.

Taking $\lambda = 1$, the number of hidden features for a single layer was varied independently of the input feature size, Figure 6.15(b). Here, in contrast with the multiplication traces, too few hidden units gives a large error as the small number of units doesn't allow for accurate modelling of the $\mathcal{K} = 256$ multi-class problem. More than 100 units doesn't further decrease the error, hence in Figure 6.15(c), the effect of the number of hidden layers is examined, while setting $\lambda = 1$ and the number of hidden units to 100. Interestingly,

increasing the hidden layers actually increases the error, with more layers giving a larger error. This isn't necessarily due to over-fitting of the data set, as the training error also increases proportionally.

Finally the error as a function of the training error using only a single hidden layer is plotted in Figure 6.15(d). The error converges quite fast with only a small decrease in the error as the set size increases beyond 10 k traces.



(a) NN lambda comparison.

(b) NN hidden units comparison.

(c) NN hidden layers comparison.

(d) NN training set size comparison.

Figure 6.15: NN attack on AES.

In the two experiments above, NN have been shown to be a particularly effective for profiling attacks, as well as being somewhat robust to parameter selection when *reasonable* values for λ and the number of hidden units are selected. The selected parameters are by no means optimal and a logarithmic grid search as implemented for the SVMs could further reduce the error.

Classifier	Parameters
LDA	single covariance matrix for all classes
QDA	separate covariance matrix for each class
SM	*Max* method with half the trace for estimating the noise, 2^{nd} order polynomial expansion performed on class labels
LR	regularisation parameter $\lambda = 100$
SVM	linear kernel with 10-fold cross validation used to estimate cost C
NN	single layer with 100 hidden units and $\lambda = 100$

Table 6.2: Classification parameters → multiplication.

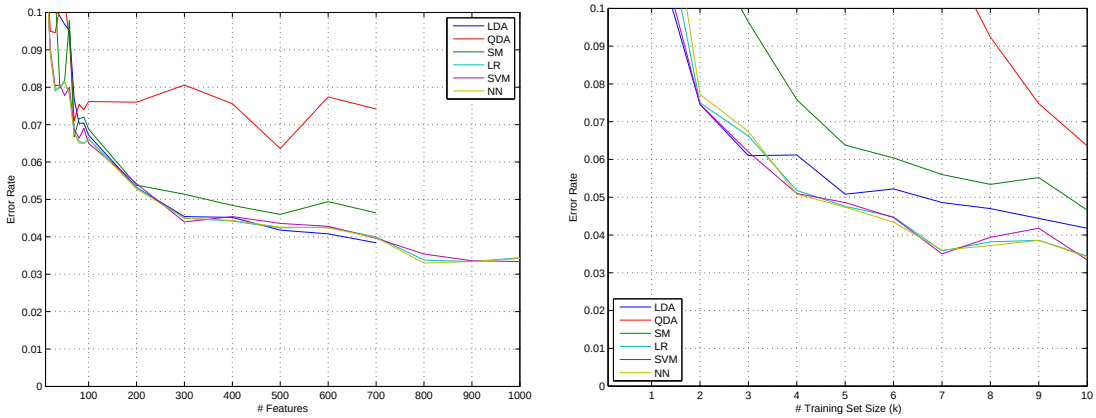
6.6 Comparison

As previously mentioned, comparisons between TAs and stochastic based attacks have previously been performed in [81, 205], and comparisons between TAs and SVMs in [19, 99, 135]. Here, the parameters for the various classification algorithms which give the lowest testing error are plotted against each other for the multiplication and AES traces.

6.6.1 Attack on Multiplication

Using the algorithm parameters as given in Table 6.2 determined by the previous sections, the various learning algorithms are compared in Figure 6.16(a) by plotting the error as a function of the number of retained features. While the multiplication traces weren't used previously for SM attacks in §6.2, experimental analysis shows that a 2^{nd} order polynomial minimises the error. It is noticeable that all the multivariate Gaussian distribution based methods run into numerical difficulties when more than 700 features are retained, however the extra features contribute some information as the error rate continues to fall for the LR, SVM and NN methods.

Taking the number of features that minimises the error for 10 k training traces, the error as a function of the training set size is given in Figure 6.16(b). Again there is little difference between using LR, SVM or NN, with QDA having close to twice the error rate of these. Note that results for LDA and QDA are different to that in §5.5.3, as a different feature selection method and training set size is used.



(a) Error as a function of features.

(b) Error as a function of set-size.

Figure 6.16: Classification performance comparison → multiplication.

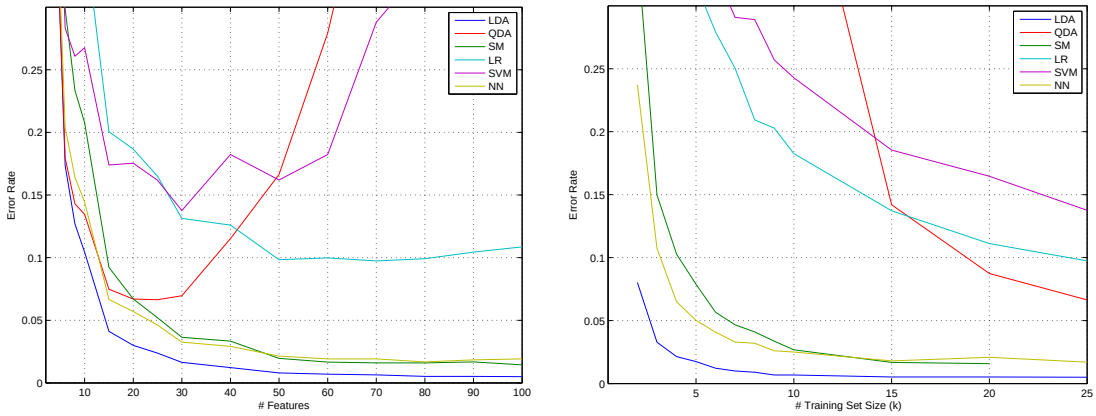
6.6.2 Attack on AES

The selection parameters for each classification algorithm when attacking the AES traces are given in Table 6.2, with the error rate as a function of the number of features given in Figure 6.17(a). It is clear that the use of LDA still gives the best classification performance, with NN and high-order polynomial based SM also giving very low error rates. In contrast the original TA (QDA) gives comparatively poor results, as well as becoming numerically unstable for higher numbers of features. This is expected due to the fewer traces available to model the covariance matrix for each class. As noise is independent of the deterministic power consumption (*i.e.* the mean vectors $\mu^{(i)}$) it's perfectly reasonable to use a single covariance matrix. The LR and SVM binary classifiers perform poorly in this multi-class experiment, however keeping in mind that the *One-v-All* method was used when the *One-v-All* method gave, in the case of LR, slightly more accurate results. Also, with regards to the SVM results, the full training set wasn't used for each model.

Looking at the effect of the training set size, the number of features that minimised the error for each classification algorithm are retained and used in Figure 6.17(b). It is clear that LDA is the most efficient here, as well as having the lowest error rate. One particular advantage of the SM is its ability to build models given very few profiling traces, as previously noted in [81, 205], as a profiling trace for each class label is not necessarily required due to the choice of base function. Note that if the *one-hot* method was used as a base function then its profiling sample size requirements would be similar to that of LDA and NNs.

Classifier	Parameters
LDA	single covariance matrix for all classes
QDA	separate covariance matrix for each class
SM	*Max* method with half the trace for estimating the noise, 6^{th} order polynomial expansion performed on bit decomposition of class values
LR	*One-v-All* multi-class, regularisation parameter $\lambda = 1$
SVM	*One-v-All* multi-class with the number of "All" samples restricted to $\times 10$ of the "One", Gaussian kernel with 10-fold cross validation used to estimate cost C and sigma σ.
NN	single layer with 100 hidden units and $\lambda = 1$

Table 6.3: Classification parameters → AES.



(a) Error as a function of features.

(b) Error as a function of set-size.

Figure 6.17: Classification performance comparison → AES.

6.7 A Note on Feature Selection

A central step prior to the application of the machine learning algorithms above, was the selection of relevant time instants to pass to the learning algorithm as features. This topic was previously explored in §3.3.3, however warrants a further mention here with regards to the non-parametric SVM and NN algorithms. Recall that the SOST method as introduced in [81], is an adaptation of the Student's t-test in Equation 3.8 (or Welch's t-test where the class variances are not assumed equal) to determine the points in time of a trace where the mean class values are different. This method of feature selection immediately biases the results in favour of LDA/QDA and SM, as they seek to classify based on distinguishing means assuming the Gaussian distribution of data (remember the SVM results are further

disadvantaged due to the use of a likely sub-optimal multi-class method, and the fact only a subset of the training samples are used due to computational issues). This is not to say that SVMs and NNs would give better results without feature selection, but it is worth keeping in mind when conducting a profiling attack.

LR and NN based classification methods are somewhat robust to feature selection as they learn a weighting or ranking value to each input feature during the profiling stage. This was shown for LR previously when targeting the multiplication traces by plotting the weights in Figure 6.6. This can equally be shown for the weighting values in a single layer NN and is given in Figure 6.18, where the weights for each of the 100 hidden units are overlaid on top of each other in the lower graph. Therefore, the feature reduction step isn't strictly required, however it can give more accurate results as it prevents over-fitting of the model to points in time which don't contribute to distinguishing between the classes. Note that LDA too is somewhat robust to feature selection, however due to the matrix inversion required on the order of the number of features, larger traces must be reduced somehow. Something similar was suggested by Bartkewitz *et al.* in [19] for SVMs where, after learning the Lagrange multipliers α_i, low values of the weight vector $|w_j|$ were set to zero as they contribute less to the classification performance. There is much further research in the field of computer vision as to how to optimally select features for SVMs such as [159].

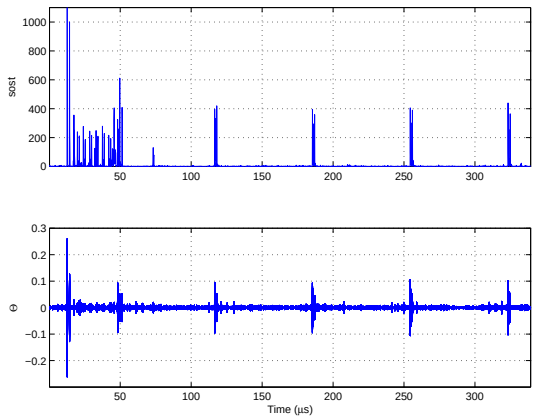


Figure 6.18: NN hidden layer weights.

An interesting application of non-parametric learning algorithms could be with regards to *feature construction*. As previously mentioned, this was already briefly looked at in [96] when using an SVM to extract DES keys from a smart-card by appending plaintext bits. Appending or replacing traces with mean (or distances to means), variance or entropy values could lead to interesting low-order non-Gaussian datasets to classify. Likewise,

non-linear or polynomial expansions of the training data (as opposed to the expansion of the basis function of the training labels as in SM attacks) could also be interesting higher order attack to explore as it would allow dependencies in the time domain to be taken into account also. A similar approach taken by Lerman *et al.* in [136] has particular merit, as it remaps the training traces to take into account said time-wise dependencies prior to applying the learning algorithm.

6.8 Conclusion

In this chapter a comprehensive empirical comparison between different profiling attacks was conducted on two data sets. Extensions to the SM of profiling attacks were looked at, and LR and NN were evaluated for the first time in the context of SCA. The performance of SVMs was also examined, using a grid search algorithm in conjunction with cross validation for optimal parameter selection.

The results show that for data sets that follow a Gaussian distribution, LDA is hard to beat and should be the first algorithm utilised by an attacker. As well as returning the lowest error rate, it is also exceptionally quick to train as only mean traces for each class and a single covariance matrix need be modelled. NN have significant potential in the context of SCA also and further research on protected implementations is an interesting topic. While they take slightly longer than LDA to train, they seem relatively robust to numerical errors and are a useful general classifier to use in conjunction with LDA as they do not assume Gaussian distributed data.

Where there was a large ($|\mathcal{K}| = 256$) number of multi-class values, the performance of the binary LR and SVM classification algorithms suffered, however they should not be discounted as in the binary case when attacking the multiplication traces, SVMs performed as well as the other algorithms. In scenarios where a particularly large data set with a high-order number of features, the simplicity of LR might ensue that its the only feasible learning algorithm to use. At the very least it would be worthwhile to initially train using LR to see if slower algorithms such as SVMs or NNs are worth pursuing or do the features need to be looked at first.

Finally the results in this chapter all pertain to software implementations, however the non-parametric nature of some of the learning algorithms would likely suit a hardware target. For example, the nature of FPGA leakage is such that it is dependent on the placement of logic within the slice look-up tables (LUTs) and the subsequent path along the interconnect

that data travels. The contribution of the interconnect can often be significantly more than that of the slices themselves. Hence it is considerably more irregular than an embedded software implementation, and the use of non-parametric learning algorithms such as NN or SVM could have significant advantages over LDA.

Chapter 7

Conclusion

In the connected world we now live in, embedded systems play an increasing role as "smart devices" become ever more prevalent. The ability to check bank balances, transfer money, send email, read social media messages on the go, *etc.*, is no longer considered a rarity but is an expectation. Coupled with the expanding use of sensor data for increasing automation in motor vehicles, the use of cryptography is essential to allow these devices perform their functionality without leaking sensitive or critical information. The recent *Snowden* revelations only highlight the need for secure end-to-end encryption in all digital communications, as it is not just faceless criminals that seek to exploit our data. In this thesis, a comprehensive evaluation of various profiling SCAs has been carried out against cryptographic algorithms implemented on an ARM7 microprocessor such as would be utilised in an embedded scenario.

7.1 Contributions of this Thesis

In Chapter 3 a thorough analysis of template attacks was carried out, with various parameters in the training stage analysed to see the effect on the resultant classification error. It was found that linear discriminant analysis with only a single covariance matrix was generally more effective than the classical method, quadratic discriminant analysis, where a covariance matrix for each class was required. It also had the advantage of being less susceptible to numerical errors as all the traces were used to estimate the single covariance matrix. The effect of noise was also examined through artificially adding normally distributed Gaussian data to the ARM7 microprocessor traces, as well as comparing the

error rate of the attack on different software platforms. Finally the underlying concept of the template attack in that power traces from one device can be used to attack another is empirically evaluated for a set of smart-cards, and it is shown how to improve the cross-device classification error by performing the training step with traces from multiple devices.

In Chapter 4, the work from the paper *Unknown Plaintext Template Attacks* [92] as presented at the *10th International Workshop on Information Security Applications* conference is presented and expanded upon. Key recovery is performed in this case without any knowledge of the input or output data by building multiple templates at different points of the power consumption trace. Theoretical results are given for the expected number of traces required when the Hamming weight model is used, as well as how to decrease the attack error by targeting only the *S-Boxes*. The effect of masking on the attack is also examined, as is an analysis where the attack is performed on the key schedule, highlighting the importance of protecting all steps of an algorithm.

Chapter 5 is based on the work from *Using templates to distinguish multiplications from squaring operations* [93] as published in the *International Journal of Information Security*. An analysis of the power consumption difference between a squaring and multiplication is presented, and a template attack is derived utilising this leakage. Practical results against an implementation of the Montgomery multiplication algorithm are presented, as well an analysis of how the length of the key affects the classification accuracy. The effect of countermeasures in the case of an RSA cryptosystem is examined, and the attack is then extended to ECC based cryptosystems where a practical attack is conducted against a protected ECDSA implementation. This attack builds templates on the underlying curve doubling and addition formulas, which are then used across the ECDSA trace length in order to recover the entire key. It is shown that many commonly applied countermeasures are ineffective against this type of attack.

Finally in Chapter 6, machine learning methods are investigated for SCA. Different encoding schemes for the stochastic method are examined, with a comparison to the original proposal given. Logistic regression and neural networks are used for the first time in the context of SCA, with various respective parameter choices examined. In the case of logistic regression different multi-class options are also examined. The use of support vector machines is also explored, and all methods are compared to classical template attacks. The comparison is conducted using the both the multiplication traces which is a binary classification problem, and the AES trace set which is a multi-class problem, to evaluate

the respective performance of the algorithms. It is shown that linear discriminant analysis or neural networks are both effective general classifiers which should be applicable in a wide range of scenarios.

7.2 Future Research Directions

Machine learning is an exciting area of research and there is considerable potential for further research in the context of SCA here. The use of unsupervised machine learning methods to overcome randomisation countermeasures needs to be further examined. As mentioned in §6.7 on feature selection, the artificial construction of features is another research topic that has a wide scope. The use of unsupervised learning methods such as clustering, auto-encoder networks or support vector machine based feature selection, to allow learning of features from unlabelled data could be applicable here also. Note that these would not just be suitable for profiling attacks, but more general non-profiling attacks also. The use of HMMs for error correction of incorrectly classified bits as in §5.7.3 also needs to be further explored.

Practically, a successful application of templates on a real world product has yet to be fully realised (the attack as presented in [169] did not transfer across smart-cards due to countermeasures present). The building of templates across multiple devices is also worth further examination, particularly with regards to hardware implementations where in the case of FPGAs, the circuit layout might not be exactly the same should a bit file be regenerated prior to programming the next device. It might be possible to determine some sort of rule of thumb as to how many devices and adversary should aim to model on based on the SNR of the traces. A *real-world* analysis of building templates on public verification functions, or indeed just a single multiplication operation, could also be powerful addition to template attack if proven practically feasible.

From a learning algorithm viewpoint, there are many more learning methods that could be used for SCA. Custom kernels could also be explored for support vector machines optimised for some leakage model, or custom activation units in the case of neural networks. Recurrent neural networks which allow for feedback within the model, could be interesting to explore where there is time-series dependencies such as in masking, or in the context of asymmetric algorithms where some patterns of key bits won't be possible. The use of ensemble learning, where a multitude of classifiers are used, to attack some cryptographic system could also be examined for optimal error reduction.

Publications Relating to this Thesis

Journal Papers

Neil Hanley, Michael Tunstall, and William P. Marnane. Using Templates to Distinguish Multiplications from Squaring Operations. *International Journal of Information Security*, 10(4):255–266, 2011.

Conference Papers with Proceedings

Neil Hanley, Maire O'Neill, Michael Tunstall, and William P. Marnane. Empirical Evaluation of Multi-Device Profiling Side-Channel Attacks. In *IEEE International Workshop on Signal Processing Systems — SiPS 2014*. IEEE Signal Processing Society, 2014.

Neil Hanley, Michael Tunstall, and William P. Marnane. Unknown Plaintext Template Attacks. In Heung Youl Youm and Moti Yung, editors, *Information Security Applications — WISA 2009*, volume 5932 of *Lecture Notes in Computer Science*, pages 148–162. Springer-Verlag, 2009.

Michael Tunstall, Neil Hanley, Robert P. McEvoy, Claire Whelan, Colin C. Murphy, and William P. Marnane. Correlation Power Analysis of Large Word Sizes. In *Irish Signals and System Conference — ISSC 2007*, pages 145–150. IET, 2007.

Book Chapters

Philipp Grabher, Neil Hanley, and Michael Tunstall. Attacking Smart Cards: Side Channel and Fault Analysis. In Karl C. Verdinand, editor, *Computer Science Research and Technology*, chapter 10, pages 207–226. Nova Publishing, 2011.

IACR ePrint Archive

Neil Hanley, Michael Tunstall, and William P. Marnane. Using Templates to Distinguish Multiplications from Squaring Operations. Cryptology ePrint Archive, Report 2011/236, 2011. `http://eprint.iacr.org/`.

Bibliography

[1] Moulay Abdelaziz El Aabid, Sylvain Guilley, and Philippe Hoogvorst. Template Attacks with a Power Model. Cryptology ePrint Archive, Report 2007/443, 2007. `http://eprint.iacr.org/`.

[2] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM Side-Channel(s): Attacks and Assessment Methodologies. In Kaliski et al. [115], pages 29–45.

[3] Dakshi Agrawal, Josyula R. Rao, Pankaj Rohatgi, and Kai Schramm. Templates as Master Keys. In Rao and Sunar [185], pages 15–29.

[4] Toru Akishita and Tsuyoshi Takagi. Power Analysis to ECC Using Differential Power Between Multiplication and Squaring. In Josep Domingo-Ferrer, Joachim Posegga, and Daniel Schreckling, editors, *Smart Card Research and Advanced Applications — CARDIS 2006*, volume 3928 of *Lecture Notes in Computer Science*, pages 151–164. Springer-Verlag, 2006.

[5] Mehdi-Laurent Akkar, Régis Bevan, Paul Dischamp, and Didier Moyart. Power Analysis, What Is Now Possible... In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 489–502. Springer-Verlag, 2000.

[6] Mehdi-Laurent Akkar, Régis Bevan, and Louis Goubin. Two Power Analysis Attacks against One-Mask Methods. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption — FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 332–347. Springer-Verlag, 2004.

[7] Mehdi-Laurent Akkar and Christophe Giraud. An Implementation of DES and AES, Secure against Some Attacks. In Çetin Kaya Koç et al. [39], pages 309–318.

[8] Mehdi-Laurent Akkar and Louis Goubin. A Generic Protection against High-Order Differential Power Analysis. In Johansson [109], pages 192–205.

[9] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In *IEEE Symposium on Security and Privacy*, pages 526–540. IEEE Computer Society, 2013.

[10] Frederic Amiel, Benoit Feix, Michael Tunstall, Claire Whelan, and William P. Marnane. Distinguishing Multiplications from Squaring Operations. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography — SAC 2008*, volume 5932 of *Lecture Notes in Computer Science*, pages 148–162. Springer-Verlag, 2009.

[11] Cédric Archambeau, Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Template Attacks in Principal Subspaces. In Goubin and Matsui [84], pages 1–14.

[12] ATMEL. Atmel CryptoMemory. `http://www.atmel.com/products/security-ics/secure-memory/default.aspx`. Online; accessed 19-Mar-2013.

[13] ATMEL. AVR XMEGA Microcontrollers. `http://www.atmel.com/products/microcontrollers/avr/avr_xmega.aspx`. Online; accessed 19-Mar-2013.

[14] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic Side-channel Attacks on Printers. In *USENIX Security Symposium*, pages 20–20. USENIX Association, 2010.

[15] Josep Balasch, Benedikt Gierlichs, Roel Verdult, Lejla Batina, and Ingrid Verbauwhede. Power Analysis of Atmel CryptoMemory - Recovering Keys from Secure EEPROMs. In Dunkelman [64], pages 19–34.

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

[17] Alessandro Barenghi, Gerardo Pelosi, and Yannick Teglia. Improving First Order Differential Power Attacks through Digital Signal Processing. In Oleg B. Makarevich, Atilla Elçi, Mehmet A. Orgun, Sorin A. Huss, Ludmila K. Babenko, Alexander G.

Chefranov, and Vijay Varadharajan, editors, *Conference on Security of Information and Networks — SIN 2010*, pages 124–133. ACM, 2010.

[18] Elaine Barker and John Kelsey. NIST SP 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators. National Institute of Standards and Technology publication, 2012. `http://www.nist.gov/publication-portal.cfm`.

[19] Timo Bartkewitz and Kerstin Lemke-Rust. Efficient Template Attacks Based on Probabilistic Multi-class Support Vector Machines. In Mangard [144], pages 263–276.

[20] Lejla Batina, Benedikt Gierlichs, and Kerstin Lemke-Rust. Differential Cluster Analysis. In Clavier and Gaj [50], pages 112–127.

[21] Lejla Batina, Jip Hogenboom, and Jasper G. J. van Woudenberg. Getting More from PCA: First Results of Using Principal Component Analysis for Extensive Power Analysis. In Dunkelman [64], pages 383–397.

[22] Josh Benaloh, editor. *Topics in Cryptology — CT-RSA 2014*, volume 8366 of *Lecture Notes in Computer Science*. Springer-Verlag, 2014.

[23] Daniel J. Bernstein. Cache-timing attacks on AES. `http://cr.yp.to/papers.html`, 2004. Online; accessed 23-Aug-2009.

[24] Daniel J. Bernstein and Tanja Lange. Faster Addition and Doubling on Elliptic Curves. In Kaoru Kurosawa, editor, *Advances in Cryptology — ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 29–50. Springer-Verlag, 2007.

[25] Guido Bertoni, Luca Breveglieri, Pasqualina Fragneto, Marco Macchetti, and Stefano Marchesin. Efficient Software Implementation of AES on 32-Bit Platforms. In Kaliski et al. [115], pages 159–171.

[26] Guido Bertoni and Jean-Sébastien Coron, editors. *Cryptographic Hardware and Embedded Systems — CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*. Springer-Verlag, 2013.

[27] Régis Bevan and Erik Knudsen. Ways to Enhance Differential Power Analysis. In Pil Joong Lee and Chae Hoon Lim, editors, *Information Security and Cryptology — ICISC 2002*, volume 2587 of *Lecture Notes in Computer Science*, pages 327–342. Springer-Verlag, 2003.

[28] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. NICV: Normalized Inter-Class Variance for Detection of Side-Channel Leakage. Cryptology ePrint Archive, Report 2013/717, 2013. `http://eprint.iacr.org/`.

[29] Eli Biham and Adi Shamir. Power Analysis of the Key Scheduling of the AES Candidates. In *Proceedings of the Second Advanced Encryption Standard (AES) Candidate Conference*. NIST, 1999.

[30] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[31] Daniel Bleichenbacher. On The Generation of One-Time Keys in DL Signature Schemes. Presentation at IEEE P1363 Working Group meeting, 2000.

[32] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997.

[33] Eric Brier, Christophe Clavier, and Francis Olivier. Optimal Statistical Power Analysis. Cryptology ePrint Archive, Report 2003/152, 2003. `http://eprint.iacr.org/`.

[34] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In Joye and Quisquater [111], pages 16–29.

[35] Eric Brier and Marc Joye. Weierstraß Elliptic Curves and Side-Channel Attacks. In David Naccache and Pascal Paillier, editors, *Public Key Cryptography — PKC 2002*, volume 2274 of *Lecture Notes in Computer Science*, pages 335–345. Springer-Verlag, 2002.

[36] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[37] Marco Bucci, Luca Giancane, Raimondo Luzzi, M. Marino, Giuseppe Scotti, and Alessandro Trifiletti. Enhancing power analysis attacks against cryptographic devices. *IET Circuits, Devices & Systems*, 2(3):298–305, 2008.

[38] Bushing, Marcan, Segher, and Sven. Console Hacking 2010. 27^{th} Chaos Communication Congress, 2010.

[39] Çetin Kaya Koç, David Naccache, and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems — CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[40] Çetin Kaya Koç and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems — CHES 1999*, volume 1717 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[41] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In Wiener [228], pages 398–412.

[42] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In Kaliski et al. [115], pages 13–28.

[43] David Chaum. Security Without Identification: Transaction Systems to Make Big Brother Obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[44] Benoît Chevallier-Mames, Mathieu Ciet, and Marc Joye. Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity. *IEEE Transactions on Computers*, 53(6):760–768, 2004.

[45] Omar Choudary and Markus G. Kuhn. Efficient Template Attacks. Cryptology ePrint Archive, Report 2013/770, 2013. `http://eprint.iacr.org/`.

[46] Kyo-Il Chung, Kiwook Sohn, and Moti Yung, editors. *Information Security Applications — WISA 2008*, volume 5379 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[47] Christophe Clavier, Benoit Feix, Georges Gagnerot, Christophe Giraud, Mylène Roussellet, and Vincent Verneuil. ROSETTA for Single Trace Analysis : Recovery of Secret Exponent by Triangular Trace Analysis. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012*, volume 7668 of *Lecture Notes in Computer Science*, pages 140–155. Springer-Verlag, 2012.

[48] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Horizontal Correlation Analysis on Exponentiation. In Miguel Soriano, Sihan Qing, and Javier López, editors, *Information and Communications Security — ICICS 2010*, volume 6476 of *Lecture Notes in Computer Science*, pages 46–61. Springer-Verlag, 2010.

[49] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Square Always Exponentiation. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *Progress in Cryptology — INDOCRYPT 2011*, volume 7107 of *Lecture Notes in Computer Science*, pages 40–57. Springer-Verlag, 2011.

[50] Christophe Clavier and Kris Gaj, editors. *Cryptographic Hardware and Embedded Systems — CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[51] William E. Cobb, Eric D. Laspe, Rusty O. Baldwin, Michael A. Temple, and Yong C. Kim. Intrinsic physical-layer authentication of integrated circuits. *IEEE Transactions on Information Forensics and Security*, 7(1):14–24, 2012.

[52] Jean-Sébastien Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Çetin Kaya Koç and Paar [40], pages 292–302.

[53] Jean-Sebastien Coron, David Naccache, and Paul Kocher. Statistics and secret leakage. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(3):492–508, 2004.

[54] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[55] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In Yuliang Zheng, editor, *Advances in Cryptology — ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer-Verlag, 2002.

[56] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[57] Cryptome. NSA TEMPEST Documents. `http://cryptome.org/nsa-tempest.htm`. Online; accessed 15-Dec-2013.

[58] Guillaume Dabosville, Julien Doget, and Emmanuel Prouff. A New Second-Order Side Channel Attack Based on Linear Regression. *IEEE Transactions on Computers*, 62(8):1629–1640, 2013.

[59] Jean-Luc Danger, Sylvain Guilley, Philippe Hoogvorst, Cédric Murdica, and David Naccache. A synthesis of side-channel attacks on elliptic curve cryptography in smartcards. *Journal of Cryptographic Engineering*, 3(4):241–265, 2013.

[60] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[61] Julien Doget, Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate side channel attacks and leakage modeling. *Journal of Cryptographic Engineering*, 1(2):123–144, 2011.

[62] Kaibo Duan and S. Sathiya Keerthi. Which Is the Best Multiclass SVM Method? An Empirical Study. In Nikunj C. Oza, Robi Polikar, Josef Kittler, and Fabio Roli, editors, *Multiple Classifier Systems*, volume 3541 of *Lecture Notes in Computer Science*, pages 278–285. Springer-Verlag, 2005.

[63] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2001.

[64] Orr Dunkelman, editor. *Topics in Cryptology — CT-RSA 2012*, volume 7178 of *Lecture Notes in Computer Science*. Springer-Verlag, 2012.

[65] François Durvaux, Mathieu Renauld, François-Xavier Standaert, Loic van Oldeneel tot Oldenzeel, and Nicolas Veyrat-Charvillon. Cryptanalysis of the CHES 2009/2010 Random Delay Countermeasure. Cryptology ePrint Archive, Report 2012/038, 2012. `http://eprint.iacr.org/`.

[66] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. National Institute of Standards and Technology publication, 2001. `http://www.nist.gov/publication-portal.cfm`.

[67] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani. On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoqCode Hopping Scheme. In David Wagner, editor, *Advances in Cryptology — CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 203–220. Springer-Verlag, 2008.

[68] M. Abdelaziz Elaabid and Sylvain Guilley. Practical Improvements of Profiled Side-Channel Attacks on a Hardware Crypto-Accelerator. In Daniel J. Bernstein and Tanja Lange, editors, *Progress in Cryptology — AFRICACRYPT 2010*, volume 6055 of *Lecture Notes in Computer Science*, pages 243–260. Springer-Verlag, 2010.

[69] M. Abdelaziz Elaabid and Sylvain Guilley. Portability of templates. *Journal of Cryptographic Engineering*, 2(1):63–74, 2012.

[70] Paul N. Fahn and Peter K. Pearson. IPA: A New Class of Power Attacks. In Çetin Kaya Koç and Paar [40], pages 173–186.

[71] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. State-of-the-art of Secure ECC Implementations: A Survey on Known Side-channel Attacks and Countermeasures. In Jim Plusquellic and Ken Mai, editors, *International Workshop on Hardware-Oriented Security and Trust — HOST 2010*, pages 76–87. IEEE Computer Society, 2010.

[72] Junfeng Fan and Ingrid Verbauwhede. An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost. In David Naccache, editor, *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, volume 6805 of *Lecture Notes in Computer Science*, pages 265–282. Springer-Verlag, 2012.

[73] Julie Ferrigno and Martin Hlavác. When AES blinks: introducing optical side channel. *IET Information Security*, 2(3):94–98, 2008.

[74] Pierre-Alain Fouque, Denis Réal, Frédéric Valette, and M'hamed Drissi. The Carry Leakage on the Randomized Exponent Countermeasure. In Oswald and Rohatgi [174], pages 198–213.

[75] Pierre-Alain Fouque and Frédéric Valette. The Doubling Attack - *Why Upwards Is Better than Downwards*. In Walter et al. [222], pages 269–280.

[76] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology — CRYPTO 1984*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer-Verlag, 1985.

[77] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In Çetin Kaya Koç et al. [39], pages 251–261.

[78] Catherine H. Gebotys, Simon Ho, and C. C. Tiu. EM Analysis of Rijndael and ECC on a Wireless Java-Based PDA. In Rao and Sunar [185], pages 250–264.

[79] Benedikt Gierlichs, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. Revisiting Higher-Order DPA Attacks: Multivariate Mutual Information Analysis. In Josef Pieprzyk, editor, *Topics in Cryptology — CT-RSA 2010*, volume 5985 of *Lecture Notes in Computer Science*, pages 221–234. Springer-Verlag, 2010.

[80] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis. In Oswald and Rohatgi [174], pages 426–442.

[81] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. Templates vs. Stochastic Methods. In Goubin and Matsui [84], pages 15–29.

[82] Benedikt Gierlichs, Elke De Mulder, Bart Preneel, and Ingrid Verbauwhede. Empirical Comparison of Side Channel Analysis Distinguishers on DES in Hardware. In *European Conference on Circuit Theory and Design — ECCTD 2009*, pages 391–394. IEEE, 2009.

[83] Damien Giry. BlueKrypt — Cryptographic Key Length Recommendation. `http://www.keylength.com/`, 2013. Online; accessed 25-Oct-2013.

[84] Louis Goubin and Mitsuru Matsui, editors. *Cryptographic Hardware and Embedded Systems — CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006.

[85] Louis Goubin and Jacques Patarin. DES and Differential Power Analysis (The "Duplication" Method). In Çetin Kaya Koç and Paar [40], pages 158–172.

[86] Peter J. Green, Richard Noad, and Nigel P. Smart. Further Hidden Markov Model Cryptanalysis. In Rao and Sunar [185], pages 61–74.

[87] Johann Großschädl, Elisabeth Oswald, Dan Page, and Michael Tunstall. Side-Channel Analysis of Cryptographic Software via Early-Terminating Multiplications. Cryptology ePrint Archive, Report 2009/538, 2009. `http://eprint.iacr.org/`.

[88] Gaël Hachez and Jean-Jacques Quisquater. Montgomery Exponentiation with no Final Subtractions: Improved Results. In Paar and Çetin Kaya Koç [175], pages 293–301.

[89] Helena Handschuh and Bart Preneel. Blind Differential Cryptanalysis for Enhanced Power Attacks. In Eli Biham and Amr M. Youssef, editors, *Selected Areas in Cryptography — SAC 2006*, volume 4356 of *Lecture Notes in Computer Science*, pages 163–173. Springer-Verlag, 2007.

[90] Neil Hanley, HeeSeok Kim, and Michael Tunstall. Exploiting Collisions in Addition Chain-based Exponentiation Algorithms Using a Single Trace. Cryptology ePrint Archive, Report 2012/485, 2012. `http://eprint.iacr.org/`.

[91] Neil Hanley, Maire O'Neill, Michael Tunstall, and William P. Marnane. Empirical Evaluation of Multi-Device Profiling Side-Channel Attacks. In *IEEE International Workshop on Signal Processing Systems — SiPS 2014*. IEEE Signal Processing Society, 2014.

[92] Neil Hanley, Michael Tunstall, and William P. Marnane. Unknown Plaintext Template Attacks. In Youm and Yung [234], pages 148–162.

[93] Neil Hanley, Michael Tunstall, and William P. Marnane. Using Templates to Distinguish Multiplications from Squaring Operations. *International Journal of Information Security*, 10(4):255–266, 2011.

[94] Fredric J. Harris. On the use of Windows for Harmonic Analysis with the Discrete Fourier Transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.

[95] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer-Verlag, 2009.

[96] Hera He, Josh Jaffe, and Long Zou. Side Channel Cryptanalysis Using Machine Learning : Using an SVM to recover DES keys from a smart card. `http://cs229.stanford.edu/proj2012/HeJaffeZou-SideChannelCryptanalysisUsingMachineLearning.pdf`, 2012. Online; accessed 28-Nov-2013.

[97] Christoph Herbst and Marcel Medwed. Using Templates to Attack Masked Montgomery Ladder Implementations of Modular Exponentiation. In Chung et al. [46], pages 1–13.

[98] Annelie Heuser, Olivier Rioul, and Sylvain Guilley. Good is Not Good Enough: Deriving Optimal Distinguishers from Communication Theory. Cryptology ePrint Archive, Report 2014/527, 2014. `http://eprint.iacr.org/`.

[99] Annelie Heuser and Michael Zohner. Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Werner Schindler and Sorin A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design — COSADE 2012*, volume 7275 of *Lecture Notes in Computer Science*, pages 249–264. Springer-Verlag, 2012.

[100] Johann Heyszl, Andreas Ibing, Stefan Mangard, Fabrizio De Santis, and Georg Sigl. Clustering Algorithms for Non-Profiled Single-Execution on Exponentiations. Cryptology ePrint Archive, Report 2013/438, 2013. `http://eprint.iacr.org/`.

[101] Johann Heyszl, Stefan Mangard, Benedikt Heinz, Frederic Stumpf, and Georg Sigl. Localized Electromagnetic Analysis of Cryptographic Implementations. In Dunkelman [64], pages 231–244.

[102] Naofumi Homma, Atsushi Miyamoto, Takafumi Aoki, Akashi Satoh, and Adi Shamir. Collision-Based Power Analysis of Modular Exponentiation Using Chosen-Message Pairs. In Oswald and Rohatgi [174], pages 15–29.

[103] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293–302, 2011.

[104] Gabriel Hospodar, Elke De Mulder, Benedikt Gierlichs, Joos Vandewalle, and Ingrid Verbauwhede. Least Squares Support Vector Machines for Side-Channel Analysis. In Schindler and Huss [194].

[105] Joshua Jaffe. A First-Order DPA Attack Against AES in Counter Mode with Unknown Initial Counter. In Paillier and Verbauwhede [177], pages 1–13.

[106] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R*. Springer-Verlag, 2013.

[107] Thorsten Joachims. SVM$^{\text{light}}$. `http://svmlight.joachims.org/`, 1998. Online; accessed 23-Mar-2011.

[108] Thorsten Joachims. Making large-Scale SVM Learning Practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, 1999.

[109] Thomas Johansson, editor. *Fast Software Encryption — FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[110] Marc Joye. Highly Regular Right-to-Left Algorithms for Scalar Multiplication. In Paillier and Verbauwhede [177], pages 135–147.

[111] Marc Joye and Jean-Jacques Quisquater, editors. *Cryptographic Hardware and Embedded Systems — CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[112] Marc Joye and Michael Tunstall. Exponent recoding and regular exponentiation algorithms. In Bart Preneel, editor, *Progress in Cryptology — AFRICACRYPT 2009*,

volume 5580 of *Lecture Notes in Computer Science*, pages 334–349. Springer-Verlag, 2009.

[113] Marc Joye and Sung-Ming Yen. The Montgomery Powering Ladder. In Kaliski et al. [115], pages 291–302.

[114] Mohaned Kafi, Sylvain Guilley, Sandra Marcello, and David Naccache. Deconvolving Protected Signals. In *Conference on Availability, Reliability and Security — ARES 2009*, pages 687–694. IEEE Computer Society, 2009.

[115] Burton S. Kaliski, Çetin Kaya Koç, and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems — CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[116] Chris Karlof and David Wagner. Hidden Markov Model Cryptoanalysis. In Walter et al. [222], pages 17–34.

[117] Michael Kasper, Werner Schindler, and Marc Stöttinger. A stochastic method for security evaluation of cryptographic FPGA implementations. In Jinian Bian, Qiang Zhou, Peter Athanas, Yajun Ha, and Kang Zhao, editors, *Field-Programmable Technology — FPT 2010*, pages 146–153. IEEE, 2010.

[118] Timo Kasper. *Security Analysis of Pervasive Wireless Devices : Physical and Protocol Attacks in Practice*. PhD thesis, Ruhr-University Bochum, September 2011.

[119] Timo Kasper, David Oswald, and Christof Paar. EM Side-Channel Attacks on Commercial Contactless Smartcards using Low-Cost Equipment. In Youm and Yung [234], pages 79–93.

[120] Aggelos Kiayias, editor. *Topics in Cryptology — CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*. Springer-Verlag, 2011.

[121] Ilya Kizhvatov. Side Channel Analysis of AVR XMEGA Crypto Engine. In Dimitrios N. Serpanos and Wayne Wolf, editors, *Workshop on Embedded Systems Security — WESS 2009*. ACM, 2009.

[122] Lars R. Knudsen and Huapeng Wu, editors. *Selected Areas in Cryptography — SAC 2012*, volume 7707 of *Lecture Notes in Computer Science*. Springer-Verlag, 2013.

[123] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[124] Cetain Kaya Koç, Tolga Acar, and Burton Stephen Kaliski Jr. Analyzing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996.

[125] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and other Systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 1996.

[126] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Wiener [228], pages 388–397.

[127] Francois Koeune and Jean-Jacques Quisquater. A timing attack against Rijndael. UCL Technical Report CG-1999/1, 1999.

[128] Sandeep S. Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer, and Manfred Schimmler. Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker. In Goubin and Matsui [84], pages 101–118.

[129] RSA Laboratories. PKCS #1 v2.2: RSA Cryptography Standard. `http://www.emc.com/emc-plus/rsa-labs/index.htm`, 2012. Online; accessed 29-Oct-2013.

[130] Xuejia Lai and James L. Massey. A Proposal for a New Block Encryption Standard. In Ivan Damgård, editor, *Advances in Cryptology — EUROCRYPT 1990*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer-Verlag, 1991.

[131] Thanh-Ha Le, Jessy Clédière, Cécile Canovas, Bruno Robisson, Christine Servière, and Jean-Louis Lacoume. A Proposition for Correlation Power Analysis Enhancement. In Goubin and Matsui [84], pages 174–186.

[132] Thanh-Ha Le, Jessy Clédière, Christine Servière, and Jean-Louis Lacoume. Noise Reduction in Side Channel Attack Using Fourth-Order Cumulant. *IEEE Transactions on Information Forensics and Security*, 2(4):710–720, 2007.

[133] Kerstin Lemke, Kai Schramm, and Christof Paar. DPA on n-Bit Sized Boolean and Arithmetic Operations and Its Application to IDEA, RC6, and the HMAC-Construction. In Joye and Quisquater [111], pages 205–219.

[134] Kerstin Lemke-Rust and Christof Paar. Gaussian Mixture Models for Higher-Order Side Channel Analysis. In Paillier and Verbauwhede [177], pages 14–27.

[135] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. Side channel attack: an approach based on machine learning. In Schindler and Huss [194], pages 29–41.

[136] Liran Lerman, Gianluca Bontempi, Souhaib Ben Taieb, and Olivier Markowitch. A Time Series Approach for Profiling Attack. In Benedikt Gierlichs, Sylvain Guilley, and Debdeep Mukhopadhyay, editors, *Security, Privacy, and Applied Cryptography Engineering — SPACE 2013*, volume 8204 of *Lecture Notes in Computer Science*, pages 75–94. Springer-Verlag, 2013.

[137] Liran Lerman, Stephane Fernandes Medeiros, Nikita Veshchikov, Cédric Meuter, Gianluca Bontempi, and Olivier Markowitch. Semi-Supervised Template Attack. In Prouff [183], pages 184–199.

[138] ARM Limited. ARM7TDMI Technical Reference Manual (Revision r4p1). `http://infocenter.arm.com/`, 2004. Online; accessed 05-Feb-2013.

[139] Victor Lomné, Emmanuel Prouff, and Thomas Roche. Behind the Scene of Side Channel Attacks. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology — ASIACRYPT 2013*, volume 8269 of *Lecture Notes in Computer Science*, pages 506–525. Springer-Verlag, 2013.

[140] François Macé, François-Xavier Standaert, and Jean-Jacques Quisquater. Information Theoretic Evaluation of Side-Channel Resistant Logic Styles. In Paillier and Verbauwhede [177], pages 427–442.

[141] Houssem Maghrebi, Olivier Rioul, Sylvain Guilley, and Jean-Luc Danger. Comparison between side-channel analysis distinguishers. In Tat Wing Chim and Tsz Hon Yuen, editors, *Information and Communications Security — ICICS 2012*, volume 7618 of *Lecture Notes in Computer Science*, pages 331–340. Springer-Verlag, 2012.

[142] Stefan Mangard. Secure Implementation of Cryptographic Algorithms. `http://physical-security.org/`. Online; accessed 25-Nov-2013.

[143] Stefan Mangard. A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion. In Pil Joong Lee and Chae Hoon Lim, editors, *Information Security and Cryptology — ICISC 2002*, volume 2587 of *Lecture Notes in Computer Science*, pages 343–358. Springer-Verlag, 2002.

[144] Stefan Mangard, editor. *International Conference on Smart Card Research and Advanced Applications — CARDIS 2012*, volume 7771 of *Lecture Notes in Computer Science*. Springer-Verlag, 2013.

[145] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer-Verlag, 2007.

[146] Stefan Mangard, Elisabeth Oswald, and François-Xavier Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Information Security*, 5(2):100–110, 2011.

[147] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully Attacking Masked AES Hardware Implementations. In Rao and Sunar [185], pages 157–171.

[148] Marcel Medwed and Elisabeth Oswald. Template Attacks on ECDSA. In Chung et al. [46], pages 14–27.

[149] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[150] Thomas S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In Paar and Çetin Kaya Koç [175], pages 238–251.

[151] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Examining Smart-Card Security under the Threat of Power Analysis Attacks. *IEEE Transactions on Computers*, 51(5):541–552, 2002.

[152] Victor S. Miller. Use of Elliptic Curves in Cryptography. In Hugh C. Williams, editor, *Advances in Cryptology — CRYPTO 1985*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer-Verlag, 1986.

[153] Peter L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, 1985.

[154] David P. Montminy, Rusty O. Baldwin, Michael A. Temple, and Eric D. Laspe. Improving cross-device attacks using zero-mean unit-variance normalization. *Journal of Cryptographic Engineering*, 3(2):99–110, 2013.

[155] Amir Moradi, Markus Kasper, and Christof Paar. Black-Box Side-Channel Attacks Highlight the Importance of Countermeasures - An Analysis of the Xilinx Virtex-4 and Virtex-5 Bitstream Encryption Mechanism. In Dunkelman [64], pages 1–18.

[156] Ruben A. Muijrers, Jasper G. J. van Woudenberg, and Lejla Batina. RAM: Rapid Alignment Method. In Prouff [182], pages 266–282.

[157] Elke De Mulder, Michael Hutter, Mark E. Marson, and Peter Pearson. Using Bleichenbacher's Solution to the Hidden Number Problem to Attack Nonce Leaks in 384-Bit ECDSA. In Bertoni and Coron [26], pages 435–452.

[158] Andrew Ng. Machine Learning. https://www.coursera.org/, 2013. Online; accessed 01-Mar-2013.

[159] Minh Hoai Nguyen and Fernando De la Torre. Optimal Feature Selection for Support Vector Machines. *Pattern Recognition*, 43(3):584–591, 2010.

[160] Phong Q. Nguyen and Igor Shparlinski. The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces. *Designs, Codes and Cryptography*, 30(2):201–217, 2003.

[161] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security — ICICS 2006*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer-Verlag, 2006.

[162] NIST. FIPS-46-3: Data Encryption Standard (DES). National Institute of Standards and Technology publication, 1999. http://www.nist.gov/publication-portal.cfm.

[163] NIST. FIPS-197: Advanced Encryption Standard (AES). National Institute of Standards and Technology publication, 2001. http://www.nist.gov/publication-portal.cfm.

[164] NIST. FIPS 180-3: Secure Hash Standard. National Institute of Standards and Technology publication, 2008. http://www.nist.gov/publication-portal.cfm.

[165] NIST. FIPS-186-3: Digital Signature Standard (DSS). National Institute of Standards and Technology publication, 2009. http://www.nist.gov/publication-portal.cfm.

[166] Colin O'Flynn. Power Analysis for Cheapskates. Blackhat Abu Dhabi, 2012.

[167] Colin O'Flynn and Zhizhang (David) Chen. Synchronous Sampling and Clock Recovery of Internal Oscillators for Side Channel Analysis. Cryptology ePrint Archive, Report 2013/294, 2013. http://eprint.iacr.org/.

[168] Yossef Oren, Mathieu Renauld, François-Xavier Standaert, and Avishai Wool. Algebraic Side-Channel Attacks Beyond the Hamming Weight Leakage Model. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems — CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 140–154. Springer-Verlag, 2012.

[169] David Oswald and Christof Paar. Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems — CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 207–222. Springer-Verlag, 2011.

[170] David Oswald and Christof Paar. Improving Side-Channel Analysis with Optimal Linear Transforms. In Mangard [144], pages 219–233.

[171] David Oswald, Bastian Richter, and Christof Paar. Side-Channel Attacks on the Yubikey 2 One-Time Password Generator. In Salvatore J. Stolfo, Angelos Stavrou, and Charles V. Wright, editors, *Research in Attacks, Intrusions, and Defenses — RAID 2013*, volume 8145 of *Lecture Notes in Computer Science*, pages 204–222. Springer-Verlag, 2013.

[172] Elisabeth Oswald. Enhancing Simple Power-Analysis Attacks on Elliptic Curve Cryptosystems. In Kaliski et al. [115], pages 82–97.

[173] Elisabeth Oswald and Stefan Mangard. Template Attacks on Masking — Resistance is Futile. In Masayuki Abe, editor, *Topics in Cryptology — CT-RSA 2007*, volume 4377 of *Lecture Notes in Computer Science*, pages 243–256. Springer-Verlag, 2007.

[174] Elisabeth Oswald and Pankaj Rohatgi, editors. *Cryptographic Hardware and Embedded Systems — CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*. Springer-Verlag, 2008.

[175] Christof Paar and Çetin Kaya Koç, editors. *Cryptographic Hardware and Embedded Systems — CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[176] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer-Verlag, 1999.

[177] Pascal Paillier and Ingrid Verbauwhede, editors. *Cryptographic Hardware and Embedded Systems — CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

[178] Eric Peeters, François-Xavier Standaert, Nicolas Donckers, and Jean-Jacques Quisquater. Improved Higher-Order Side-Channel Attacks with FPGA Experiments. In Rao and Sunar [185], pages 309–323.

[179] Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Power and electromagnetic analysis: Improved model, consequences and comparisons. *Integration*, 40(1):52–60, 2007.

[180] John C. Platt. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. In A.J. Smola, P. Bartlett, B. Schoelkopf, and D. Schuurman, editors, *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

[181] Emmanuel Prouff. DPA Attacks and S-Boxes. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption — FSE 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 424–441. Springer-Verlag, 2005.

[182] Emmanuel Prouff, editor. *International Conference on Smart Card Research and Advanced Applications — CARDIS 2011*, volume 7079 of *Lecture Notes in Computer Science*. Springer-Verlag, 2011.

[183] Emmanuel Prouff, editor. *Constructive Side-Channel Analysis and Secure Design — COSADE 2013*, volume 7864 of *Lecture Notes in Computer Science*. Springer-Verlag, 2013.

[184] Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In Isabelle Attali and Thomas P. Jensen, editors, *Smart Card Programming and Security — E-Smart 2001*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer-Verlag, 2001.

[185] Josyula R. Rao and Berk Sunar, editors. *Cryptographic Hardware and Embedded Systems — CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[186] Christian Rechberger and Elisabeth Oswald. Practical Template Attacks. In Chae Hoon Lim and Moti Yung, editors, *Information Security Applications — WISA 2004*, volume 3325 of *Lecture Notes in Computer Science*, pages 440–456. Springer-Verlag, 2004.

[187] Mathieu Renauld, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Algebraic Side-Channel Attacks on the AES: Why Time also Matters in DPA. In Clavier and Gaj [50], pages 97–111.

[188] Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. A Formal Study of Power Variability Issues and Side-

Channel Attacks for Nanoscale Devices. In Kenneth G. Paterson, editor, *Advances in Cryptology — EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 109–128. Springer-Verlag, 2011.

[189] Mathieu Renauld and Francois-Xavier Standaert. Algebraic Side-Channel Attacks. Cryptology ePrint Archive, Report 2009/279, 2009. `http://eprint.iacr.org/`.

[190] Research Center for Information Security. Side-channel Attack Standard Evaluation Board (SASEBO). `http://www.risec.aist.go.jp/project/sasebo/`. Online; accessed 05-Feb-2013.

[191] Ronald L. Rivest. Cryptography and Machine Learning. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT 1991*, volume 739 of *Lecture Notes in Computer Science*, pages 427–439. Springer-Verlag, 1993.

[192] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[193] Tanja Römer and Jean-Pierre Seifert. Information Leakage Attacks against Smart Card Implementations of the Elliptic Curve Digital Signature Algorithm. In Isabelle Attali and Thomas P. Jensen, editors, *Smart Card Programming and Security — E-smart 2001*, volume 2140 of *Lecture Notes in Computer Science*, pages 211–219. Springer-Verlag, 2001.

[194] Werner Schindler and Sorin A. Huss, editors. *Constructive Side-Channel Analysis and Secure Design — COSADE 2011*, 2011.

[195] Werner Schindler, Kerstin Lemke, and Christof Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In Rao and Sunar [185], pages 30–46.

[196] Kai Schramm, Thomas J. Wollinger, and Christof Paar. A New Class of Collision Attacks and Its Application to DES. In Johansson [109], pages 206–222.

[197] Claude Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423, 1948.

[198] Nigel Smart. *Cryptography: An Introduction (3rd Edition)*. `http://www.cs.bris.ac.uk/~nigel/Crypto_Book/`, 2013.

[199] Nigel Smart, Elisabeth Oswald, and Dan Page. Randomised representations. *Proceedings on Information Security*, 2(2):19–27, 2008.

[200] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997.

[201] Youssef Souissi, Shivam Bhasin, Sylvain Guilley, Maxime Nassar, and Jean-Luc Danger. Towards Different Flavors of Combined Side Channel Attacks. In Dunkelman [64], pages 245–259.

[202] Youssef Souissi, Sylvain Guilley, Jean-Luc Danger, Sami Mekki, Guillaume Duc, and Guillaume Duc. Improvement of power analysis attacks using Kalman filter. In *Conference on Acoustics, Speech, and Signal Processing — ICASSP 2010*, pages 1778–1781. IEEE, 2010.

[203] François-Xavier Standaert and Cédric Archambeau. Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages. In Oswald and Rohatgi [174], pages 411–425.

[204] François-Xavier Standaert, Benedikt Gierlichs, and Ingrid Verbauwhede. Partition vs. Comparison Side-Channel Distinguishers: An Empirical Evaluation of Statistical Tests for Univariate Side-Channel Attacks against Two Unprotected CMOS Devices. In Pil Joong Lee and Jung Hee Cheon, editors, *Information Security and Cryptology — ICISC 2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 253–267. Springer-Verlag, 2009.

[205] François-Xavier Standaert, François Koeune, and Werner Schindler. How to Compare Profiled Side-Channel Attacks? In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *Applied Cryptography and Network Security — ACNS 2009*, volume 5536 of *Lecture Notes in Computer Science*, pages 485–498. Springer-Verlag, 2009.

[206] François-Xavier Standaert, Tal Malkin, and Moti Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In Antoine Joux, editor, *Advances in Cryptology — EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer-Verlag, 2009.

[207] François-Xavier Standaert, Eric Peeters, Gaël Rouvroy, and Jean-Jacques Quisquater. An Overview of Power Analysis Attacks Against Field Programmable Gate Arrays. *Proceedings of the IEEE*, 94(2):383–394, 2006.

[208] François-Xavier Standaert, Nicolas Veyrat-Charvillon, Elisabeth Oswald, Benedikt Gierlichs, Marcel Medwed, Markus Kasper, and Stefan Mangard. The World Is Not Enough: Another Look on Second-Order DPA. In Masayuki Abe, editor, *Advances in Cryptology — ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 112–129. Springer-Verlag, 2010.

[209] Takeshi Sugawara, Naofumi Homma, Takafumi Aoki, and Akashi Satoh. Profiling attack using multivariate regression analysis. *IEICE Electronics Express*, 7(15):1139–1144, 2010.

[210] Johan A. K. Suykens and Joos Vandewalle. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[211] TELECOM ParisTech SEN research group. DPA Contest. `http://www.dpacontest.org/v3/index.php`, 2011. Version 3.

[212] Andriy Temko. *Acoustic Event Detection and Classification*. PhD thesis, Universitat Politcnica de Catalunya, December 2007.

[213] Kris Tiri and Ingrid Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *Design, Automation and Test in Europe Conference and Exposition — DATE 2004*, pages 246–251. IEEE Computer Society, 2004.

[214] Michael Tunstall, Neil Hanley, Robert P. McEvoy, Claire Whelan, Colin C. Murphy, and William P. Marnane. Correlation Power Analysis of Large Word Sizes. In *Irish Signals and System Conference — ISSC 2007*, pages 145–150. IET, 2007.

[215] Wim van Eck. Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk? *Computers & Security*, 4(4):269–286, 1985.

[216] Jasper G. J. van Woudenberg, Marc F. Witteman, and Bram Bakker. Improving Differential Power Analysis by Elastic Alignment. In Kiayias [120], pages 104–119.

[217] Alexandre Venelli. Analysis of Nonparametric Estimation Methods for Mutual Information Analysis. In Kyung Hyune Rhee and DaeHun Nyang, editors, *Information Security and Cryptology — ICISC 2010*, volume 6829 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2011.

[218] Colin D. Walter. Montgomery Exponentiation Needs no Final Subtractions. *Electronic Letters*, 35(21):1831–1832, 1999.

[219] Colin D. Walter. Sliding Windows Succumbs to Big Mac Attack. In Çetin Kaya Koç et al. [39], pages 286–299.

[220] Colin D. Walter. Longer Keys May Facilitate Side Channel Attacks. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography — SAC 2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 42–57. Springer-Verlag, 2004.

[221] Colin D. Walter. Simple Power Analysis of Unified Code for ECC Double and Add. In Joye and Quisquater [111], pages 191–204.

[222] Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems — CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[223] Colin D. Walter and David Samyde. Data Dependent Power Use in Multipliers. In *IEEE Symposium on Computer Arithmetic — ARITH 2005*, pages 4–12. IEEE Computer Society, 2005.

[224] Colin D. Walter and Susan Thompson. Distinguishing Exponent Digits by Observing Modular Subtractions. In David Naccache, editor, *Topics in Cryptology - CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 192–207. Springer-Verlag, 2001.

[225] Carolyn Whitnall and Elisabeth Oswald. A fair evaluation framework for comparing side-channel distinguishers. *Journal of Cryptographic Engineering*, 1(2):145–160, 2011.

[226] Carolyn Whitnall and Elisabeth Oswald. Profiling DPA: Efficacy and Efficiency Trade-Offs. In Bertoni and Coron [26], pages 37–54.

[227] Carolyn Whitnall, Elisabeth Oswald, and Luke Mather. An Exploration of the Kolmogorov-Smirnov Test as a Competitor to Mutual Information Analysis. In Prouff [182], pages 234–251.

[228] Michael J. Wiener, editor. *Advances in Cryptology — CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[229] Erik De Win, Serge Mister, Bart Preneel, and Michael J. Wiener. On the Performance of Signature Schemes Based on Elliptic Curves. In Joe Buhler, editor, *Algorithmic Number Theory — ANTS 1998*, volume 1423 of *Lecture Notes in Computer Science*, pages 252–266. Springer-Verlag, 1998.

[230] Marc F. Witteman, Jasper G. J. van Woudenberg, and Federico Menarini. Defeating RSA Multiply-Always and Message Blinding Countermeasures. In Kiayias [120], pages 77–88.

[231] Xilinx. Virtex-II Platform FPGAs: Complete Data Sheet. `http://www.xilinx.com/support/documentation/virtex-ii_data_sheets.htm`. Online; accessed 05-Feb-2013.

[232] Sung-Ming Yen and Marc Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000.

[233] Sung-Ming Yen, Wei-Chih Lien, Sang-Jae Moon, and JaeCheol Ha. Power Analysis by Exploiting Chosen Message and Internal Collisions - Vulnerability of Checking Mechanism for RSA-Decryption. In Ed Dawson and Serge Vaudenay, editors, *International Conference on Cryptology in Malaysia — Mycrypt 2005*, volume 3715 of *Lecture Notes in Computer Science*, pages 183–195. Springer-Verlag, 2005.

[234] Heung Youl Youm and Moti Yung, editors. *Information Security Applications — WISA 2009*, volume 5932 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[235] Li Zhuang, Feng Zhou, and J. D. Tygar. Keyboard acoustic emanations revisited. *ACM Transactions on Information and System Security*, 13(1), 2009.

Appendix A

Advanced Encryption Standard

In 2001, the block cipher Rijndael by Joan Daemen and Vincent Rijmen, was selected by NIST to become AES [163] as a replacement for the outdated DES algorithm. It is an substitution-permutation network (SPN) based iterative block cipher which acts on plaintext blocks of 128-bits and supports significantly larger key sizes than DES, *i.e.* 128-bits, 192-bits or 256-bits. Depending on the key size, the number of rounds is either 10, 12 or 14 respectively. For the work in this thesis, only the 128-bit key size is used.

Algorithm A.1: Advanced Encryption Standard.

Input: Plaintext p , 128-bit Key s

Output: Ciphertext c

```
1 state ← p ;
2 state ← AddRoundKey(state, s_0) ;
3 for i = 1 to 10 do
4 |   state ← S-Box(state) ;
5 |   state ← ShiftRows(state) ;
6 |   if i ≠ 10 then state ← MixColumns(state) ;
7 |   state ← AddRoundKey(state, s_i) ;
8 end
9 return c ←state
```

Algorithm A.1 outlines a high-level description of the AES algorithm. First, the plaintext block p is copied into the state variable, which is a 4×4 matrix of bytes. Then, an initial *AddRoundKey* function simply XORs the initial key to the state. This is followed by nine identical round transformations which include the steps *S-Box*, *ShiftRows*, *MixColumns*, and *AddRoundKey*. The tenth round skips the *MixColumns* operation to generate the

ciphertext block c.

***AddRoundKey*:** Each byte of the state array is XORed with the corresponding byte of a sub-key that is generated from the secret key. The key applied in the first *AddRoundKey* operation is the secret key, and subsequent sub-keys are generated using a key schedule.

***S-Box*:** Each byte of the state undergoes a non-linear byte substitution by combining its multiplicative inverse in $GF(2^8)$ with an invertible affine transformation. Typically, the *S-Box* operation is pre-computed and stored in a 256-entry table. In this way, the *S-Box* operation becomes a simple table look-up.

***ShiftRows*:** This operation cyclically shifts the last three rows of the state by different offsets.

***MixColumns*:** As the name implies, this operation acts on the columns of the state individually. The four bytes of each column are mixed by applying a linear transformation.

Appendix B

Elliptic Curve Digital Signature Algorithm

ECDSA [165] provides an elliptic curve varient of the digital signature standard. This allows a smaller key size for a given security level compared to DSA. The system parameters are $\{\mathcal{E}, \mathbf{G}, n\}$ where $\mathbf{G}$ is a point of prime order n on the elliptic curve $\mathcal{E}$. The signer has a private key d (randomly chosen from $[1, n-1]$) and public key $\mathbf{E}$ where $\mathbf{E} = [d]\mathbf{G}$. H is a suitable hash function such as SHA-256 [164].

Signature: To sign a message m, the signer picks a random $0 < k < n$ and computes:

$$r \leftarrow x \bmod n \quad \text{where} \quad (x, y) = [k]\,\mathbf{G}$$
$$s \leftarrow k^{-1}\,(H(m) + r\,d) \bmod n$$

The signature of m is the pair: $\{r, s\}$.

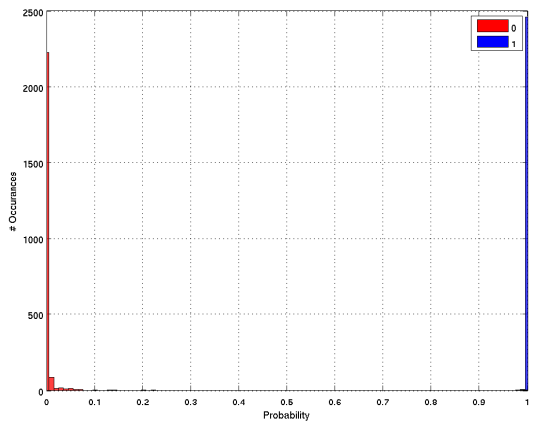
Verification: To check $\{r, s\}$ the verifier ascertains that:

$$r \stackrel{?}{=} x \bmod n \quad \text{where} \quad (x, y) = [u_1]\,\mathbf{G} + [u_2]\,\mathbf{E}\,,$$
$$u_1 = H(m)\,s^{-1} \bmod n \quad \text{and} \quad u_2 = r\,s^{-1} \bmod n\,.$$

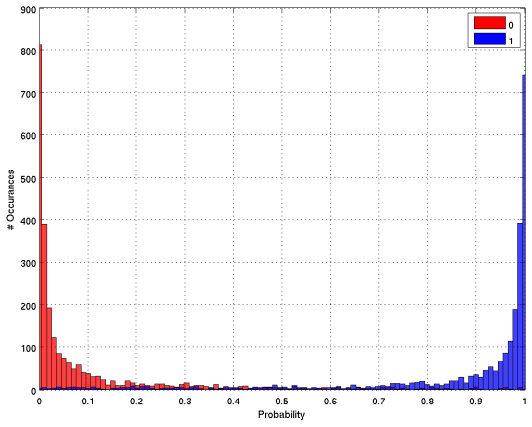
Appendix C

Logistic Regression Bit Analysis

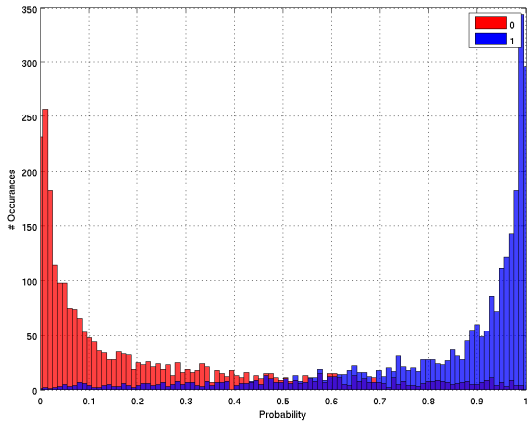
The leakage of each bit of the AES *S-Box* contributes differing amounts to the overall side channel leakage. Hence when training a LR classifier on the individual bits, (using 25 k training traces and 100 features) there is a large divergence between the error rates. Histograms of the classification probabilities for each bit are plotted below, with any red samples having a probability greater than 0.5 incorrectly classified, and any blue sample less than 0.5 incorrect. It is clear that bit 5, and to a lesser extent bits 3 & 7 have poor classification accuracy which contributes to the disappointing byte classification in §6.3.2.



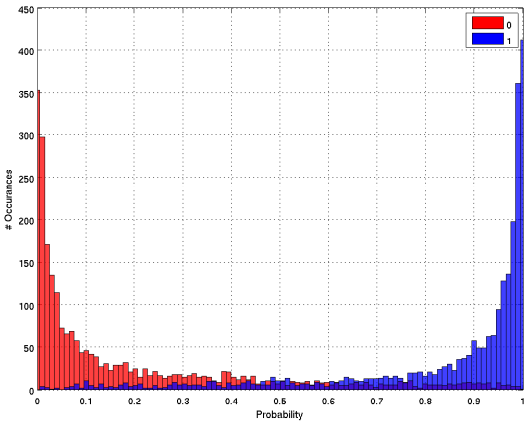
(a) S-Box output - Bit 1.



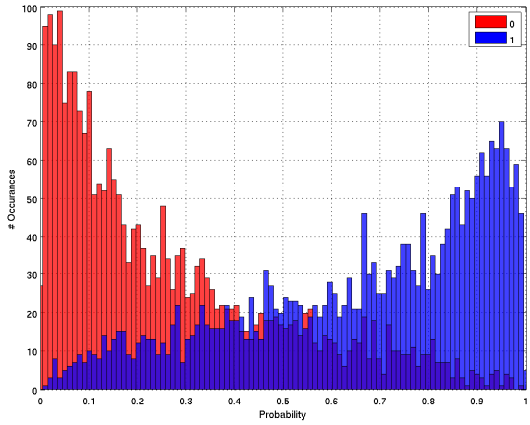
(b) S-Box output - Bit 2.



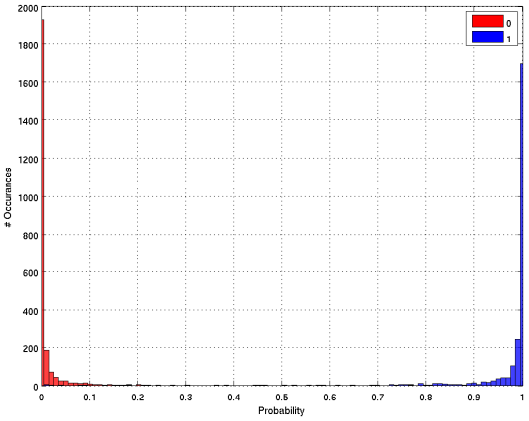
(c) S-Box output - Bit 3.



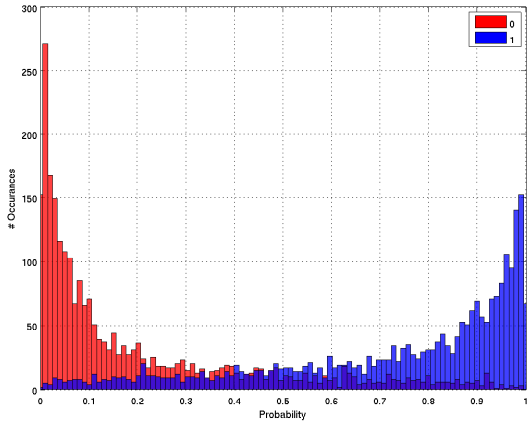
(d) S-Box output - Bit 4.



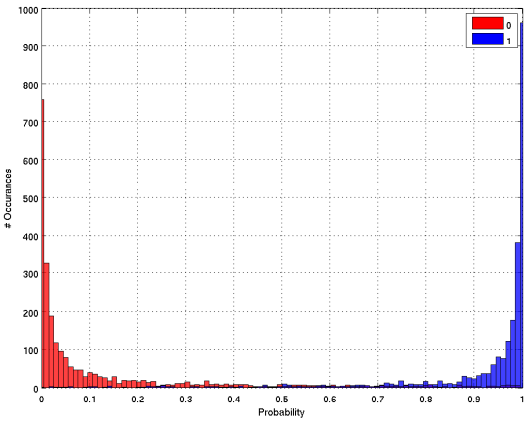
(e) S-Box output - Bit 5.



(f) S-Box output - Bit 6.



(g) S-Box output - Bit 7.



(h) S-Box output - Bit 8.

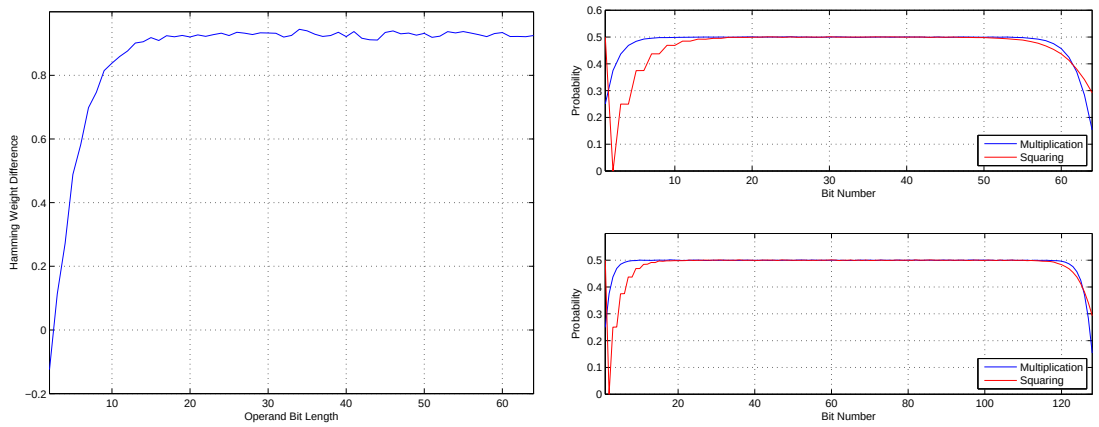
Figure C.1: Analysis of individual bit leakages with LR.

Appendix D

Multiplier Hamming Weight Difference

The calculation of Equation 5.5 and Equation 5.6 in §5.4 is practically feasible for up to $l \approx 16$. For $l \geq 16$ the computational requirements begin to become excessive, and for $l \approx 32$ or larger its not practical to evaluate over all possible inputs.

To illustrate that side-channel leakage due the difference in Hamming weight of multiplication and squaring operations will also present for larger operand bit-lengths, the experiment was re-run using 1M random, uniformly distributed inputs for bit-lengths up to 64. As can be seen in Figure D.1(a), for input bit-lengths greater than about 20 the Hamming weight difference no longer increases. Looking at Figure D.1(b), it is clear from both the 32 and 64-bit graphs, as well as the 16-bit graph in Figure 5.1(b), that the leakage only occurs at the upper and lower bits, and the middle bits are equally likely to be 0 or 1.



(a) Hamming weight difference for operand bit lengths up to 64.

(b) Probability that an output bit is 1 for 32-bit (upper) and 64-bit (lower) operands.

Figure D.1: Hamming weight analysis.