NOVEL COUNTERMEASURES AND TECHNIQUES FOR DIFFERENTIAL POWER ANALYSIS

By

John Goodwin

A thesis submitted for the degree of Doctor of Philosophy

School of Electronics and Computer Science,

University of Southampton,

United Kingdom.

August, 2009

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING

SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

Novel Countermeasures and Techniques for Differential Power Analysis

by John Goodwin

Research in the last few years has indicated that, despite modern algorithms being secure against all published mathematical attacks and being far too complex to break by brute force, secret key data can be gathered by monitoring the power consumption. This is known as a *power analysis attack*, the most successful has been *differential power analysis* (DPA). Several countermeasures have been proposed for preventing power analysis attacks with varying degrees of efficacy. One thing all the countermeasures have in common is their large cost in terms of performance and or cost. In this thesis several modifications to the AES algorithm are proposed that seek to inherently secure it against DPA and their effectiveness and cost are investigated.

Due to the statistical nature of DPA there is no set amount of power consumption data that will always give the correct result for a given device, rather, a value for the SNR and the number of power measurements involved in the attack will equate to a probability of success. In this thesis a statistical model of the DPA attack is derived and it is used to find a method for calculating the probability that a particular attack will be successful.

A more benign use for DPA is also discussed. If the signature of a specific pattern of register transitions can be detected in the power consumption of a device then designers can add hardware whose sole purpose is to be detectable in a power trace and act as a watermark to prove the presence of intellectual property.

Contents

List of Figures

List of Tables

Acknowledgements

There were lots of people who contributed to this thesis either directly or indirectly. While they all deserve my thanks, I'm only going to single a few out here. Firstly, my supervisor Peter Wilson provided me with near constant support and always gave me considered feedback and replied promptly to my more panicked emails.

I'd like to thank everyone in the ESD lab, especially Karthik Baddam, who is one of the few people other than me to care about DPA and with whom I had several excellent discussions, and Julian Bailey who showed me I can concentrate on work, even when someone is talking to me.

Finally, thanks to my parents for giving birth to me etc. and my sister, Ann Brookes, whose extensive knowledge of statistics I frequently abused.

Declaration

I hereby declare that this thesis is my own work, unless stated otherwise.

John Goodwin

Definitions of Terms and Abbreviations

AES	Advanced Encryption Standard
ANOVA	Analysis of Variance
ASIC	Application Specific Integrated Circuit
BDD	Binary Decision Diagram
CBC	Cipher Block Chaining
CDF	Cumulative Density Function
CFB	Cipher Feedback
Ciphertext	The output of an encryption algorithm
Consumption Matrix	A matrix containing the power consumption data across a number of encryption blocks
CTR	Counter
DDL	Dynamic and Differential Logic
DES	Data Encryption Standard
DFF	D-Type Flip Flop
DPA	Differential Power Analysis
DVFS	Dynamic Voltage and Frequency Switching
ECB	Electronic Code Book
EDE	Encryption-Decryption-Encryption
FPGA	Field Programmable Gate Array
HODPA	High-Order Differential Power Analysis
ICA	Independent Component Analysis
IPA	Inferential Power Analysis
IV	Initialisation Vector
KS	Kolmogorov-Smirnov

LDPA	Leakage-Based Differential Power Analysis
LUT	Look Up Table
NIST	National Institute of Standards and Technology
OFB	Output Feedback
PDF	Probability Density Function
Plaintext	The data that requires encryption
PRBG	Pseudo Random Bit Generator
Prediction Matrix	A matrix of the predictions of the number of bit transitions in a register across a number of plaintexts for all possible values of a byte of the key.
PTE	Power Trace Entropy
Round	A single instance of the iterative transforms that makes up most block ciphers.
Round Function	The function (often made up of a series of others) that forms the round
Round Key	The key data that is added on a given round
SABL	Sense Amplifier Based Logic
SAC	Strict Avalanche Criterion
SDPA	Sign-Based Differential Power Analysis
SNR	Signal-to-Noise Ratio
SPN	Substitution Permutation Network
TDES	Triple Data Encryption Standard
Trace	The power consumption data from a single encryption
TTE	Time Trace Entropy
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
WDDL	Wave Dynamic Differential Logic

Chapter 1 Introduction

1.1 Motivation

This thesis investigates methods of attacks on secret-key cryptographic systems. A current pervasive approach for symmetric encryption is the use of block ciphers and specifically the *Advanced Encryption Standard* (AES) [1]. Research in the last few years has indicated that despite modern algorithms being secure against all published mathematical attacks and being far too complex to break by brute force, secret key data can be gathered by monitoring the power consumption. This is known as a *side-channel attack* as instead of attacking the cipher in a traditional manner it is cracked using information extracted from the physical implementation of the cryptosystem. When the side channel is the power consumption it is known as a *power analysis attack*. In CMOS technology, when the value inside a register changes from 0 to 1 or 1 to 0 the power consumption is significantly higher than when the value remains constant, this leads to a large data dependence in the power consumption. The most effective power analysis attack is *differential power analysis* (DPA), it combines power consumption information from several encryptions with predictions about the transitions of register values and uses statistical evaluation to determine the most likely value of the key [2]. DPA is powerful enough to successfully retrieve the key from implementations of AES [3].

Several countermeasures have been proposed for preventing power analysis attacks, these include balancing the logic there is always the same number of bit transitions irrespective of the data that is being processed, masking data to hide intermediate values and randomly changing the supply voltage and clock frequency of the device [4-11]. The countermeasures have varying degrees of efficacy, but even the most successful ones do not offer complete protection from the attack [5]. One thing

all the countermeasures have in common is their large cost in terms of performance and or area, this is due to the fact that they all add additional hardware to try to defeat power analysis. A better approach is needed. Rather than trying to protect the implementations of cryptographic algorithms after the fact it would be preferable if algorithms were designed in such a way as to already be immune from DPA.

1.2 Objectives

It was noted in the previous section that DPA can break even the most modern algorithms and that the current techniques for protecting implementations from the attack do not offer complete security, rather they simply reduce the correlation between the data that is being processed and the power consumption. While this will make it harder for an attacker to retrieve the key, if there is any correlation a determined enough attacker, who collected enough power consumption data, could exploit it. The exact relationship between the amount of power consumption data required and the level of correlation between the data and the power consumption is not known. This leads to two objectives: the derivation of the relationship between the correlation and the level of information required to perform the attack, and the development of a new type of countermeasure that defeats DPA. These are described in more detail in the following two paragraphs.

Despite the fact that DPA was developed in 1998 and the mathematical concepts that are employed in the attack are well developed there are still elements of the attack that are poorly understood. It is clear that the greater the dependence of the power consumption on the processed data the lower the amount of power consumption data that must be recorded in order to attack the cryptographic device. There is no quantitative relationship, however, therefore it is not possible to calculate the effect of halving the correlation between the processed data and the power consumption. This is one of the aims of this thesis. It will allow the evaluation of the effects of a particular countermeasure on the effort involved in performing DPA and enable designers to tailor the amount of noise in a design to the particular security constraints of the system that they are creating.

The only countermeasures to DPA that have been proposed so far are *ad hoc* modifications to cryptographic implementations that are costly in terms of the design's speed and area requirements. Only partial protection from DPA is gained for

the significant price that is paid. The main goal of this thesis is to develop a modification to the AES algorithm that will protect it completely from DPA while still allowing efficient implementation. It will then be possible to apply the technique to other existing algorithms and, most importantly, new algorithms can be designed to be immune from DPA.

1.3 Thesis Structure

This thesis is divided into 9 chapters. Chapters 2-4 deal with the background material and chapters 5-7 cover the original work. Chapter 8 summarises the thesis and Chapter 9 concludes and suggests some further work that can be done on the topic.

Chapter 2 gives an overview of the history of cryptography and cryptanalysis.

Chapter 3 introduces the basic concepts and techniques in cryptography and the goes on to explain block ciphers in greater detail. Three important block ciphers: the *Data Encryption Standard* (DES), *Triple DES* (TDES) and the *Advanced Encryption Standard* (AES) are described in detail including security concerns surrounding their use.

Chapter 4 deals with attacks on block ciphers. This includes both the perceived weaknesses of the AES algorithm and power analysis attacks, specifically DPA. Various different approaches for the attacks are described, as are some countermeasures. The effectiveness of the attacks and countermeasures are discussed using published results of the application of power analysis to real cryptographic systems.

Chapter 5 describes the systems that were developed in the course of this research to investigate DPA. First the attack was performed in simulation using Modelsim with a post-synthesis VHDL design of AES, and Matlab with a mathematical model of the power consumption of a general crypto-device. Also an oscilloscope was used to record the power consumption of an FPGA configured to perform AES. These systems were used to investigate the basic properties of DPA such as the effect of what part of the algorithm was being targeted by the attack.

Chapter 6 gives a detailed analysis of the statistical properties of DPA and develops a model of the attack. Using this model, a method was developed to

determine the probability that a particular DPA attack will be successful given the number of power traces available to an attacker and the signal-to-noise ratio (SNR) of the crypto device being attacked. Also, another use for DPA is proposed, using a pseudo-random number generator to add a pattern to the power consumption of a design to act like a watermark that will allow the identification of intellectual property within a larger system.

In Chapter 7 a number of modifications were made to the AES algorithm with the hope of rendering it impervious to DPA. The most successful of these techniques, using the key schedule to change the round keys for each block, was then applied to TDES. While it did still protect it from DPA, an implementation of the new algorithm used significantly more resources and was slower. The structure of the key schedules of a selection of modern algorithms were analysed for their suitability for implementing the technique efficiently, and the important properties were identified. The algorithm ARIA has these properties and so an implementation of the modified algorithm was made. It performed well in terms of both protection from DPA and implementational efficiency.

Chapter 2 Background

2.1 Introduction

Since the invention of writing, people have sought to keep the nature of their most sensitive messages secret from their enemies. Failure to do so has led to lost battles, revealed secrets and the fall of monarchs. Cryptography is a constant battle between code makers and code breakers. Codes are developed and then techniques are developed to crack them, when it has become clear that a code is no longer secure new methods must be found to restore the protection that was offered previously. These new codes are inevitably eventually broken by increases in computing power, knowledge of mathematics, or the sheer determination of attackers.

Historically cryptography has been purely the domain of generals and statesmen – and Casanova[1] – but in the digital age cryptography has become ubiquitous and an important tool for everyone to send data electronically. Members of the public use modern encryption techniques, albeit sometimes unknowingly, every time they

---

1"Shall I tell you the key?"

"Pray do so."

I gave her the word, which belonged to no language that I know of, and the marchioness was quite thunderstruck.

"This is too amazing," said she; "I thought myself the sole possessor of that mysterious word--I had never written it down, laying it up in my memory--and I am sure I have never told anyone of it."

I might have informed her that the calculation which enabled me to decipher the manuscript furnished me also with the key, but the whim took me to tell her that a spirit had revealed it to me. This foolish tale completed my mastery over this truly learned and sensible woman on everything but her hobby. This false confidence gave me an immense ascendancy over Madame d'Urfe, and I often abused my power over her.[12] J. Casanova, "The Complete Memoirs of Casanova," Globusz Publishing, New.

withdraw money from a cash machine, make a mobile phone call, or even watch a DVD.

2.2 History of cryptography

2.2.1 Classical ciphers

There were two main types of cipher that existed in the ancient world, the transposition cipher and the substitution cipher. Transposition ciphers hide the meaning of a message by rearranging the characters in it, turning it into a large difficult anagram. Substitution ciphers change letters for other letters or symbols making the message unintelligible unless the reader knows how the letters have been changed.

The first military cryptographic device encoded and decoded messages using transposition, it was the Spartan Scytale which dates back to the 5th century BC [13]. A Scytale is a wooden rod of fixed diameter, a strip of leather or parchment is wound around it and the message is written across the length of the rod so that each successive letter appears on a different section of the parchment. When unwound and read down the length of the parchment the letters are mixed and can only be read by wrapping the parchment around a stick of similar diameter.

The first recorded use of substitution ciphers dates back to 600-500 BC. Invented by Hebrew scholars, it is called the Atbash cipher and substitutes the first letter of the alphabet for the last, the second for the penultimate and so on. It is used in Jewish mysticism and in some bible passages, its main purpose is probably to create an air of mystery rather than actual concealment.

The Caesar Cipher is named after Julius Caesar as he is reported to have used it when sending important military communications by Suetonius, a biographer of the first 12 emperors of Rome who was also Emperor Hadrian's personal secretary. It is a very simple substitution cipher, the alphabet is shifted a number of letters in a particular direction, for example if it was used on our alphabet and shifted two places to the right then 'a' would become 'c', 'b' would become 'd' and so on until 'z' became 'b'.

The weakness with these ciphers is that if the attacker knows the type of cipher it is then it becomes very easy to crack, with Atbash, there is only one possible set of substitutions, and with the Caesar cipher there are only 26. A slightly more complex substitution cipher is one where the letters are randomly substituted for others rather than in order. This would give a total of 26! ($4*10^{26}$) possible combinations. Even with this large number of combinations substitution ciphers are still not particularly secure as the distribution of letters in the plaintext is unhidden, this leaves it vulnerable to frequency analysis. This looks at the frequency of letters in the ciphertext and compares them to the frequencies of various letters in the language that the plaintext is written in. For example, if a ciphertext contains a large number of 'w's then trying the substitution 'e' = 'w' is a reasonable place to start. The first known record of this technique is by the 9th century Arabic scholar Abu Yusuf Yaqub ibn Ishaq al-Sabbah Al-Kindi.

Probably the most famous historical demonstration of the weakness of substitution ciphers is the Babington plot. In 1568 Mary Queen of Scots fled Scotland after a failed attempt to regain the crown of Scotland from her half-brother. She sought refuge in England on her way to France, unfortunately for her she had misjudged the mood of her cousin Queen Elizabeth, who imprisoned her. She remained confined in a series of castles and manors and in 1586 after 18 years in prison she was allowed to neither send nor receive letters. Then suddenly, a large parcel of correspondence arrived in her possession. They were smuggled into her prison by Gilbert Gifford who had placed them in a hollow bung inside a barrel of beer. It was through this channel of communication that she was approached by Anthony Babington, a charming and charismatic Catholic who hated the current protestant rule and wanted to see a Catholic monarch on the throne of England. He and six other conspirators informed Mary of a plan to assassinate Queen Elizabeth and free her from her prison. Unfortunately for Babington and his co-conspirators Gifford was a double-agent working for Elizabeth's spymaster Francis Walsingham. Babington was rightly cautious and had not only hidden his messages, but also encrypted them. His cipher had substitutions for all the letters, 35 extra symbols for common phrases, four nulls to confuse any potential attacker and one symbol that meant that the following letter was double. The messages were delivered to Walsingham by Gifford who had them copied and replaced and over the course of the

correspondence between Mary and Babington the cipher was broken by his code-breaker Thomas Phelipes. Even when Walsingham had enough evidence to arrest Babington he waited, he wanted Mary to implicate herself so she too could be executed. When she sent a message to Babington endorsing the plot Walsingham knew he had her, he just had one more thing to do before he sprung his trap. He had Phelipes, also an expert forger, add a post-script to the message, encrypted in the same cipher, asking for the identities of Babington's co-conspirators. Now Walsingham had everything he wanted he arrested everyone involved, Babington and his accomplices were hung, drawn and quartered, and on the 8th of February 1587 Mary Queen of Scots was beheaded.

This is a clear example of how once an attacker has the key to a cipher the system is completely broken, they can both read any messages and, if they also have appropriate access to the channel of communication, can forge messages. This leaves the system completely vulnerable.

2.2.2 Development of Poly-alphabet Ciphers

Frequency analysis was such a successful technique against substitution ciphers that cryptographers had to develop new techniques to counter this. The next most significant group of ciphers that was developed were *poly-alphabet ciphers*. These use more than one set of substitutions to encrypt different letters in the message. The basis for these was work done by Leon Battista Alberti, a Florentine polymath, in the 1460s. He wrote an essay on cryptography after a casual conversation with his friend Leonardo Dato, the pontifical secretary, in the Vatican gardens. Alberti proposed using two substitution ciphers on alternating letters in a message but failed to develop the concept into a fully formed cryptographic system. His idea was developed by Johannes Trithemius, Giovanni Porta and finally Blaise de Vigenère. Vigenère was born in 1523, a French diplomat, his initial interest in cryptography was purely professional and he had read the work by Alberti, Trithemius and Porta while on a two year diplomatic mission in Rome. In 1562 he decided he had earnt enough money to retire and dedicate his life to study. It was then that he developed a powerful new cipher.

The Vigenère cipher consists of 26 alphabets each shifted by an increasing number of letters. The key is a code word or phrase repeated over and over until it has

the same number of letters as the message. Each letter of the message is then substituted using the corresponding letter from the alphabet starting with the current letter in the key. In 1586, ironically the same year as Mary Queen of Scots was plotting with Babington, Vigenère published *Traicte des Chiffres* where he detailed his cipher. Even if Babington had have read Vigenère's work it is possible that he still would not have used it as the new, more secure, system was largely overlooked for another 200 years.

2.2.3 Adoption of Poly-Alphabet Ciphers

The main complaint against poly-alphabet ciphers was that they were more complicated to encode and decode and hence more prone to errors. In order to compromise between the security of polyalphabet ciphers and the simplicity of mono-alphabet ones a series of other techniques were developed to defeat frequency analysis.

Homophonic ciphers are those that use more than one symbol to encode each letter. The number of symbols for a given letter is related to the frequency of that letter in the language that the message is written in, each time the letter is then encoded a random symbol from the set is chosen and used. This would mask the frequencies of the letters. For example, in English, 'e' accounts for approximately 13% of the letters in a given block of text and 'g' for about 2%, so the cipher could have two symbols for 'g' and 13 for 'e'. The weakness in this system is that pairs of letter are not used homogenously, for example 'q' is almost always followed by 'u'. This creates another potential avenue of attack for a cryptanalysts.

In 1626 Antoine and Bonaventure Rossignol, a father and son team, were able to quickly decode a message that was captured by the French army. After it was revealed that their secret message had been read the opposing force surrendered to the French. The Rossigols were appointed senior positions in the court of Louis XIII, they also worked for Louis XIV, who was so impressed he moved their office next to his own apartment. They were so successful the word Rossignol became slang for a lock picking device. Using their knowledge of cryptanalysis they developed the so called Great Cipher. After their deaths the cipher was no longer used and the details were soon lost. There were lots of historical documents, especially Louis XIV's personal documents, which were only written in this encoded form, they remained unread for

hundreds of years. In 1890 some of these documents were passed onto the French army's cryptography department where Etienne Bazeries spent three years trying to decode them. After going down several dead ends he discovered that each symbol in the code represented a syllable, with other little tricks, such as the symbol that meant ignore the previous symbol, added to fool would be attackers. The newly deciphered documents were a historical boon, one of them even identifying the Man in the Iron Mask as General Bulonde, a French general who had disgraced himself through cowardice.

By the 1700s cryptanalysis in Europe had become an industrial process with teams of cryptanalysts working to decode copies of supposedly secret messages that were being sent to embassies. It became clear that extra security was required and the Vigenère cipher was finally adopted for widespread use.

Vigenère's cipher remained unbroken for nearly 300 years until an argument between Charles Babbage and the Bristol dentist John Hall Brock Thwaite, who, claiming he had invented a new cipher when really he had just re-invented Vigenère's one, inspired the eccentric English polymath to turn his mind to code-breaking. Babbage's main breakthrough came when realising that with a finite length key repeated blocks of plaintext could only be encrypted in a finite number of ways, and so could repeat in the ciphertext as well. By looking at two repeated blocks in a message it would be possible to determine the maximum length for the key, and all the possible lengths would be factors of that value. By looking for several repeated blocks of characters one would probably be able to find a unique value for the key-length. After this has been determined the cipher becomes a group of n mono-alphabet ciphers that are now susceptible to frequency analysis. Once the frequencies of the letters in the n different streams have been noted peaks in the frequencies of letters can be matched and the key can be determined. Although Babbage cracked the Vigenère cipher in 1854 he didn't publish his findings and they were only discovered when scholars were examining his notes, credit sometimes goes to Friedrich Kasiski, a Prussian infantry officer and cryptographer, who independently cracked the Vigenère cipher and published his findings in 1863.

2.2.4 World War 1

The invention of the radio created a communication revolution. Previously all communication had to be done over fixed lines which had to be laid before any communication could occur so could not be used to communicate with mobile units such as warships. The advantage of fixed line communication is that it is a lot harder for someone to eavesdrop on communication as they also need physical access to the line whereas radio transmissions can be listened to by anyone in range with a suitably tuned receiver. This meant that encryption became more important than ever for military communications.

Most of the codes developed for use during World War 1 (WW1) were based on ciphers from the previous century that had already been cracked. While they had been improved there was nothing radically different and they posed little challenge for the cryptanalysts at the time. This fact and the massive increase in intercepted communications that radio transmission allowed meant that cryptanalysis paid a very important part in WW1. Probably the most significant example of its use was the deciphering of the Zimmerman Telegram.

Initially America did not join the war in Europe, their president, Woodrow Wilson thought that the conflict could only be resolved through diplomacy and that America could best serve the world by acting as mediator to any talks that may occur. However, this was threatened when a German U-boat sunk the Lusitania, 1198 people were killed including 128 American civilians. Germany agreed to surface their U-boats before attacking so as to reduce the risk of accidentally attacking civilian ships. By 1917 the war was not going well for Germany, they realised that if they reinstituted the policy of unrestricted U-boat warfare then Britain would soon starve and have to surrender, worried that this would bring America into the war they hatched a plan to keep them occupied until their enemies could be defeated. The plan was to convince Mexico to declare war on the USA and an encrypted telegram was sent to the German ambassador in America with instructions to forward details of the plan on to the ambassador in Mexico.

On the first day of WW1 the British ship *Telconia* sailed under cover of darkness to near the German coast and cut Germany's transatlantic cables. In order to carry out their plan they had to send a message to their Mexican embassy via the

American embassy over cables that passed through Britain. Britain intercepted this message and decoded it in Room 40, their cryptographic department, named after the office that it originally occupied. After decoding the message the British cryptanalysts passed it onto Admiral Sir William Hall. Hall realised the significance, he knew that if Germany were to reinitiate the full force of their U-boat campaign it would not be long before Britain would be forced out of the war. As Britain was reluctant to let the Germans know that they could read their secret messages and there was a chance that the U-boat attacks would bring the Americans into the war anyway Hall initially did nothing. On the 3rd of February, Wilson announced that American would remain neutral in spite of the renewed German policy and Britain was forced to act. In order to hide their code-breaking activities Britain sent an agent to the German embassy in Mexico to steal a copy of the forwarded telegraph and handed it to the Americans. On the 6th April America declared war on Germany and cryptanalysis had changed the course of WW1.

2.2.5 Post World War 1 and the Development of the Enigma

In 1918 Arthur Scherbus and Richard Ritter started an engineering company that developed lots of things, from turbines to heated pillows. Probably their most famous invention was the Enigma machine that was used to encrypt German Military communications during World War 2 (WW2). One of the reasons people were reluctant to use the Vigenère cipher was its complexity, this made its use error prone. After it was broken it became clear that even more complicated ciphers had to be developed, these would be even harder for people to use and particularly impractical in the chaos of a battlefield. Scherbus had created a machine that used mechanical and electric signals to automate the encipherment of a message, thus eliminating human error. The Enigma machine consisted of a keyboard, a plug-board, a series of scramblers, a reflector and a series of output lamps. When a button was pressed on the keyboard, corresponding to an input character, an electrical signal travelled through a complex path until it reached the output lamp signifying a different letter, this represented the enciphered character. The path the electrical signal took, and hence the output character was determined by the settings of the scramblers and the plug-board. The plug-board could be used to swap letters, by connecting cables between the various plugs, each representing a different letter, they were swapped. For

example if the letters ‘a’ and ‘p’ were connected on the plug-board then when ‘a’ was pressed on the keyboard it would be the same as pressing ‘p’ on the keyboard when there were no plug-board connections. The scramblers were cylinders with a series of different mappings between the letters. A large part of the security of the enigma was due to the fact that each time a letter was pressed the scramblers would rotate one letter so if the same letter was pressed twice it would lead to a different output letter, and Enigma behaved like a polyalphabet cipher. The scramblers can also be removed so they can go in any order. The reflector sent the electrical signal back through the scramblers to the output. This had no cryptographic significance, but was there to make encryption the same as decryption. If the letter ‘r’ was entered and the electrical signal weaved its way through the plug-board cables and the scramblers via the reflector to the letter ‘h’ then someone trying to decode the message could press ‘h’ on their machine with the same settings and the electrical signal would do the reverse path and light the lamp for ‘r’. The entire machine was 34 * 28 * 15 cm, but weighed a hefty 12 kg.

Scherbus initially had trouble finding anyone to buy Enigma machines, they costs the equivalent of £20,000 in today’s money, and most businesses said that they could not afford it. The German military was unaware of the damage enemy cryptanalysts had done to their war effort and so were initially not interested. Fortunately for Scherbus, in 1923 Winston Churchill published *The World Crisis* detailing an early German cryptographic failure, and later that year the Royal Navy published their official history of the WW1. The German military realised what their weak ciphers had cost them and started ordering Enigma machines. In 1925 they went into mass production.

2.2.6 Cracking Enigma

Until 1925 the rest of Europe were still receiving a large amount of intelligence from Germany via decrypted transmissions. After the German adoption of the Enigma machine this rapidly stopped. Previously British and French cryptanalysis had been tenacious in their efforts to decipher previously unbreakable codes, but when faced with Enigma they quickly gave up. After WW1 Germany’s military was had been largely neutralised and the country was in ruins. The French no longer feared her might. On the other side of Germany, her neighbours weren’t as complacent. In 1925

Poland was caught between a strengthened Germany and Russia, a nation bent on spreading communism. Faced with these threats Poland was desperate for intelligence and had a very strong cryptanalysis department called Biuro Szyfrów. There was little that they could do without first understanding the workings of the cipher.

On the 8th of November they got their first break. A German working in the department responsible for secure communication, Hans-Thilo Schmidt, sold the plans to the Enigma machine to a French agent for 20,000 Marks. The French were not particularly interested in their new found knowledge, they assumed that even if they understood how the Enigma worked they would still not be able to work out a way to break the cipher. They did however have a decade old treaty of military cooperation with Poland, who had expressed interest in anything to do with Enigma. Thinking it of little practical value the French gave the information to Poland. Using this information the Biuro was able to create a replica of the Enigma machine to study. As well as details of the Enigma machine the French intelligence contained the protocol that the Germans were using. Codebooks were distributed amongst the German radio operators. The books contained a month's worth of plug-board settings and scrambler arrangement and orientations, one for each day, called day-keys. To make the system more secure messages were encrypted with different scrambler orientation settings. This was called the message-key and was encrypted twice with the day-key and transmitted at the start of the message. This was done to ensure that the message-key was received correctly and the message could be decoded without error, but it introduced an insecurity into the system as the attacker knew that the 1st plaintext character of the message was the same as the 4th although the ciphertext characters would be different. The difference between them would be determined by the scrambler settings.

This was studied by the Polish cryptanalyst Marian Rejewski. He had at his disposal hundreds of messages every day, the first six characters of each of which would be encrypted using the same settings. Although he did not know the plaintext characters he studied the way they changed, finding they formed chains with varying numbers of links. For example, if the first character of one message was 'L' and the fourth was 'W', in another message the first would be 'W' and the forth 'G' and then in a third 'G' would change back to 'L', forming a chain with three links. Rejewski realised that properties of these chains would be affected only by the scramblers and

not by the plug-board settings, they would only change the values individual letters in the chain. This meant that the chains could act as a fingerprint for the different scrambler settings. He spent a year cataloguing the chain lengths for all of the 105,456 possible scrambler arrangements, from this he would be able to determine the scrambler settings of the day-key, he could then use this to try and decrypt a message key and use that to decrypt a message, if the plug-board settings did not affect any of the letters in the message-key it would be possible to mostly decrypt the message. The plug-board setting could then be determined by looking at the generated message and changing letters until it made sense. Using this technique Rejewski could retrieve a day-key and read all of that day's messages.

When the Germans adapted the way they transmitted messages it made Rejewski's catalogue of chains obsolete. Instead of painstakingly recreating it he developed a mechanical device, based on an enigma machine, which was capable of trying lots of different scrambler settings until it spotted the correct one. As the scramblers could be arranged in six different ways six of the so called *bombes* were required. In December 1938 the Germans augmented the security of Enigma, increasing the number of different scramblers to five and the number of plug-board cables to ten. This vastly increased the number of possible plug-board permutations and increased the number of bombes required by a factor of ten. The cost of manufacturing the new bombes was beyond the resources of the Polish cryptographic department, and in 1939 the flow of German intelligence into Poland dried up. Sensing an imminent German invasion the Polish were willing to share their cryptanalysis breakthroughs with their allies. On the 24th of July senior cryptanalysts from France and Britain arrived in Poland where they were informed, to their surprise, of the Polish successes in reading secure German messages. Spare Enigma machines and blueprints for the bombes were shipped to London and Paris where the Polish work could continue. On the 1st of September Hitler invaded Poland and WW2 had begun.

In Britain the responsibility for breaking German codes had moved from Room 40 to Bletchley Park, a large Victorian mansion in Buckinghamshire. The British cryptanalysts quickly mastered the polish techniques and with greater resources had created the bombes necessary to break the encryption. The Polish technique hinged on the fact that the Germans always transmitted the message-key twice at the start of the

message, this repetition was the weakness that allowed the cryptanalysts to peer inside the Enigma code. Some cryptanalysts at Bletchley Park were responsible for continuing the research into weaknesses in the code, in case the Germans strengthened their transmission protocol and stopped sending the key twice. One of the researchers was Alan Turing, he realised that there was another potential avenue for attack due to the fact that Enigma was being used by the military. The military thrive on routine, the contents of some parts of the messages would be predictable, for example, a weather report would be transmitted shortly after 6 am every day. The section of plaintext that was known to the attackers was referred to as a *crib*. Using these cribs Turing developed a new technique for decoding Enigma messages. He also studied chains in encipherment of various letters, for example if a 'w' was ciphered as an 'e' and the next plaintext letter was 'e' that had been changed to a 't' and later on in the message there was a plaintext 't' that was converted to a 'w', this was the type of chain Turing was interested in. Turing imagined a machine that was a series of Enigma machines in parallel. Details of a chain would be entered into it by connecting the output of the first one to the input of the second and so on. In between the input of the first and the output of the third there was a lamp. The scrambler settings on the three machines would rotate. The lamp would only light when the circuit was complete, this would only happen when the scrambler settings were correct for all three machines. Again the study of these chains allowed the cryptanalysts to divorce the plug-board settings from the scramblers. This is because the plug-board settings are constant, although in the example chain above it is not known what letter the first 'w' is converted to by the plug-board, it is known that when the 't' is converted to a 'w' the signal has travelled through the plug-board cable twice, cancelling out the effect. While the plug-board contributes the majority of the different combinations of settings an Enigma machine can have, it is only a mono-alphabetic substitution, and, as there were only ten plug-board cables, an incomplete one at that. By decoding the original message with the scrambler settings the plug-board settings can soon be determined. Turing's decoding machine was built; it arrived on the 14th of March 1940 but was a lot slower than anticipated. The design was refined and a new one was ordered, but it was going to take four months to build. On the 10th of March 1940 the Germans changed their key transmission protocol so that the key was only sent once and the decoded Enigma messages dried up until the 8th of August when the new

machine arrived. This fulfilled all of Turing's hopes and messages could be decoded for the rest of the war.

The information that was gained by cracking the Enigma code was of crucial value to the wartime effort of the allies. Lessons can be learnt for both cryptanalysts and the users of cryptographic algorithms. The Polish code-breakers, motivated by desperation, never gave up hope that Enigma could be broken, and through their tenacity found a technique that could retrieve German Keys. The real weakness in the enigma code wasn't the code itself but the way it was used. The first method of breaking it used the fact that the message-key was enciphered twice and the second method relied on knowing sections of the plaintext. While the cipher isn't secure by today's standards, as it is important to assume that an attacker may have access to plaintexts as well as ciphertexts, any code can be made much weaker by using it improperly. It is important to minimise any additional information that is leaked to any potential attacker.

2.2.7 Modern Cryptography

In 1949 *Communication Theory of Secrecy Systems* was published in the Bell Labs Technical Journal by Claude Shannon [14]. It established the mathematical basis for modern cryptography and developed two metrics for measuring the security of a cipher, *confusion* and *diffusion*. Confusion is related to the relationship between the key and the ciphertext, this will be very complicated in a cipher with good levels of confusion. Substitution, generally performed by so called s-boxes, is a key component in ensuring confusion. Diffusion is the effect the plaintext has on the ciphertext, it is related to the *avalanche effect*, a term first used by Horst Feistel [15], which describes how changing one bit at the input causes an avalanche of changes through to the output. It was developed into the *Strict Avalanche Criterion*, which states that, for optimal diffusion, changing one bit in the plaintext should change on average half of the bits of the ciphertext. Diffusion is mostly associated with transposition operations in ciphers. Now that the mathematical theory behind encryption had been formalised ciphers could be designed in a rigorous way as opposed to the *ad hoc* methods that had previously been used.

2.2.7.1 *Block Ciphers*

As computing developed the price of computers steadily reduced. By the 1970s companies were able to afford computers and they became an important part of business. Businesses would often have to send and receive secure messages, these would have to be encrypted and so businesses would develop their own encryption schemes. This posed no problems if the secure data had to be sent between different offices of the same company, but it would be problematic if data had to be sent between different companies as the algorithms would not be compatible. To address this problem the US government decided to create a standard encryption algorithm that could be used by everyone. The new *Data Encryption Standard* (DES) was based on the algorithm Lucifer which was developed by Horst Feistel [15], an engineer working for IBM. The NSA examined the algorithm to ensure that it was secure and made a couple of changes. They reduced the key length from 64 bits to 56. This meddling lead to speculation that the NSA had deliberately weakened the cipher in order for them, and only them, to be able to decrypt it.

As DES was so widely used the security of the algorithm was heavily researched. The first attack to be proposed was Differential Cryptanalysis, which uses several different, but related, plaintexts to gain information about the key. When it was published in 1990 it was discovered that the chosen s-boxes had strengthened the cipher against this attack. Differential cryptanalysis was known to IBM researchers in 1974 but as it is a powerful attack that can be applied to lots of different ciphers the NSA asked them to keep it secret. Although the cipher was resilient against this form of attack, an attacker with access to 2^{47} chosen plaintexts can still break DES. In 1992 Linear Cryptanalysis was developed [16], this involved generating linear approximations to sections of the cipher with either a high or a low probability of correctness. DES can be broken with 2^{43} known plaintexts [17].

While in the strictest of senses this means that the security of DES has been compromised, the attacks are still not feasible, requiring unrealistically large amounts of known or chosen plaintexts. The death knell for DES came in 1997 when RSA Security ran a competition to crack DES as a demonstration that with modern computers the 56-bit key no longer provided adequate security. The competition was won by a distributed computing project called the DESCHALL Project who managed

to retrieve the key in 96 days. This has since been improved and DES can now be cracked in an average of 7.2 days [18].

It was clear that a new stronger encryption standard was required and in 1997 the National Institute of Standards and Technology (NIST) started the search for a new algorithm that would be dubbed the Advanced Encryption Standard (AES). In 2001 the new algorithm was chosen, it was Rijndael, developed by Vincent Rijmen and Joan Daemen. There are currently no successful mathematical attacks on AES, although it is vulnerable to side channel attacks.

2.2.7.2 *Public Key*

In order for secure communication to work both parties need to have access to the key. Transferring this in a secure way can be problematic. In the 1970s couriers were used to transfer keys to recipient of the secure data so it could be deciphered. While this is more secure than having a courier carry sensitive documents as a potential attacker needs to get both the key from the courier and intercept the encrypted transmission it is still less than ideal. In 1976 Whitfiel Diffe, Martin Hellman and Ralph Merkle developed *Diffie-Hellman key exchange*. This technique is best explained by an analogy. Alice and Bob want to get married and Alice needs to send her wedding ring to Bob. She can only send it through the post, but she does not trust her postman not to steal it. She locks the ring in a box and sends the box to Bob, he unfortunately does not have the key and Alice can't post it to him for fear of the postman getting it. Bob puts his own padlock on the box and sends it back to Alice who then unlocks her padlock and sends it back to Bob. The ring is now only protected by Bob's padlock to which he has the key. In this analogy the ring is the key to a block cipher and the padlocks are special encryption algorithms. The biggest problem in developing this scheme was finding a way to encrypt the data where encryption and decryption was commutative, that is to say that it doesn't matter that the second encryption is performed before the first decryption. This was overcome by the use of discrete logarithms.

This form of key exchange does have some problems; it requires three transfers of data between the two parties which is not always convenient, if communicating from vastly different time zones, for example. In 1975, while developing the idea for key exchange, Diffie also published his idea for *asymmetric key algorithms*. These are

group of algorithms where encryption and decryption are performed using different keys, the encryption key can be freely distributed so it is called the *public key*, it can be used by anyone to send secure data to the holder of the decryption or *private key*. In this way there needs to be no secure transfer of keys. Although Diffie described the concept of public key encryption he could not find any mathematical functions with suitable properties. It was left to Ron Rivest, Adi Shamir and Leonard Adleman to develop a working system which was patented in 1977. Rivest, Shamir and Adleman created the company RSA Security to commercialise the research.

Cryptography is a world of secrets, just like the invention of the computer at Bletchley park, British cryptanalysts were busy researching public key encryption and secure key exchange. In the late 60s Peter Ellis was working at GCHQ on the problem of key exchange, like Diffie, he came up with the idea of separate keys for encryption and decryption, but was unable to think up any suitable function. Then in 1973 a new mathematician joined, Clifford Cocks. He had been previously working in number theory and recognised the potential of prime numbers and factorisation to solve the problem. Unfortunately in the early 70s computers were still quite primitive and the amount of processing power required to implement the system was a stumbling block. In 1974 Cocks was explaining his idea to his old school and university friend Malcolm Williamson, who had also started working at GCHQ. He was suspicious of the idea and studied it in detail intent on finding a flaw. Instead, in 1975, he discovered the Diffie-Helman key exchange. GCHQ scientists had discovered all of the principles of public key encryption before anyone else, but as they were sworn to secrecy this was not revealed until 1997 when Cocks was allowed to give a brief history of GCHQ's contribution to public key encryption while presenting some unclassified research he had performed on RSA at a conference.

2.2.7.3 *The Digital Revolution*

In the 1970s Phil Zimmerman was deeply concerned about the threat of nuclear war and became an anti-nuclear political activist. In the 80s tensions between the US and the USSR calmed and Zimmerman's focus changed to another political cause, the public's right for privacy. Hundreds of millions of emails are sent every day and if they are unencrypted then they are particularly vulnerable to eavesdropping. At the time there was no software available for members of the public to use to encrypt their email. Zimmerman spent years developing Pretty Good Privacy (PGP) a system

designed to do just that. At the heart of the software was the public key algorithm RSA which is a lot more computationally intensive than a symmetric key system with an equivalent level of security. To get round the problem of the software being prohibitively slow on the personal computers of the average user PGP only uses RSA to encrypt the key for a much faster symmetric key system which encrypts the message. By 1991 Zimmerman had a fairly polished product, but there were problems. He was worried that Congress were going to try an ban products like PGP in order to ensure that law enforcers could read criminals email, so in June he asked a friend to upload it to a Usenet bulletin board.

Zimmerman had released his software and it was being used exactly for the purposes that he had intended, human rights groups all over the world were using PGP to protect their communications. Unfortunately his problems were not over yet. PGP used the RSA algorithm which was protected by a patent for which he did not have a licence. More seriously, Zimmerman's work had attracted the attention of the FBI. The US had export controls on cryptographic systems and systems with more than 40 bits were considered munitions, in 1993 Zimmerman became the target of an investigation into exporting munitions without a licence. Zimmerman found an interesting loop-hole to the export controls, he published the source code in a book and as the export of books is protected by the First Amendment all someone had to do was scan the book with OCR software and compile the program. In 1996 after three years of investigation the case was dropped. Zimmerman also managed to reach an agreement with RSA Inc. PGP was finally a legitimate program.

In the 90s the whole cryptographic climate was changing, the internet was starting to develop allowing for the development of ecommerce. In order for this to be successful customers had to have faith in security of the new medium. In 1995 Hal Finney set a challenge to break the 40-bit RC4 encryption that came with the international version of the Netscape browser. This was completed in less than two weeks, a worrying achievement for anyone buying anything over the internet. The government control on encryption had another economically damaging side-effect, the weakness of the *Content Scrambling System* (CSS). CSS is the encryption system used to protect the content of DVDs, it was developed in 1996 and in order for DVD players to be freely exportable was restricted to 40 bits. This allowed the copy protection to be easily cracked, and the free sharing of thousands of movies over the

internet, allegedly costing movie studios millions of dollars in lost profits. Due to the changing climate and the sense that strong encryption was needed domestically the export restrictions were dropped in 1996, paving the way for people all over the world to adopt secure ciphers to protect their secrets.

2.2.7.4 *Side Channel Attacks*

It has long been known that the various emissions that real devices make during their operation can reveal secret information. The purpose NSA's TEMPEST program, started in the 1960s, was to ensure that electronic emissions that escaped from a device would not reveal sensitive information about its operation. It wasn't until the mid 90s that the information gathered through side channels was used to break encryption systems.

In 1996 Paul Kocher developed a radical new type of attack. All previous cryptanalysis had relied on weaknesses in the cipher, exposing patterns in the ciphertext that could be exploited. The new method used information gained through a side channels to determine the internal state of the device performing the encryption. The first side channel attack was timing analysis, it exploits the fact that operations take a certain amount of time to perform and so information can be gained by timing how long it takes a device to respond to a query [19]. Then in 1999 he extended the idea to power consumption [2]. In modern transistor technology more power is consumed when a value changes from '0' to '1' or '1' to '0' than if it stays the same, by measuring the power consumption information can be gathered about the state of registers inside a crypto-device. Some information can be gained by examining the power trace, the power consumption data for one encryption, and looking for significant features, this is called Simple Power Analysis. A much more powerful technique is Differential Power Analysis, this combines information from several encryptions with the same key and with enough power traces can retrieve the complete key, even from a state of the art algorithm such as AES [3].

Since the demonstration of the general technique a number of side channels have been proposed, from emitted EM radiation [20] to the acoustic noise a processor emits [21]. Although several countermeasures have been proposed [9, 22-25] none work with complete efficacy and how to protect algorithms against side channel attacks is still an open problem.

2.2.8 Conclusion

Over the last 30 years cryptography has become increasingly pervasive in modern life. Ciphers no longer just protect state secrets and military plans but trade secrets, finances and privacy. Cryptographic hardware has been transformed from large cumbersome devices to small sections of existing chips or programs that are constantly carried. Since their development block ciphers have provided an efficient and conceptually simple way of generating the complex transforms that implement the principles of secure communication outlined in Shannon's seminal 1949 paper. The advantages of block ciphers are such that they have become by far the most common type of cipher in use today.

While the increase in public use of cryptography is beneficial, as the general public are able to protect their secrets and ensure their privacy with secure encryption. There is another side to the coin, cryptanalysis is becoming increasingly more important to criminals. Additionally, the increase in mobile cryptography has made ciphers increasingly vulnerable to attacks like Differential Power Analysis, which requires physical access to the device and does not rely on a mathematical weakness in the algorithm. The ability to successfully protect secrets has always been, and will continue to be of very high importance to society.

Chapter 3 Block Ciphers

3.1 Introduction

Block ciphers have been an important area of cryptography since their development in the 1970s. They work on a fixed size block of data and use the secret key to transform the unencrypted data, or *plaintext*, into its encrypted form, or *ciphertext*. In the 1970s a standard block cipher was developed by the US called *Data Encryption Standard* (DES). This reigned supreme for over two decades until increases in computing power rendered its security questionable. A modification to increase the effort an attacker must use to crack it by chaining three DES blocks together was developed and given the apt name *Triple DES* (TDES). Finally, in 2001, after the submission and lengthy evaluation of algorithms from the public, they were both superseded by the *Advanced Encryption Standard* (AES), which remains the accepted algorithm today.

In section 3.2 the basic concepts in modern cryptography are introduced. Section 3.3 examines the basics of block ciphers in more detail. Sections 3.4 - 3.6 describe the three most important block ciphers of the last 40 years, DES, TDES and AES.

3.2 Basic Concepts in Cryptography

3.2.1 Cryptographic Methods

Cryptographic algorithms seek to make data unreadable by everyone except a trusted subset of the population. To achieve this, the data is put through a transformation, the output of which is determined not only by the original data, but

also by a *secret key*. The data can then only be decoded by someone who has the relevant key to unlock it.

Figure 3-1 shows an example of a complete cryptographic system. Alice wants to send a message to Bob without Eve (or anybody else) being able to read it. She takes her unencrypted message, known as a *plaintext*, and encrypts it with a secret key; the enciphered message is called a *ciphertext*. She can then send this message to Bob, who can read it as he also has the secret key. Although Eve is able to intercept the message she cannot make sense of it as she does not have the key.

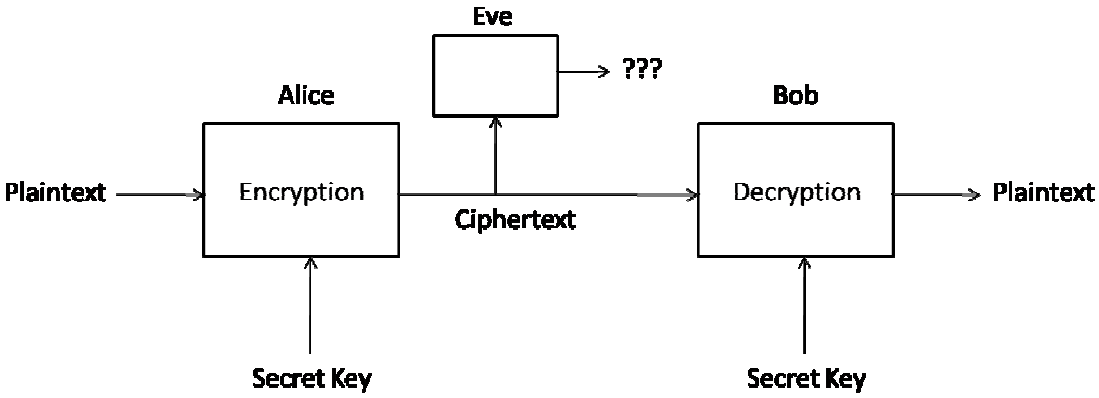


Figure 3-1: An example of an encrypted communication.

There are two main types of cryptographic algorithm, symmetric and asymmetric key. Symmetric key algorithms have symmetry in the sense that the same key is used for both encryption and decryption. In order to use a symmetric algorithm all parties involved in the communication must have the key.

Asymmetric key algorithms, also known as public key ciphers, use a different key for encryption and decryption. The key used to encrypt the data is called the *public key* and can be freely distributed. Only the *private key*, which is kept secret, can decrypt the message.

Additionally there are also cryptographic hash functions. These are one-way functions that return a fixed length output and do not have a key. Once the data has been put through them it cannot be retrieved. They are used for a variety of purposes such as password verification and checking the integrity of a message.

3.2.1.1 *Symmetric Key Algorithms*

There are two main types of symmetric algorithms, block ciphers and stream ciphers and they are discussed in the following two subsections.

3.2.1.1.1 Block Ciphers

Block ciphers are symmetric algorithms that act on fixed length groups of bits called blocks, modern algorithms typically have block sizes of around 128 bits. Block ciphers are key dependent, bijective transforms, meaning each plaintext maps uniquely to exactly one ciphertext and the particular mapping is determined by the key (and the cipher being used). The specific transform is controlled by the secret key. Decryption is similar, application of the secret key and the inverse cipher reveals the original data. Block ciphers are discussed in more detail in section 3.3.

3.2.1.1.2 Stream Ciphers

Stream ciphers work on smaller blocks, typically bits or bytes, and the transform for successive blocks does not remain fixed throughout the entire encryption. The difference between block and stream ciphers is not always that distinct. Block ciphers can be modified to act as stream ciphers, although bespoke stream ciphers are generally faster and less complex than block ciphers.

3.2.1.2 *Asymmetric Key Algorithms*

Asymmetric public key cryptography uses two different keys. The encryption, or public, key may be freely given to anyone, while the decryption, or private, key is kept secret. The keys are related mathematically, but it is not feasible to determine the private key from the public key.

The two main applications of public key cryptography are:

- **Public Key Encryption:** used to ensure the confidentiality of communication to the owner of the keys. Messages are encrypted with the public key, they then cannot be decrypted by anyone unless they possess the corresponding private key.
- **Digital Signatures:** used to verify the identity of the sender and the authenticity of the message. Messages are signed with the sender's private key, they can be verified by anyone who has access to the sender's public key

Probably the most well known system is PGP which was developed by Philip Zimmerman in 1991and released free of charge on Usenet. It allows both the encryption and signing of messages.

3.2.1.3 *Cryptographic Hash Functions*

A *cryptographic hash function*, h, takes an arbitrarily sized message, m, and creates a fixed length *message digest*, $h(m)$, that appears random. Cryptographic hash functions have other required properties:

- Given $h(m)$ it must be difficult to find m.
- Given $h(m)$ it must be difficult to find m_2 such that $h(m_2) = h(m)$.
- It must be difficult to find m_1 and m_2 such that $h(m_2) = h(m_1)$.

Hash functions are one of the most versatile cryptographic primitive and have several different applications:

- **Commitment scheme:** By concatenating a message with a random nonce, a value used to ensure uniqueness of output, and taking its hash a user can commit to a message while still keeping it hidden. By later revealing the nonce to another user it can be shown that the original user did commit to the message.
- **Message integrity:** Comparing the hash of a received message to a hash that was sent verifies that the message was received correctly.
- **Digital signature:** In order to increase performance most digital signature algorithms only sign the message digest rather than the entire message.
- **Password verification:** When a password is entered the hash of the entry is taken and compared to the hash of the actual password. Thus the actual password does not need to be stored as that would be insecure.

3.2.2 Cryptanalysis Methods

If data is worth protecting then it will be of some value, therefore code-breaking is as old as codes themselves. This section briefly introduces some techniques to break cryptographic algorithms. Different cryptanalysis methods assume that the attacker has different levels of knowledge or access to the cipher:

- **Ciphertext-only:** the attacker has a list of ciphertexts.
- **Known-plaintext:** a set of ciphertexts linked to corresponding plaintexts.

- **Chosen-plaintext/-ciphertext:** a set of ciphertexts (plaintexts) linked to corresponding plaintexts (ciphertexts) chosen by the attacker.
- **Related-key attack:** like a chosen-plaintext, except the attacker can obtain ciphertexts encrypted with different keys. The relationship between the keys is known e.g. one bit difference, although the actual values are not.

Classical cryptanalysis mostly involved looking at the frequencies of various letters in ciphertext and comparing those to the frequencies of letters in the plaintext language. As cryptographic algorithms got more advanced this was no longer possible, so new techniques had to be created. There are several general attacks on block ciphers, such as differential cryptanalysis, a chosen plaintext attack that uses differentials, pairs of plaintexts related by a constant difference, to detect patterns in statistical distribution [26], and linear cryptanalysis, which involves generating linear approximations to sections of the cipher that have either a high or low probability of being correct [16]. It is also possible to exploit specific weaknesses in algorithms by designing bespoke attacks, for example the Davies attack [27], which makes use of the fact that adjacent s-boxes, non-linear functions that take a number of bits as input whose output is determined by the value of the input, in the *Data Encryption Standard* (DES) share some input bits.

The simplest form of attack is to simply go through all possible keys until the correct one is discovered. This is called a brute force attack. With current levels of computing power it is possible to attack outdated encryption algorithms like the block cipher DES [28, 29], which has a key length of 56 bits, giving a total of $7.2 * 10^{16}$ possible key values. Modern block ciphers, like AES generally use between 128 and 256 bit keys, giving between $3.4 * 10^{38}$ and $1.2 * 10^{77}$ possible key values. This means it is not currently feasible to perform brute force attacks on modern block ciphers.

Another type of attack relies not on information derived from the algorithm, but instead information that is leaked from the physical implementation. Real world systems are not simple black boxes where the input goes in one end and the output comes out the other, but rather are complicated devices that consume power and emit electromagnetic radiation and take varying amounts of time to perform different

calculations. All of these things can leak information about what is going on inside the device and attacks that exploit this are known as *side channel attacks*.

The majority of the work presented here is related to side channel attacks, specifically *Differential Power Analysis* (DPA). Power analysis uses the fact that the power consumption has a high dependence on the data that is being processed. DPA combines the power consumption data from several encryptions using different plaintexts and uses statistical techniques to determine the most probable value for the key. Power analysis in general and DPA in particular are discussed in more detail in sections 4.3 and 4.3.2 respectively.

3.3 Block Ciphers

Block ciphers are a bijective transform that take the plaintext as an input and convert it into the ciphertext. In order to be secure this transform need to have certain mathematical properties. Section 3.3.1 introduces the most important properties required for security: confusion and diffusion. While the mathematical properties are clearly very important, knowing them and implementing a cipher that satisfies them are very different things. Section 3.3.2 describes the structures that make up the majority of modern block ciphers. Section 3.3.3 gives details on the various ways that a block cipher can be used to encrypt a set of data that is larger than a single block.

3.3.1 Confusion and diffusion

In order for a block cipher to be secure against statistical attacks it must effectively deal with the redundancy in the plaintext data. In 1949 Claude Shannon published a seminal paper that is the mathematical basis for modern cryptography [14]. In this he defined two concepts, *diffusion* and *confusion*.

Diffusion means that redundancy in the plaintext and key are dissipated in the ciphertext; the influence of the value of a single input bit will be diffused over several ciphertext bits and hence it will be difficult for an attacker to gain knowledge about the plaintext from the ciphertext. Diffusion is characterised by the *Avalanche Effect* and the *Strict Avalanche Criterion* (SAC), terms first used by Horst Feistel. The Avalanche Effect results in a significant change in the output bits, ideally one half of the bits change when a single input bit is complemented. The SAC is an extension of

this and it is satisfied if a change in each of the input bits changes each of the output bits with a probability of 0.5. This means that the ciphertext will appear to change randomly between related messages, hiding message relationships which could be used by an attacker. Operations that transpose bits increase the level of diffusion.

Confusion refers to making the relationship between the inputs and the ciphertext as complex as possible, this is to ensure that it is difficult for an attacker to discern information about the inputs from the ciphertexts. Ideally it would be impossible for an attacker to distinguish a series of ciphertexts from a random bit-stream. Confusion is ensured by using s-boxes, these are look up tables that implement highly non-linear transformations.

3.3.1.1 *Quantifying Confusion and Diffusion*

Diffusion is the distribution of the effect of the value of the plaintext in the ciphertext, in the ideal situation a change in any bit of the plaintext would affect each bit of the ciphertext with a probability of 0.5. This can be determined by calculating the ciphertexts for a large number of random plaintexts and counting the number of ciphertext bits that are affected by changing one bit of the plaintext. The distribution of the number of affected bits can then be compared to the theoretical distribution of the ideal case using the Kolmogorov-Smirnov (KS) test [30]. The KS test is a statistical test used specifically to test the equivalence of two probability distributions using a finite number of samples. This is then repeated for each bit of the plaintext and then the entire process is repeated for the key.

Determining whether or not sufficient levels of confusion have been reached in a cryptographic algorithm can be achieved by testing for statistical randomness. A simple frequency test, checking that there is an approximately equal number of 1s and 0s, can give an indication whether or not a cipher provides adequate confusion [30].

3.3.2 Block Cipher Structure

Although block ciphers represent a very complicated transformation most are composed of repeating iterations of simpler functions. By combining simple operations that mix in key data or increase either confusion or diffusion and having several iterations of sequences of these blocks, commonly referred to as *rounds*, a secure cipher can be built up out of small, easily implementable blocks. When taken

as a whole, the combination of simple operations that forms a round is called the *round function*. Two popular schemes for designing block ciphers are *Substitution-Permutation Networks* (SPN) and *Feistel ciphers*.

As the name implies, Substitution-Permutation Networks based ciphers are mainly made up of operations that either substitute values, or permute bits. During substitution the data is separated into smaller blocks and the values in these blocks are substituted for others, typically using a non-linear s-box, this increases the confusion. Permutation works across several blocks and mixes the data, swapping bits or combining values so the influence of data from one part of the plaintext is diffused through the whole ciphertext. An example of an algorithm based on SPN is AES. Using an SPN approach it is easy to design ciphers with sufficient levels of confusion and diffusion to be secure using a fairly simple set of cryptographic primitives.

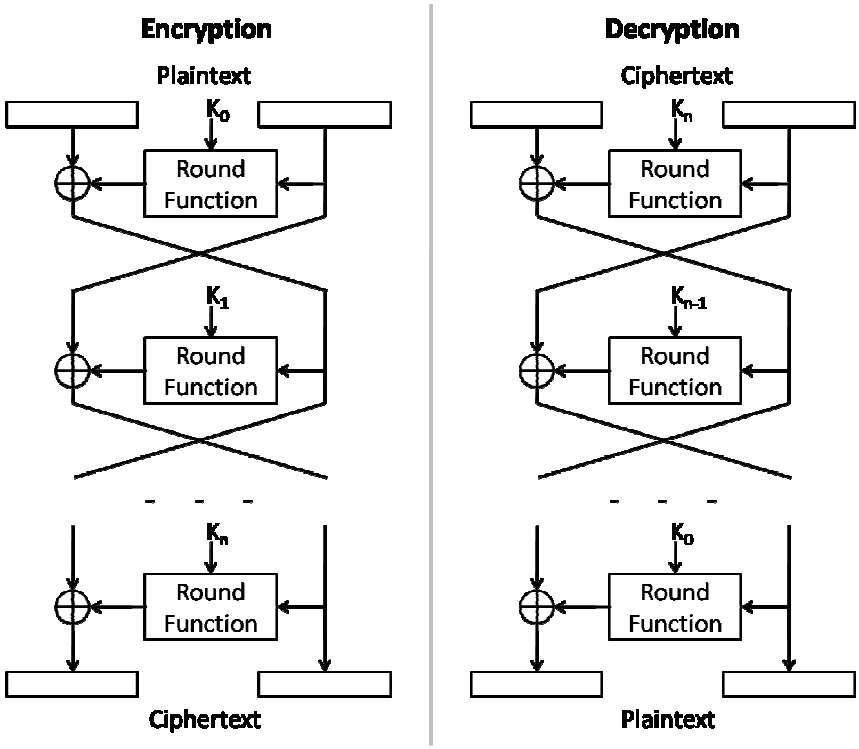


Figure 3-2: The structure of a Feistel cipher.

Feistel networks were first used in the cipher Lucifer, developed at IBM by Don Coppersmith and the eponymous Horst Feistel. They are a subset of SPN so are also made up of a series of simple functions repeated in rounds. The plaintext is split into two equal halves. The round function is applied to the right hand half which is then XORed with the left hand side and it becomes the new right hand side, the original

right hand side becomes the left. This is shown in Figure 3-2. An advantage of Feistel networks is that encryption and decryption is very similar, often requiring little more than a reversal of the key schedule. An example of an algorithm based on a Feistel structure is DES.

3.3.3 Cryptographic Modes of Operation

Block ciphers only work on fixed length blocks of data, but the actual data that needs to be encrypted can be of any arbitrary length. Several different modes of operation for block ciphers have been devised. The most common ones are described in this section, they are: Electronic Code Book, Cipher Block Chaining, Cipher Feedback, Output Feedback and Counter.

3.3.3.1 *Electronic Code Book*

The simplest mode is called *Electronic Code Book* (ECB), the input data is separated into blocks and each is encrypted individually. A block diagram is shown in Figure 3-3. Plaintexts with the same value will always give the same ciphertext, this means that patterns in the data can still be seen in the encrypted data. Also this method is susceptible to the replay attack, a network attack where an attacker repeats valid data that was gained from eavesdropping on a previous session.

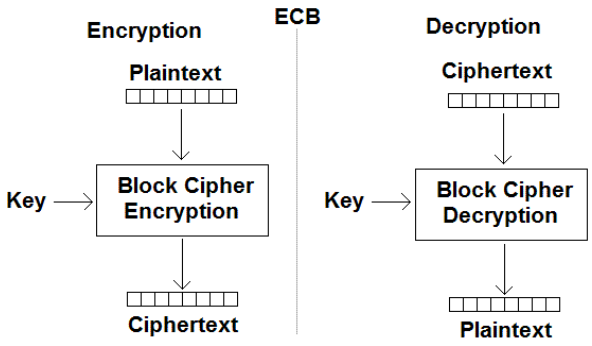


Figure 3-3: Block diagram of the ECB cryptographic mode of operation.

3.3.3.2 *Cipher Block Chaining*

In *Cipher Block Chaining* (CBC) the plaintext is XORed with the previous ciphertext before encrypting it; the first plaintext is XORed with an initialisation vector, see section 3.3.3.6. A block diagram is shown in Figure 3-4. Each ciphertext is now dependent on all previous plaintexts so 1 bit error in the plaintext corrupts all following ciphertexts, one bit error in the ciphertext corrupts the corresponding

plaintext block and flips the corresponding bit in the next block. Encryption must be done sequentially as the output from each block is needed at the input to the next, but as the converse is true, i.e. the only data required from the previous block is the input and it is only needed to convert the output of the decryption to the actual plaintext, decryption can be parallelised.

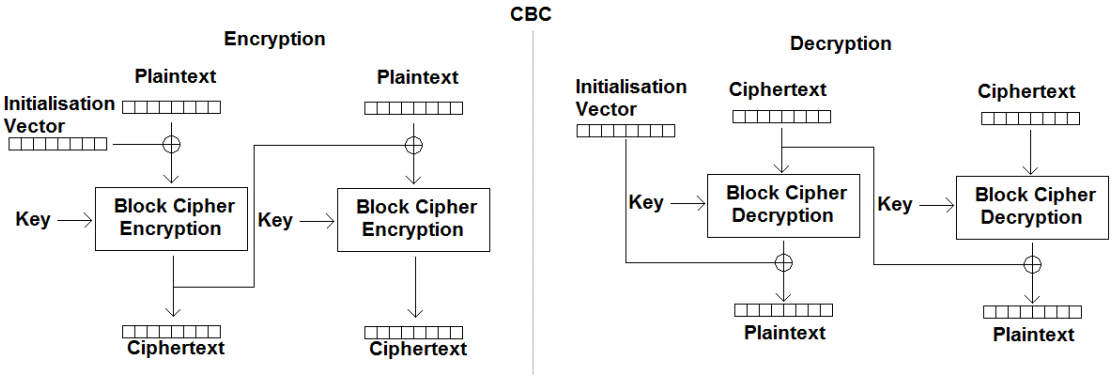


Figure 3-4: Block diagram of the CBC cryptographic mode of operation.

3.3.3.3 *Cipher Feedback*

In *Cipher Feedback* (CFB) an initialisation vector is encrypted, the plaintext is then XORed with the output from the encryption to form the ciphertext, this ciphertext is then encrypted and XORed with the next plaintext and so on, a block diagram is shown in Figure 3-5. 1 bit error in the plaintext corrupts the entire cipher stream; 1 bit error in a ciphertext flips the corresponding bit in the corresponding plaintext and the entire next block. Encryption must be done sequentially, but decryption can be parallelised. It is important to realise that as the plaintext interacts with the output of the block cipher in both the encryption and the decryption forms the block cipher is used in encryption mode.

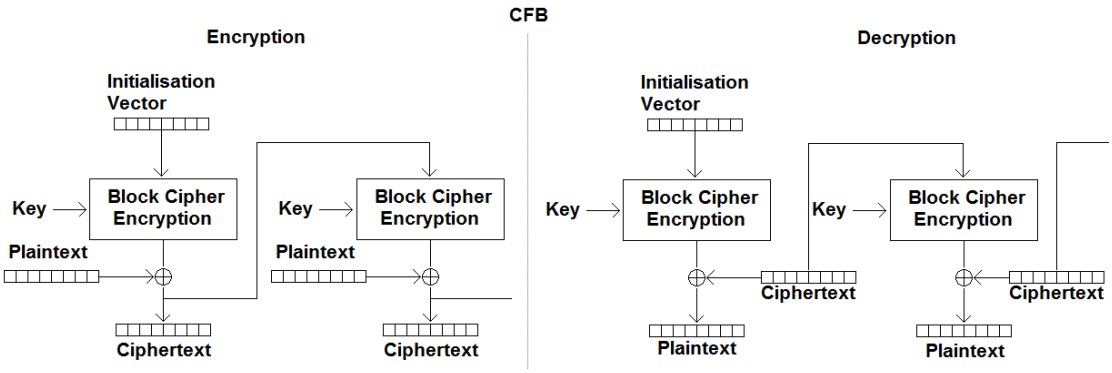


Figure 3-5: Block diagram of the CFB cryptographic mode of operation.

3.3.3.4 *Output Feedback*

Output Feedback (OFB) is similar to CFB, an initialisation vector is encrypted and is XORed with the plaintext data to form the ciphertext, the difference is that the output of the encryption is fed back before the plaintext is added. A block diagram is shown in Figure 3-6. Neither encryption nor decryption using OFB can be parallelised, but unlike CFB and CBC modes errors do not propagate and will only affect the bits in question. As in CFB, both encryption and decryption use block ciphers in their encryption mode, in fact the encryption and decryption modes are exactly the same, simplifying any implementation. It is very important to not use the same initialisation vector with the same key; this will result in an identical random bit-stream and will leak a lot of information about the plaintexts. Another possible insecurity with OFB is that if the output of the block cipher happens to give the same value as the initialisation vector then the random bit-stream will repeat. The probability of this happening is related to the number of plaintexts that are encrypted with the same key and so this problem can be mitigated by changing the key regularly.

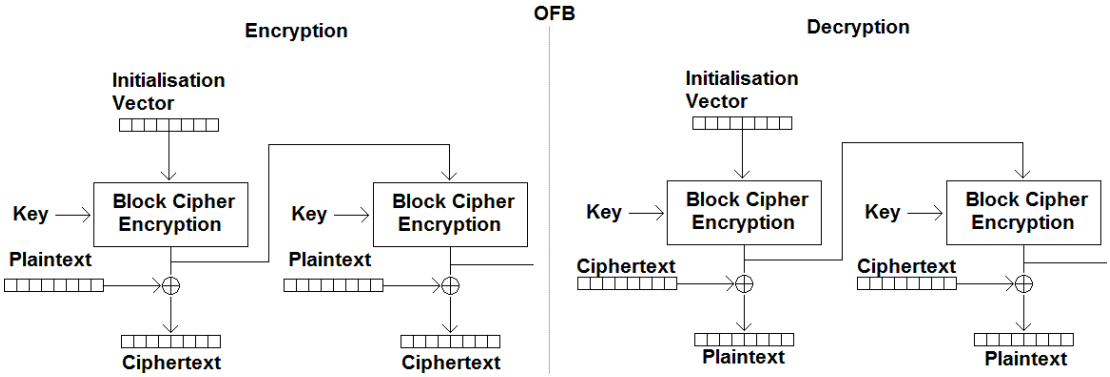


Figure 3-6: Block diagram of the OFB cryptographic mode of operation.

3.3.3.5 *Counter*

Counter mode (CTR) is similar to CFB and OFB in the sense that it uses the output of a block cipher to generate a random bit-stream that is then XORed with the plaintext to form the ciphertext. The input to the block cipher is a unique number, called a nonce, a contraction of *number used once*, concatenated with a counter. It is important not to use the same key / nonce combination as it will leak information about the plaintext.

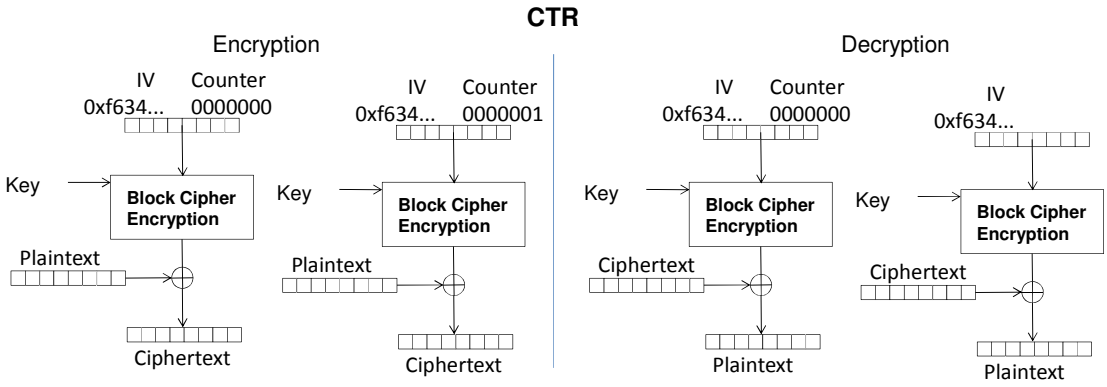


Figure 3-7: Block diagram of the CBC cryptographic mode of operation.

3.3.3.6 *Initialisation Vector*

The choice of initialisation vector (IV) can have a significant impact on the security of an encrypted message. If the same IV is used across several messages and those messages start with the same block, the first block of ciphertext will be the same, this will reveal information to any potential attacker. A random block of data can be generated and used as the IV, this will require the encryption algorithm to have access to a source of randomness, and also, in order to perform the decryption the IV must be known. If it is random then it must be sent along with the message, this increases the size of the ciphertext by 1 block. If there are a large number of relatively short messages this can form a significant overhead.

A better method is to use a cryptographic nonce (a **n**umber used only **once**) to generate the IV, typically this takes the form of a message counter. The nonce must also be sent with the message, this still creates an overhead, but it can be much shorter than a block. It is converted into an entire block by encrypting it with padding.

3.3.3.7 *Summary of Modes of operation*

As the ciphertext of a constant plaintext is always the same with ECB it can leak some information about the data, additionally it is susceptible to a replay attack and it is generally suggested that it not be used [31]. OFB is very similar to CFB, it does have a number of advantages though, errors do not propagate, and both encryption and decryption are exactly the same, significantly simplifying an implementation, this more than makes up for the fact that the decryption cannot be parallelised. CTR, in turn, is preferable to OFB as the random bit-stream generated using CTR will not repeat unless the same nonce, counter and key are re-used; no matter how many times

the encryption is performed. There are also a number of advantages to using CTR over CBC. CTR does not require padding, it can be parallelised arbitrarily and it has a simpler structure. The advantage of CBC is that it is more robust and leaks less information if it is not setup securely.

3.4 Data Encryption Standard

3.4.1 Introduction

The Data Encryption Standard (DES) was developed in the early 1970s by cryptographers at IBM, it is a Feistel cipher based on Lucifer. The National Bureau of Standards (renamed to the National Institute of Standards and Technology (NIST) in 1988) identified a need for an encryption standard to protect unclassified but sensitive government information. After consulting the NSA they solicited proposals for a cipher on 15th May 1973, none of the algorithms were suitable, IBM made their submission after the second request was issued on 27th August 1974. DES uses a 56-bit key and works on 64-bit blocks of data [32].

3.4.2 Structure of DES

As DES is a Feistel cipher the structure is very much like that shown in Figure 3-2, the only difference is there is an initial permutation that re-orders the bits and a final permutation that performs the inverse.

The round function for DES is shown in Figure 3-8. The first stage is the expansion operation that converts the 32-bit half block into 48 bits. This is achieved by duplicating some bits, each 4-bit block of the input provides the middle 4 bits in a 6-bit block of the output, the 2 remaining bits at the edge of the block come from the bits at the edge of the adjacent 4 bit input blocks. This is shown in detail in Table 3-1. The expanded data is then mixed with the key and divided into 8 6-bit blocks which are each put through a different s-box with 4-bit outputs. The 8 4-bit blocks are then re-arranged by a fixed permutation, as shown in Table 3-2. There are a total of 16 rounds in DES

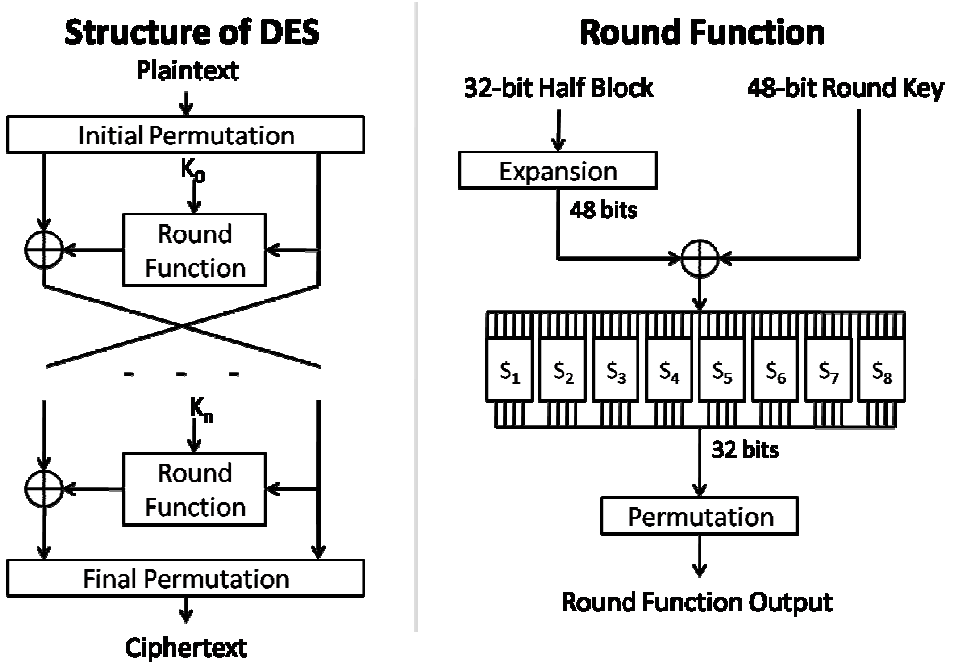


Figure 3-8: The overall structure of DES and its round function.

The 56-bit key is expanded into 16 48-bit blocks, a total of 768 bits. This is achieved by separating the initial 56 bits into two halves, each 28-bit half is then rotated left by either 1 or 2 bits depending on the round, 24 bits are then selected from each half by a fixed permutation. The process is repeated for each round.

Output Bit	Input Bit	Output	Input	Output	Input	Output	Input
0	31	12	7	24	15	36	23
1	0	13	8	25	16	37	24
2	1	14	9	26	17	38	25
3	2	15	10	27	18	39	26
4	3	16	11	28	19	40	27
5	4	17	12	29	20	41	28
6	3	18	11	30	19	42	27
7	4	19	12	31	20	43	28
8	5	20	13	32	21	44	29
9	6	21	14	33	22	45	30
10	7	22	15	34	23	46	31
11	8	23	16	35	24	47	1

Table 3-1: The DES Expansion function.

The inverse of the cipher is very similar, the final permutation is applied first, after that the algorithm is exactly the same except the round keys are provided in the reverse order, finally the initial permutation is applied to the data [33].

Output Bit	Input Bit	Output	Input	Output	Input	Output	Input
0	15	8	0	16	1	24	18
1	6	9	14	17	7	25	12
2	19	10	22	18	23	26	29
3	20	11	25	19	13	27	5
4	28	12	4	20	31	28	21
5	11	13	17	21	26	29	10
6	27	14	30	22	2	30	3
7	16	15	9	23	8	31	24

Table 3-2: The DES Permutation function.

3.4.3 Security of DES

There have been a few attacks that can reduce the complexity of attacking full round DES to lower than that of a brute force attack, although generally not by much, and often they involve collecting large numbers of known or chosen plaintexts. These attacks are discussed briefly in section 3.4.3.1. DES is no longer considered secure as the key length is not long enough to make brute force attacks infeasible with current levels of processing power available.

3.4.3.1 *Theoretical Attacks*

There have been several attacks published on DES. Differential cryptanalysis is a chosen plaintext attack that uses differentials, pairs of plaintexts related by a constant difference, to detect patterns in statistical distribution. It was known to IBM in 1974 and resistance to this type of attack was one of the design goals of the algorithm [34]. When applied to DES differential cryptanalysis requires 2^{47} chosen plaintexts.

Linear cryptanalysis was developed by Matsui in 1992 [16]. It involves generating linear approximations to sections of the cipher that have either a high or low probability of being correct. If bits were chosen at random there would be an

expected probability of ½. It is the deviation from this that provides the cryptanalyst with information. To attack DES using a linear cryptanalysis approach requires 2^{43} known plaintexts [17].

The Davies attack is a statistical attack designed specifically for DES, it was developed by Davies in 1987 [27]. It is a known plaintext attack that exploits the fact that each adjacent s-box shares two input bits that are XORed with different key bits. After collecting enough known plaintext / ciphertext pairs some bits of the key can be calculated. This reduces the complexity of a brute force attack. There is a trade-off between the number of plaintexts, the number of key bits recovered and the probability of success. With 2^{52} plaintexts 24 key bits can be recovered 53% of the time.

3.4.3.2 *Brute Force Attacks*

DES only uses a 56-bit key; this gives $7.2*10^{16}$ possible combinations. In the 1970s this was adequate for brute force to be infeasible. Computers are currently fast enough for this to no longer be true. To highlight this fact RSA Security created a series of contests called the DES Challenges. The first one was in 1997 and was solved by the DESCHALL Project in 96 days, a distributed computing project designed to crack DES. DES Challenge II-1 was solved in 41 days in 1998 by distributed.net, a worldwide distributed computing project that uses the idle time of lots of machines to solve large, computationally intensive problems. DES Challenge II-2 was solved in just 56 hours using Deep Crack, a custom built machine made by the Electronic Frontier Foundation. DES Challenge III was solved as a joint effort between Deep Crack and distributed.net in 22 hours and 15 minutes [29]. Additionally in 2006 the universities of Bochum and Kiel developed COPACOBANA, this retrieves DES keys in an average of 7.2 days and all keys can be tested in 14.4 days [28]. The aim was to get the best cost to performance ratio, as such it is built entirely from off the shelf components. It uses 120 FPGAs (Xilinx Spartan3-1000) and can be built for less than $10,000 [18].

Clearly DES does not provide adequate security against brute force attacks by modern computers and DES is no longer considered secure. In order to increase the security against brute force attacks without having to change to a completely different algorithm a variant of DES was developed called Triple DES, which is discussed in

section 3.5. In 1997 NIST announced the development of a new standard. It was published in 2002 and is called the Advanced Encryption Standard; it is discussed in section 3.6.

3.4.3.3 *Conclusion*

Even though they have a lower theoretical complexity than a brute for attack, the three attacks discussed in section 3.4.3.1 all require a large number of known plaintexts. Linear cryptanalysis requires 2^{43}, differential cryptanalysis requires 2^{47}, and the Davies attack requires 2^{52} just to retrieve 24 key bits 53% of the time, these numbers of plaintexts are not realistic for a real attacker. However, in the strictest sense the algorithm can be described as being broken. Also, the relatively small size of the key compared to the availability of modern processing power enables brute force attacks to be successful in an average of 2 weeks. DES can therefore no longer be considered secure.

3.5 Triple DES

3.5.1 Introduction

Triple DES (TDES) is a derivative of DES that is essentially 3 DES blocks in a row. It was developed as a way to increase the size of the key space provided by DES when it was realised that 56 bits was not enough to ensure security against brute-force attacks with the levels of computing power that had been developed. As it is derived from DES the security of a system can be vastly improved while not having to change the underlying algorithm. TDES is slowly being replaced by the current standard algorithm AES. A notable exception is within the electronic payments industry, which still makes extensive use of TDES.

3.5.2 Structure of Triple DES

The simplest form TDES can take is simply the linking of 3 DES encryption blocks, this is commonly known as EEE, as all steps are encryptions. Generally, in order to make TDES systems more backwards compatible with DES ones, an EDE structure is used, this is one where the 2^{nd} DES block is in decryption mode, so that if all blocks are given the same key the output is the same as that of a single DES block. The structure of EEE and EDE are given in Figure 3-9 a. and b. It is important to note

than when decrypting not only does the operation of each block have to change, but also *k1* and *k3* must be swapped, the structure of EDE decryption is given in Figure 3-9 c. Additionally there are 2 other variants of TDES, a 2-key version where *k1* = *k3* and a 3-key version where *k1*, *k2* and *k3* all have different values.

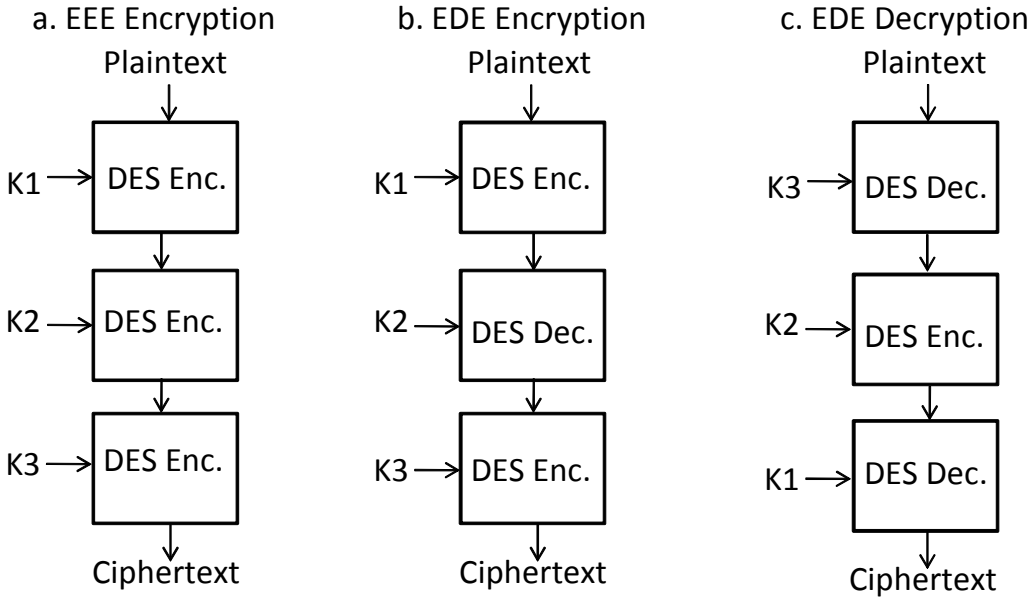


Figure 3-9 : The Structure of TDES for EEE Encryption (a), EDE Encryption (b) and decryption (c).

3.5.3 Security of Triple DES

When trying to increase the key space of an algorithm by using more than one independent key and performing a number of encryption algorithms it might be assumed that the security would square each time the number of encryptions doubled, as an exhaustive search of all possible keys would take 2^{2n} attempts for each key n bits long. In 1977 Diffie and Hellman showed that this wasn't true by developing the *Meet-in-the-Middle Attack [35]*. It is a known plaintext attack where the attacker calculates one encryption of the plaintext for all possible n keys and stores the results. Then the attacker calculates one decryption of the ciphertext for each key in turn, if the result is also in the previous list of results then it is likely that the correct keys have been found, this can then be verified with another plaintext / ciphertext pair. For this reason double DES would not increase the security from 2^{2n}, but to 2^{n+1}.

3-key TDES has a key size of 168 bits, but due to the Meet-in-the-Middle Attack the effective security it provides is only 112 bits. 2-key TDES is susceptible to certain chosen-plaintext [36] or known-plaintext attacks [37] and thus it is officially designated to have only 80-bits of security.

In 1998 Lucks improved the Meet-in-the-Middle attack on triple encryption algorithms in general and TDES in particular [38]. His version requires around 2^{32} known plaintexts, 2^{90} single DES encryptions, and 2^{88} memory.

3.6 Advanced Encryption Standard

3.6.1 Introduction

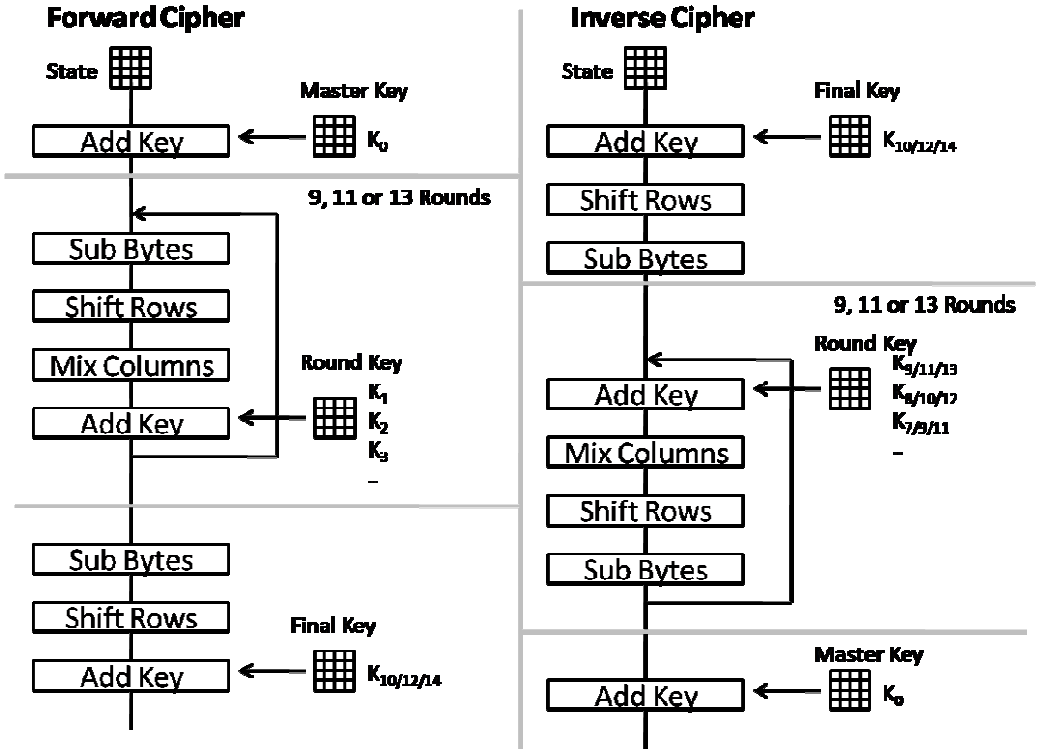


Figure 3-10: The structure of the forward and inverse AES algorithm.

In January 1997 the National Institute of Standards and Technology (NIST) body announced the initiation of the Advanced Encryption Standard (AES) development effort, to create a new standard for a block cipher that would provide secure encryption well into the next century. In September of that year NIST officially announced a call for algorithms to be submitted by the public and evaluated for their appropriateness. NIST stipulated that the algorithm had to work on a 128-bit block size and support key-lengths of 128, 192 and 256 bits. In October 2001 the algorithm

Rijndael, developed by Vincent Rijmen and Joan Daemen [1], was selected to be AES and the standard was published in November 2002 [39]. The structure of AES is shown in Figure 3-10:

The 128-bit input is split into a 4*4 matrix of 8 bits called a state, an example is given in Figure 3-11 and it is put through a number of rounds of operations designed to encrypt the data, the number of rounds being determined by the size of the key, 10 for 128, 12 for 192 and 14 for 256.

00	04	08	0c
01	05	09	0d
02	06	0a	0e
03	07	0b	0f

Figure 3-11: An example of a state.

Each round consists of a number of operations; Sub Bytes, Shift Rows Mix Columns and Add Key, all of the manipulation in these operations are performed in the finite field GF (2^8). GF (2^8) mathematics is explained in section 3.6.2 and the operations are described in detail in sections 3.6.3 - 3.6.7.

3.6.2 Finite Field Mathematics

A finite field is a field, an algebraic construct in which addition, subtraction, multiplication and division can be performed, in which there is a finite number of elements. The order of a field, the number of elements in it, is of the form p^n where p is a prime number called the *characteristic* and n is a positive integer. There is more than one forms of notation for finite fields, for example. F_{p^n}, the notation used in this document is GF (p^n). In this notation GF stands for Galois Field, an alternative name for finite fields named after Évariste Galois who discovered them shortly before his death in a duel in 1832 aged 20 [40]. AES makes use of finite field mathematics, and as previously stated all the normal arithmetic operations can be performed on finite fields. The next section details the specific finite field that is used in AES, explains how it is used and gives examples of its manipulation.

3.6.2.1 *GF (2^8) and AES*

When performing mathematical operations in AES the data is interpreted as being in the finite field GF (2^8). In the normal representation of numbers in binary notation the i^{th} bit of number represents 2^i and the resultant values from all the individual bits are summed to give the total. In GF (2^8) the number represents a polynomial where the i^{th} bit represents $b_i x^i$ where b is a modulo-2 coefficient and i can range from 0 to 7. An example is given in equation (3-1) with its hexadecimal and binary equivalents; it represents the hexadecimal number A7.

$$x^7 + x^5 + x^2 + x + 1 = 0\text{xA7} = 10100111 \qquad (3\text{-}1)$$

When performing addition the coefficients are added, as it is modulo 2 this is equivalent to an exclusive or (XOR), an example is given in lines 2 – 4 of the example in Figure 3-12. When performing multiplication each term in one operand is multiplied by each term in the other by adding the indices and multiplying the coefficients. Coefficients with the same index are then summed modulo-2. The new polynomial might have an order greater than 7; if this is the case then it could not be represented in 1 byte and the order needs to be reduced. This is achieved by representing all results modulo an irreducible polynomial of degree 8. A polynomial is irreducible if its only divisors are one and itself. The irreducible polynomial for AES is shown in equation (3-2).

$$x^8 + x^4 + x^3 + x + 1 = 0\text{x011B} \qquad (3\text{-}2)$$

An example multiplication between the values 0x18 and 0x09 is shown below in Figure 3-12.

1.	$(x^5 + x^3) * (x^3 + x)$	Initial multiplication
2.	$(x^{5+3} + x^{5+1}) + (x^{3+3} + x^{3+1})$	Multiply out brackets
3.	$x^8 + x^6 + x^6 + x^4$	
4.	$x^8 + x^4$	Addition is XOR so x^6 cancels
5.	$(x^8 + x^4) + (x^8 + x^4 + x^3 + x + 1)$	Result has order greater than 8
6.	$x^3 + x + 1$	Result is reduced by the irreducible polynomial

Figure 3-12: An example multiplication in GF (2^8).

3.6.3 Sub Bytes

The Sub Bytes operation is a data dependent substitution of the values in the state. The transform is made up of 2 steps, finding the multiplicative inverse in the finite field GF (2^8) and adding an affine transform. The multiplicative inverse is the number that when multiplied by the original in GF (2^8) and reduced by the relevant irreducible polynomial, gives the value 1 as the answer. An affine transform is a linear transform followed by a translation. A general affine transform is shown in equation (3-3) and the particular one used by AES is shown in equation (3-4).

$$x \rightarrow Ax + C \tag{3-3}$$

$$x \rightarrow 0\text{x}1\text{F} * x + 0\text{x}63 \tag{3-4}$$

The affine transform is implemented by calculating each bit in turn using the formula given in equation (3-5) it required XORing bits from the original number together and then one bit from the constant C.

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i \tag{3-5}$$

3.6.4 Shift Rows

The Shift Rows operation rotates the rows in the state one byte to the left for each row there is above it, i.e. the top row remains the same but the one below it is rotated one byte to the left.

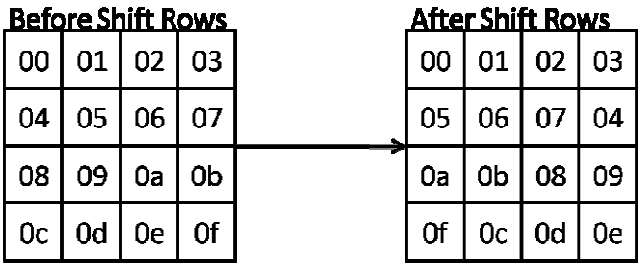


Figure 3-13: The effect of the Shift Rows operation.

3.6.5 Mix Columns

The Mix Columns operation performs a vector dot product on each column in turn with a constant matrix. This is shown in Figure 3-14.

$$\text{Encryption Matrix: } \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

$$\text{Hence: } \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \text{ becomes } \begin{bmatrix} (\{02\} * c_0) \oplus (\{03\} * c_1) \oplus (c_2) \oplus (c_3) \\ (c_0) \oplus (\{02\} * c_1) \oplus (\{03\} * c_2) \oplus (c_3) \\ (c_0) \oplus (c_1) \oplus (\{02\} * c_2) \oplus (\{03\} * c_3) \\ (\{03\} * c_0) \oplus (c_1) \oplus (c_2) \oplus (\{02\} * c_3) \end{bmatrix}$$

$$\text{Decryption Matrix: } \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$$

$$\text{Hence: } \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \text{ becomes } \begin{bmatrix} (\{0e\} * c_0) \oplus (\{0b\} * c_1) \oplus (\{0d\} * c_2) \oplus (\{09\} * c_3) \\ (\{09\} * c_0) \oplus (\{0e\} * c_1) \oplus (\{0b\} * c_2) \oplus (\{0d\} * c_3) \\ (\{0d\} * c_0) \oplus (\{09\} * c_1) \oplus (\{0e\} * c_2) \oplus (\{0b\} * c_3) \\ (\{0b\} * c_0) \oplus (\{0d\} * c_1) \oplus (\{09\} * c_2) \oplus (\{0e\} * c_3) \end{bmatrix}$$

Figure 3-14: The matrices for the encryption and decryption versions of the Mix Columns operation in hexadecimal.

3.6.6 Add Key

Add Key adds the 128-bit round key to the state using an XOR, an example of this is shown in Figure 3-15.

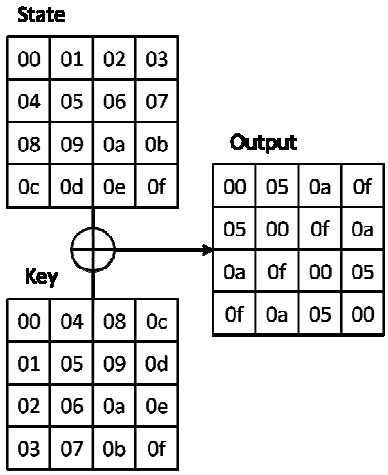


Figure 3-15: An example of the Add Key operation in AES.

3.6.7 Key Expansion

The round key for each round is different and is derived from the secret key by the key scheduler. The number of encryption rounds that a block must go through is determined by the key length, 10 rounds for a 128-bit key, 12 for 192-bit and 14 for a 256-bit key, there is also an initial Add Key operation at the start of the encryption process. The size of the key used in each round key is 128 bits, and the original secret key data is always used first. This means that for a key size of 128-bits a total of 11 keys are needed, as the original secret key is always used first the key scheduler needs to create only another 10 keys, or 1280-bits of data. For a 192-bit key length 13 round keys are required, equivalent to 1472 bits of expanded data. For a key length of 256-bits 15 round keys are needed, requiring 1664 bits of expanded data.

128-bit

00	04	08	0c
01	05	09	0d
02	06	0a	0e
03	07	0b	0f

192-bit

00	04	08	0c	10	14
01	05	09	0d	11	15
02	06	0a	0e	12	16
03	07	0b	0f	13	17

256-bit

00	04	08	0c	10	14	18	1c
01	05	09	0d	11	15	19	1d
02	06	0a	0e	12	16	1a	1e
03	07	0b	0f	13	17	1b	1f

Figure 3-16: Examples of the key matrices for the three different key lengths in AES.

During the expansion process the original key is arranged into a matrix similar in structure to that of the state. Each element has 8 bits and there are 4 rows, the number of columns is determined by the key length, a 128-bit key has 4, a 192-bit key had 6 and a 256-bit key has 8, examples are shown in Figure 3-16. Each expansion round produces a block of data equal in size to the key matrix, so although longer keys need to generate greater amounts of round keys they produce more data in each key expansion round and hence require less of them. 10 rounds are needed for 128 bits, 8 for 192 and 7 for 256 bits.

To perform one round of the key expansion the last column of the previous matrix is rotated downwards, i.e. the bottom byte becomes the top one and the rest are shifted down by one; the values are substituted using the same s-box as during encryption, that column is then XORed with the first column in the key matrix and a column from the constant RCON, as shown in Figure 3-17. To get the rest of the key matrix the previously generated column is XORed with the new columns counterpart from the previous matrix, e.g. the 3rd column in the new block is the 2nd column in the

new block combined with the 3rd column in the old one. An example of the generation of the first 2 columns of a round key is shown in Figure 3-18.The only exception to this is when there is a 256-bit key, in this case another substitution performed on the data in the forth generated column before it is XORed it with the previous matrix's column.

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

Figure 3-17: The constant RCON.

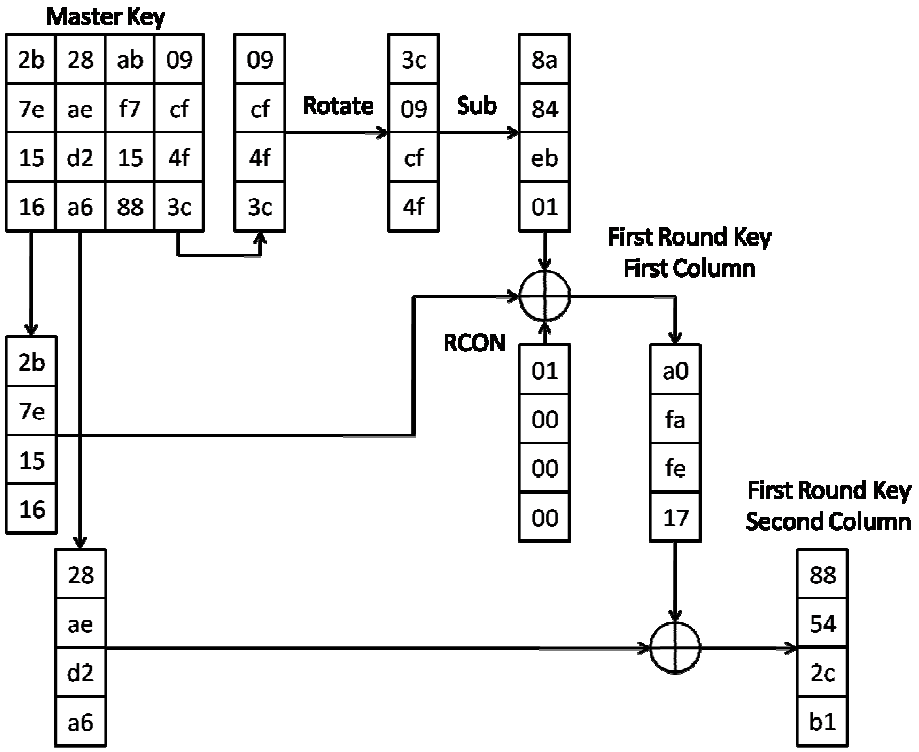


Figure 3-18: Example of the expansion of the first two columns of the first round key of a 128-bit key.

3.6.8 Inverse Cipher

To decrypt data using AES the inverse of the cipher has to be performed, this requires performing the inverse of each operation in the reverse order to the forward cipher. The structure of the inverse cipher is also shown in Figure 3-10. The inverse of Sub Bytes is again a substitution, the inverse of the affine transform is applied and

then the multiplicative inverse of the value is found. The inverse of the Shift Rows operation is exactly the same except the rows are shifted to the right instead of the left. The inverse of the Mix Columns has a similar structure; the only difference is the values in the matrix are now the multiplicative inverses of the original ones. The XOR operation is its own inverse so Add Key remains the same; the only difference is that the first inverse Add Key that is performed has to cancel out the last Add Key that was performed during encryption so the round keys are used in the opposite order.

3.6.9 Implementing the Algorithm

The following sections give details on implementing the various blocks that make up the algorithm and describe some of the reported implementations.

3.6.9.1 *Shift Rows*

The Shift Rows operation is a simple remapping of the order of bytes within the state. This can be accomplished by the way the bytes are wired between the Sub Bytes and Mix Columns operations, e.g. the first byte on the second row of the output of the Sub Bytes becomes the input to the second byte of the second row of Mix Columns. Changing the order of the wiring between blocks does not increase the delay and so there are no reasonable improvements that can be made to this approach.

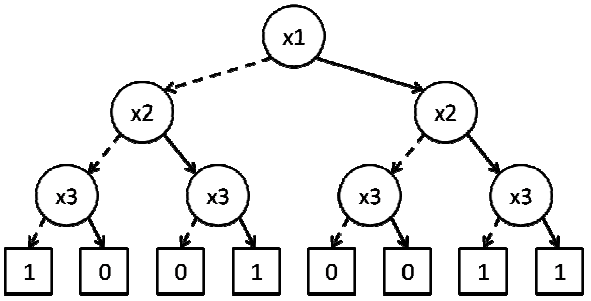
3.6.9.2 *Sub Bytes*

Sub Bytes is the hardest operation as it involves calculating the multiplicative inverse in GF (2^8), which can be computationally intensive [41]. It is possible to calculate it using Euclid's algorithm [42], but this is an iterative process so would require several clock cycles [43]. In order to perform the substitution in one clock cycle a look up table (LUT) is required, this is commonly referred to as an s-box. This approach is not very area efficient. If a standard LUT is implemented in ROM then it requires 256 bytes of memory. The s-box is potentially the most replicated element in an AES implementation as 16 are needed to perform a complete Sub Bytes operation in one clock cycle and an additional four are used to expand the key, this represents a significant proportion of the area. The s-box is also the slowest function [41]. There is an optimisation that is commonly used in software implementations of AES that merges the Sub Bytes and Mix Columns stages of the algorithm by storing modified

s-boxes that give the result of the substitutions multiplied by the relevant constant. The modified s-boxes are called t-boxes [41].

In order to improve the timing performance of the LUT for the AES s, or t-box, Morioka and Satoh [41] developed a twisted *binary decision diagram* (BDD). They reported an increase in speed by a factor of between 1.5 and 2 compared to conventional implementations. A BDD is a rooted, directed, acyclic graph, where each non-terminating node has two directed edges. Each level of nodes represents a different variable and by following the graph to a terminating node it is possible to determine a value for the function represented by the graph. An example BDD is shown in Figure 3-19.

There are a number of characteristics of the s-box that would decrease the performance if it was implemented in a standard BDD form. A large sharing of selectors in the first and second stages of the diagram causes a large fan-out. In the proposed twisted BDD architecture, eight BDDs are arranged in parallel, each corresponding to an output bit. No node is shared between them and their variable ordering is twisted so that each primary input i drives the $((8 - i + j \bmod 8) + 1)th$ input of BDD j. This causes the fan-out to be greatly reduced. Morioka and Satoh reported a delay of 440 ps using 2815 gates in a 0.13-μm technology. This fast s-box allowed them to achieve encryption rates of 11.6 Gbits/s without using pipelining.



x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Figure 3-19: An example Binary Decision Diagram and associated Truth Table.

There are other more space efficient implementations of the s-box. Rijmen [44] suggested calculating the multiplicative inverse of a GF (2^8) value by converting it into a polynomial of degree 1 with coefficients in GF (2^4). Denoting the irreducible polynomial used for multiplication as $x^2 + Ax + B$ and the converted polynomial as $bx + c$, the multiplicative inverse is given by:

$$(bx+c)^{-1} = b(b^2B + bcA + c^2)^{-1}x + (c+bA)(b^2B + bcA + c^2)^{-1} \quad (3\text{-}6)$$

A flow diagram showing the required operations is given in Figure 3-20. This approach still requires calculating the multiplicative inverse of a GF (2^4) value. This is a much easier problem requiring a smaller LUT as there are only 16 possibilities for 4 bits. This creates a much smaller s-box, but greatly increases the critical path. This optimisation was developed further by Hodjat and Verbauwhede [45] by adding pipelining inside the s-box. Pipelining is discussed in section 3.6.9.7. The speed area trade-off that they reported on a 1.8-µm technology is shown in Figure 3-21.

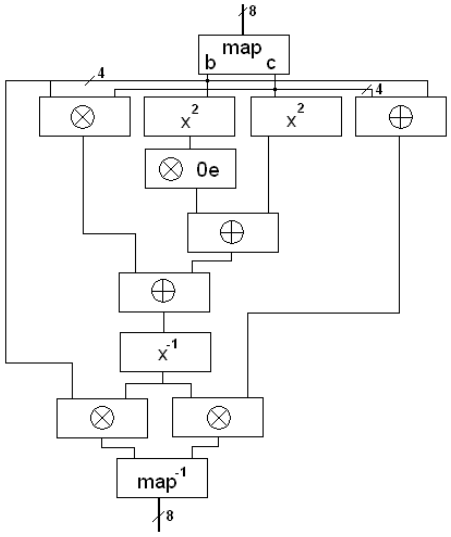


Figure 3-20: Calculating the multiplicative inverse in GF (2^8) using GF (2^4).

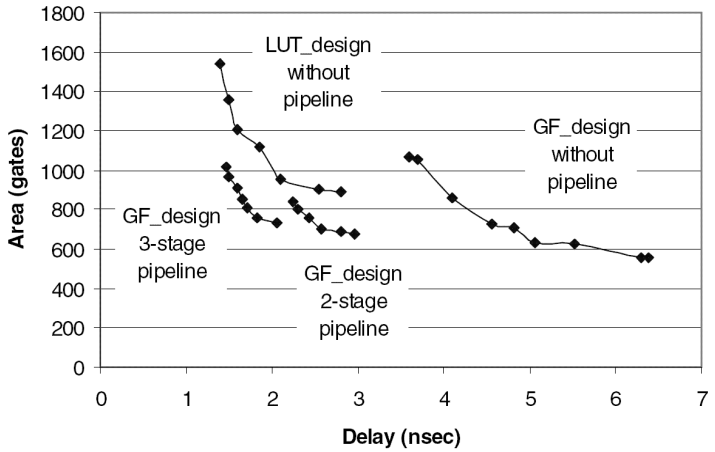


Figure 3-21: The speed area trade-off for different s-boxes using a 1.8-µm technology. [45]

3.6.9.3 *Add Key*

The Add Key operation is only an XOR, there is no reasonable way that is can be further optimised.

3.6.9.4 *Mix Columns*

Mix Columns is made up of multiplications and additions. Additions, as in the Add Key module are simply XOR gates and as stated in section 3.6.9.3 additions are too simple to be further optimised. Multipliers are more complicated and the design of them is discussed in the following section.

3.6.9.5 *Multiplication*

c_7	$(a_7.b_0) \oplus (a_6.b_1) \oplus (a_5.b_2) \oplus (a_4.b_3)$
c_6	$(a_6.b_0) \oplus (a_5.b_1) \oplus (a_4.b_2) \oplus (a_3.b_3) \oplus (a_7.b_3)$
c_5	$(a_7.b_3) \oplus (a_7.b_2) \oplus (a_6.b_3) \oplus (a_5.b_0) \oplus (a_4.b_1) \oplus (a_3.b_2) \oplus (a_2.b_3)$
c_4	$(a_7.b_2) \oplus (a_6.b_3) \oplus (a_4.b_0) \oplus (a_3.b_1) \oplus (a_2.b_2) \oplus (a_1.b_3)$
c_3	$(a_7.b_3) \oplus (a_7.b_1) \oplus (a_6.b_2) \oplus (a_5.b_3) \oplus (a_3.b_0) \oplus (a_2.b_1) \oplus (a_1.b_2) \oplus (a_0.b_3)$
c_2	$(a_7.b_3) \oplus (a_7.b_2) \oplus (a_6.b_3) \oplus (a_2.b_0) \oplus (a_1.b_1) \oplus (a_0.b_2)$
c_1	$(a_7.b_2) \oplus (a_6.b_3) \oplus (a_7.b_1) \oplus (a_6.b_2) \oplus (a_5.b_3) \oplus (a_1.b_0) \oplus (a_0.b_1)$
c_0	$(a_0.b_0) \oplus (a_7.b_1) \oplus (a_6.b_2) \oplus (a_5.b_3)$

Table 3-3: The equations for a generic 8-bit by 4-bit GF (28) multiplier.

In order to design a multiplier it is important to understand how multiplication in GF (2^8) is performed, this is described in detail in section 3.6.2. When two finite field elements are multiplied together they cause an effect in the element representing the sum of the value of their individual elements, e.g. $x^3 * x^4 = x^7$. Therefore each element is the addition (XOR) of each possible combination of pairs of bits, one from each multiplicand, whose element numbers sum to the value of the element in question. This could give a number that is larger than 2^8 so it has to be reduced modulo an irreducible polynomial, the one that is used is given in equation (3-2). The 8^{th} element of the irreducible polynomial is '1'; this means by adding it to a number that is too large it can be used to cancel the 8^{th} element. Any other values can be

cancelled by multiplying it by the relevant power of *x*. The equations for a generic 8-bit by 4-bit GF (2^8) are shown in Table 3-3.

It is important to note that when performing the multiplication in the mix columns operation one of the operands is a constant. This can lead to a slightly smaller design by creating a series of separate fixed value multipliers. The bits of the polynomial *x* that must be XORed together for each bit of the polynomial y for each of the constant multipliers are given in Table 3-4 [46]. This only reduces area if the multipliers are going to be replicated enough times to mix at least one column in a clock cycle.

	02	03	09	0b	0d	0e
y_7	x_6	$x_6\ x_7$	$x_4\ x_7$	$x_4\ x_6\ x_7$	$x_4\ x_5\ x_7$	$x_4\ x_5\ x_6$
y_6	x_5	$x_5\ x_6$	$x_3\ x_6\ x_7$	$x_3\ x_5\ x_6\ x_7$	$x_3\ x_4\ x_6\ x_7$	$x_3\ x_4\ x_5\ x_7$
y_5	x_4	$x_4\ x_5$	$x_2\ x_5\ x_6\ x_7$	$x_2\ x_4\ x_5\ x_6\ x_7$	$x_2\ x_3\ x_5\ x_6$	$x_2\ x_3\ x_4\ x_6$
y_4	$x_3\ x_7$	$x_3\ x_4\ x_7$	$x_1\ x_4\ x_5\ x_6$	$x_1\ x_3\ x_4\ x_5\ x_6\ x_7$	$x_1\ x_2\ x_4\ x_5\ x_7$	$x_1\ x_2\ x_3\ x_5$
y_3	$x_2\ x_7$	$x_2\ x_3\ x_7$	$x_0\ x_3\ x_5\ x_7$	$x_0\ x_2\ x_3\ x_5$	$x_0\ x_1\ x_3\ x_5\ x_6\ x_7$	$x_0\ x_1\ x_2\ x_5\ x_6$
y_2	x_1	$x_1\ x_2$	$x_2\ x_6\ x_7$	$x_1\ x_2\ x_6\ x_7$	$x_0\ x_2\ x_6$	$x_0\ x_1\ x_6$
y_1	$x_0\ x_7$	$x_0\ x_1\ x_7$	$x_1\ x_5\ x_6$	$x_0\ x_1\ x_5\ x_6\ x_7$	$x_1\ x_5\ x_7$	$x_0\ x_5$
y_0	x_7	$x_0\ x_7$	$x_0\ x_5$	$x_0\ x_5\ x_7$	$x_0\ x_5\ x_6$	$x_5\ x_6\ x_7$

Table 3-4: The bits that must be XORed together to calculate each bit for the constant multipliers.

3.6.9.6 *Key Scheduler*

There are two main types of key scheduler that have been reported, offline and online. In an offline approach all of the round keys are generated at the start and stored in memory, whereas in an online approach round keys are generated as they are required. If the encryption architecture is unrolled then all the round keys are needed on any given clock cycle. An offline key scheduler can reduce the area requirements if the key does not change very often compared to the data. This is because the advantage gained by unrolling the key scheduler is minimal so it is more efficient to calculate the values once and store them. If the encryptor is not unrolled then there is

no space advantage to making the key scheduler offline as there would only be the need to generate one key at any given time anyway so storing them would only waste space. Also the memory access time could potentially be greater than the time it takes to generate the keys online. It is important to note that as the keys are needed in the reverse order when performing decryption the online key scheduler is only appropriate for a device that only performs encryption.

3.6.9.7 *Pipelining*

There are two approaches to pipelining, outer-pipelining and inner-pipelining. Outer-pipelining involves the addition of registers between rounds so that multiple blocks of data can be processed in parallel. An architecture is said to be fully pipelined if the number of pipeline stages, k, is equal to the number of rounds. If an architecture is only partially pipelined it can process k blocks in the same number of clock cycles as there are rounds, and after k clock cycles data has to be fed back round from the final stage. For this reason k is generally chosen to be a factor of the number of rounds otherwise when one block has been completely processed the new data would have to be added to a non-constant point in the pipeline and this would increase the complexity of the controller. Due to the need to replicate round blocks in order to process more than one data block at once the area of the pipelined architecture is proportional to k.

Inner-pipelining is similar except the registers are inserted inside the combinational logic of a round block. If there are n blocks with the same delay then the inner pipelining can achieve an increase in speed of almost a factor of n, with only a marginal increase in area. The minimum clock period is determined by the longest critical path between registers, so dividing blocks that are not the longest has no effect on the clock speed.

The effects of pipelining on the performance of an AES implementation were investigated by Hodjat and Verbauwhede in [47]. They designed and simulated different implementations for an AES processor using a 1.8-µm technology. One with both inner and outer pipelining, one with only outer pipelining and a third design that has 5 pipeline stages that each contain 2 rounds and take 2 clock cycles to complete. The rounds and key generators were split into four sections for the inner pipelining. The throughput and area for different implementations are shown in Figure 3-22. The

effect that pipelining has on the area is less than the effect that pipelining has on the speed, this can be seen from the graph below**.** When moving up to an implementation with more pipelining the throughput is increases by a greater factor than the number of gates. For example, the difference in the number of gates for the largest multi-round pipeline implementation and the largest inner and outer round pipelined implementation is slightly greater than 2 while the throughput increases by nearly a factor of 4.

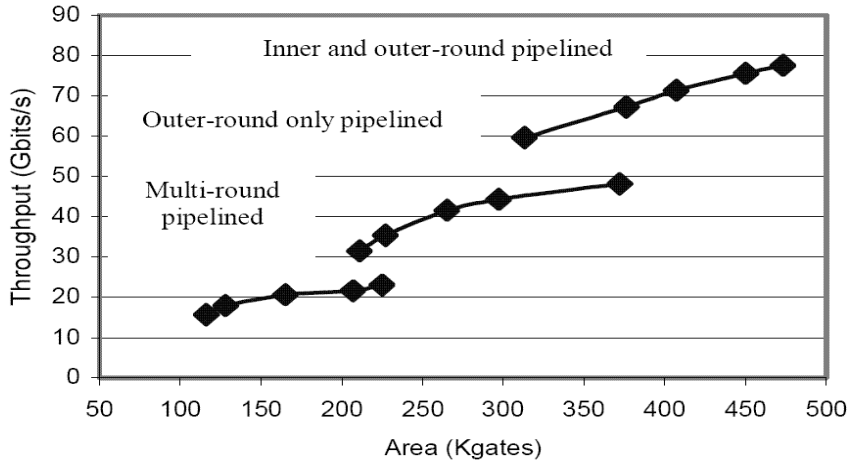


Figure 3-22: The Area-throughput trade-off for a 1.8-µm AES implementation. [45]

It is important to note that if pipelining is used it limits the cryptographic modes of operation that can be used with the device to ECB only. With modes like CBC the previous ciphertext is XORed with the plaintext, so an entire encryption must be complete before the next one is started. This is the reason that pipelining was not included in the design reported by Morioka and Satoh in [41].

3.6.10 Reported Performance of Hardware Implementations

There have been several implementations of AES developed that use different optimisations in order to improve the performance of the hardware in some. The results for ASIC implementations are summarised in Table 3-5 and the FPGA implementations in Table 3-6.

The implementations using t-boxes rather than s-boxes produced faster chips, an increase in throughput of nearly 30% compared to similar implementations that used s-boxes [41]. There is a price to pay in area though, as it increased by 350%. This is because in order to run one column through the Sub Bytes and Mix Columns requires 4 s-boxes and 8 multipliers, or 12 t-boxes and t-boxes are much larger than

multipliers. To perform decryption 4 s-boxes and 16 multipliers, or 16 t-boxes are required.

Description	Throughput (Gbits/s)	Clock (MHz)	Gates	Tech. (μm)
LUT s-box full AES offline key [48]	1.64	465	28626	0.18
GF (2^4) s-box, offline key, full AES [49]	2.381	200	58430	0.35
GF (2^4) s-box, offline key, full AES [50]	2.977	250	63400	0.25
BDD s-box 128 bit dec. no pipe [41]	8.9	699	61841	0.13
BDD t-box 128 bit dec. no pipe [41]	11.3	885	282494	0.13
BDD t-box 128 bit enc. no pipe [41]	11.6	909	167566	0.13
128 bit enc. multi pipelining [47]	23.1	362	222000	0.18
128 bit enc. outer pipelining [47]	48.2	377	482000	0.18
128 bit enc. both pipelining [47]	77.6	606	471000	0.18

Table 3-5: Reported ASIC implementation performances.

In both [51] and [52] they have made a similar encryptor / decryptor combination and encryptor pair. In [51] the throughput of the implementation is reduced by a factor of 4 and the area requirements increase by two thirds. In [52] the area more than doubles and the speed halves although it is later implemented on a more complex FPGA. Supporting more than one key length can also significantly reduce the performance of AES chips due to increased complexity.

Description	Throughput (Gb/s)	Clock (MHz)	Slices	FPGA
Small low cost Enc/Dec [53]	0.208	71.5	163	XC3S50
Small low cost Enc/Dec [53]	0.358	123	146	XC2V40
Generic 128 bit Enc. [52]	0.310	25.4	4681	XCV600E
LUT s-box 128 bit enc/dec [51]	0.463	76	5150	XCV1000E
LUT s-box 128 bit enc [51]	1.604	125.38	1857	XCV1000E

Description	Throughput (Gb/s)	Clock (MHz)	Slices	FPGA
Full pipeline 128 bit enc/dec [52]	3.239	25.3	7576	XCV3200E
Full pipeline 128 bit enc [52]	6.956	54.35	2222	XCV812E
GF (2^4) 3 stage inner pipeline [54]	9.184	71.8	9406	XCV800
GF (2^4) 3 stage inner pipeline [54]	11.965	93.5	9406	XCV812E
GF (2^4) 7 stage inner pipeline [54]	16.032	125.3	11014	XCV1000
GF (2^4) 7 stage inner pipeline [54]	21.556	168.4	11022	XCV1000E
Full pipe online key 128 bit enc [55]	16.54	129.2	11719	XCV1000E
Full pipe online key 128 bit enc [55]	17.8	139.1	10750	XC2V2000

Table 3-6: Reported FPGA implementation results.

3.6.11 Testing and Validation of AES

In order to check the validity of an AES implementation NIST created the Advanced Encryption Standard Algorithm Validation Suite (AESAVS) [56]. It is designed to perform automated testing of an implementation, using Known Answer Test (KAT), the Multi-block Message Test (MMT), and the Monte Carlo Test (MCT). The KATs can be split into four groups GF s-box, key s-box, variable key and variable plaintext. In variable key tests the plaintext is always made entirely of zeros and the key is made of increasing number of contiguous ones starting from the left hand side. The relevant ciphertexts are given for all of these, for all of the possible key lengths. Similarly the keys for the variable plaintext tests are made entirely of zeros and the plaintext is made of an increasing number of ones.

MMT tests the implementation's ability to correctly process multi-block messages. These require the chaining of information between consecutive blocks. Several different modes of operation are tested by MMT, namely: ECB, CBC, OFB, and Cipher Feedback with 128, 8 and 1 blocks of data (CFB128, CFB8 and CFB1). The block length is 8 bits for CFB8, 1 bit for CFB1 and 128 bits for the others. For each supported mode 10 messages are supplied with lengths of $i * blocklength$, where $1 \leq i \leq 10$.

In the MCT, the implementation under test encrypts 100 plaintexts iteratively 1000 times, by feeding the generated ciphertext back round using the appropriate method for the cryptographic mode under test. Hence any MCT test involves 100,000 encryption, or decryption, operations, this requires a long simulation.

To perform the test a request file is generated that contains all of the plaintexts, keys and initialisation vectors, for the tests. The implementation then reads in this data, processes it and creates a response file. The data in the response file is then verified with a trusted implementation of AES.

3.6.12 Security of AES

There have been no attacks of full strength AES, however there have been some concerns voiced over its security. AES has a simple algebraic structure and while this has not yet led to the discovery of any vulnerabilities it has been criticised as a potential weakness [57]. AES is based on the algorithm Square, also designed by Rijmen and Daemen [58], in the specification they include a potential attack that utilises the byte oriented structure of the algorithm, this is described in more detail in section 4.2.1. The basic square attack can only break four rounds but it can be extended to up to eight [30]. It is possible that it could be potentially be extended further in the future. The algorithms itself achieves good levels of confusion and diffusion, the same cannot be said for the key-schedule [30], this is discussed further in section 4.2.2. Finally, like all block ciphers, AES is susceptible to power analysis attacks and Differential Power Analysis is able to retrieve the key [3, 59-61], this is discussed further in section 4.3.2.

3.7 Conclusion

Block ciphers were the first type of cipher to be developed in the modern era of digital cryptography. The foundations were laid by Shannon in the 1949 and they have evolved significantly since the 1970s, increasing in size and complexity and becoming immune to several different classes of attacks along the way. They are a valuable and versatile weapon in the cryptographer's arsenal, being able to secure messages sent to trusted recipients, protect files and even encrypt arbitrary length streams of data.

In the 1970s DES was developed and it remained the standard block cipher for nearly three decades, until computing power and cryptanalysis had advanced to such a degree that it was no longer deemed secure. After the cracks started to appear its successor was developed, the Advanced Encryption Standard (AES). There have been no published mathematical attacks on full round AES and although there have been criticisms of some elements of the design it is accepted to be currently secure. There is one class of attacks that no algorithm can currently claim to be immune from and that is side channel attacks, these are explored in chapter 4.3.

Chapter 4 Security of Algorithms

4.1 Introduction

There are several different techniques for cryptanalysis. The attacks all make assumptions about how much information can be observed by the attacker and what kind of access he has to the device. It is generally assumed that the structure of the algorithm is known by the attacker. Keeping the algorithm secret is a dangerous way to try and ensure security, there is no real guarantee that an attacker could not acquire an implementation of the algorithm and reverse engineer it, or find some other way to get the details. The complete details of AES have been published and are in the public domain. When it was being developed as well as being evaluated by various US government security agencies it also went through a system of public review to ensure it was secure. If an algorithm stands up to public review then there is more faith in its security and if a weakness is discovered then it reported.

The majority of analyses focus on algorithmic weaknesses. There is another group of attacks called side channel attacks; these use information gained from analysis of emissions from the physical cryptosystem, and can result in the extraction of the secret key or some important intermediate values. As the attacks use information generated during the specific encryption an attacker, or some of their equipment, must be present when that encryption was performed. Generally side channel attacks fall into one of the following groups:

- **Timing attacks** – the attacker exploits the fact that some computation time for some operations is data-dependent. This attack applies more to asymmetric ciphers [19].
- **Power consumption based attacks** – the attacker uses variations in the power consumption during the encryption to try and retrieve key data from the chip.
- **Emitted electromagnetic radiation attacks** – the attacker uses the emitted electromagnetic radiation to try and gain information about what is going on inside a cryptographic chip [20].
- **Acoustic cryptanalysis** – the attacker uses the acoustic noise emitted by the keyboard during data entry [62], or by the hum of the processor during a cryptographic operation [21].

Section 4.2 reviews the general security of AES, it contains details of a proposed attack, outlines a possible weakness and an improvement for the key schedule. The rest of the section gives the background theory and some examples of power analysis attacks. This chapter is mostly a review of current research with some analysis to draw the ideas together.

4.2 Security of AES

Although no successful attacks on a complete implementation of AES have been published some concerns have been expressed over its security. Section 4.2.1 describes the square attack, an attack that was identified in the original paper about the square algorithm on which Rijndael, and hence AES, is based [58]. Section 4.2.2 is about the security of key schedules in general and the AES one in particular.

4.2.1 The Square Attack

While no attacks on full round AES have been published, there have been some concerns of the security of AES due to the simplicity of its algebraic structure [57]. The byte-oriented structure of AES causes it to be susceptible to an attack known as the Square Attack. It was published in the original paper that proposed the algorithm Square, on which AES is heavily based [58]. It is a chosen plaintext attack that recovers the last round sub-key of a reduced round AES [1], the basic attack can break four rounds but it can be extended to up to eight [30].

The plaintexts are chosen to have a specific number of active and passive bytes, where in this context all passive bytes have the same value and all active ones have a different value. The plaintexts are chosen in groups of 256 so that the active bytes vary over the range of all possible values. The sub bytes and add key operations do not change the positions of the active bytes. The mix columns operation creates a column of active bytes if there is at least one active byte in the column, the next shift rows operation then spreads these active bytes into all four columns so after the second mix columns there are four columns of only active bytes. As the values of active bytes range over all possible values, the inputs to the third round are balanced over each input set, i.e. the bitwise XOR of all the values of an active byte in the set of chosen plaintexts is 0. As it is a reduced round AES and the 4^{th} round is the final round it does not include a Mix Columns operation. This means the output bytes of the 4^{th} round each depend on a single input byte of the 4^{th} round and are given by the following formula:

$$4th\_output_{i,j} = Sbox(4th\_input_{i',j'}) \oplus sub\_key\_byte_{i,j} \quad (4\text{-}1)$$

The input bytes to the 4^{th} round are balances over the set of chosen plaintexts. By assuming a value for the sub-key byte, the value of the input byte for each chosen plaintext in the set can be calculated from the ciphertexts. If these values are not balanced, the hypothesised sub-key byte was incorrect. This can then be repeated for all possible values of all bytes of the sub-key.

By increasing the number of plaintexts the attack can be extended up to an eight-round attack. The key schedule of AES is not a one way function and exhibits bit-leakage, this means that the Square Attack can be used to recover all the sub-keys including the master key from knowledge of a single *n*-round sub-key.

4.2.2 The Security of the Key Schedules

When designing cryptographic algorithms, lots of care is given to the design of the cipher itself, assuring it quickly reaches sufficient levels of diffusion and confusion, two properties related to the overall cryptographic strength of a cipher, defined by Shannon in [14]. Key schedule design receives much less attention, with the majority of block ciphers having *ad hoc* designed ones [63]. This is despite the fact that the complexity of the key schedule can have a significant impact on a

cipher's susceptibility to linear and differential cryptanalysis. Knudsen and Mathiassen [63] demonstrate using experiments on small, simplified ciphers that the complexity of the key schedule influences the probability of differentials and linear hulls (the linear hull of a set *S* is the intersection of all subsets in a field that contain *S*). These affect a cipher's susceptibility to differential and linear cryptanalysis. They argue that the more complex the key schedule the greater the resistance to these types of attack.

In [30], May *et al* provide a list of three properties that are necessary for a key schedule to be efficacious, they are as follows:

1. **Collision-resistant one-way function:** If the key schedule is a one-way function then it will not be possible for an attacker to gain information about the master key or other sub-keys from a known sub-key. It may also be easier to find weak keys and related keys for key schedules which are not one-way [64].
2. **Minimal mutual information:** This property aims to eliminate bit leakage between sub-keys and the master key. Leakage of information to an adjacent sub-key is impossible if property 1 is satisfied. The direct use of master key bits in sub-keys gives worst case bit leakage; however this can be easily avoided.
3. **Efficient implementation:** The cipher algorithm and the key schedule should complement each other in implementation aspects as well as security. By re-using already optimised components of the encryption algorithm and with some careful consideration during the key schedule design, a fast implementation is attainable, without the necessity for major additional cost in circuitry or code size due to design constraints.

4.2.2.1 *Analysis of AES Key Schedule*

The main weakness in the AES key schedule is that given knowledge of a sub-key (or part of one), knowledge of other sub-keys (or parts) is derivable, i.e. there is significant bit leakage. This is due to the fact that a column is XORed with its equivalent in the previous sub-key to get the next column, and hence knowledge of two adjacent columns leads to knowledge about the previous sub-key. The iterative

nature of the sub-key generation leads to good computational efficiency but the iteration is too simplistic leading to the bit leakage problem.

Sub-key	Freq	SAC
1	0.0000	125.053
2	0.0000	105.433
3	0.0000	72.563
4	0.0000	46.858
5	0.0593	31.840
6	0.0000	28.057
7	0.0000	28.153
8	0.0034	28.237
9	0.0000	28.161
10	0.0110	28.215

Table 4-1: AES key schedule Crypt-X statistical test results [30].

Round	Freq	SAC
2	0.0000	96.083
3	0.0048	20.687
4	0.7560	1.183

Table 4-2: AES cipher Crypt-X statistical test results [30].

As well as not fulfilling the three necessary properties for a strong key schedule defined in section 4.2.2, the AES key schedule performs poorly, in contrast to the rest of the cipher, in terms of quickly achieving acceptable levels of confusion and diffusion. To show this May *et al* used two statistical tests, the frequency test and the *Strict Avalanche Criterion* (SAC) test, available in the software package Crypt-X. The frequency test is used to test the randomness of a sequence of zeroes and ones, more specifically in this context it is being used to test the level of confusion, the influence of each key bit on the output bits, achieved by the algorithm. The result of this test is a probability, where a value greater than 0.01 / 0.001 indicates that bit mixing is satisfied with a confidence of 99% / 99.9%. The SAC test measures the level of diffusion, the degree of change in the output after the change of a single bit of the input. This is tested with the *Kolmogorov-Smirnov* test (KS test). The KS test is a goodness-of-fit test used to determine whether two sets of samples come from the same probability distribution. It can be used to determine whether the underlying

probability distribution for a finite set of samples differs from a hypothesized distribution, in this case, that the probability of each output bit changing is 0.5 after the change of a single input bit. A value less than 1.628 / 1.949 indicates that bit diffusion is satisfied with a probability of error of 1% / 0.1%.

It is evident from the above tables that the AES cipher achieves confusion and diffusion by round 4 but the majority of the sub-keys do not achieve complete bit mixing and hence do not achieve significant levels of confusion. Additionally none of the sub-keys satisfy the SAC test.

4.2.2.2 *Improved AES Key Schedule*

In order to combat these weaknesses May *et al* [30] designed a new key schedule for AES. In order to maximise the efficiency of the new design, functions from the AES cipher are used in the new key schedule. The key schedule takes the master key, adds a round constant, and this value is put through three rounds of AES using itself as the key, a more detailed description of the algorithm is given in Figure 4-1.

for $round = 0$ to 10

  for $j = 0$ to 15

    $KS\ Plaintext_j = KS\ Round\ Key_j = Master\ Key_j \oplus$ Sub Bytes $((round * 16) + j)$

  for $i = 0$ to 2

    Sub Bytes

    Shift Rows

    Mix Columns

    Add Key

Figure 4-1: Pseudo-code for the improved AES key schedule [30].

The performance of both the cipher with the new key schedule and the schedule itself were measured in Crypt-X and the results are reported in Table 4-3 and Table 4-4, Table 4-5 repeats the results for AES without the key schedule modifications. The results show that both confusion and diffusion in the new key schedule reached significant levels after three rounds, and this increased the speed with which the new algorithm also meets these criteria.

Round	Freq	SAC
2	0.1557	15.775
3	0.8757	1.212
4	0.3498	1.689

Table 4-3: Crypt-X results for the new 128-bit key schedule [30].

Round	Freq	SAC
2	0.0000	96.083
3	0.0048	20.687
4	0.7560	1.183

Table 4-5: Crypt-X results for normal AES [30].

Round	Freq	SAC
2	0.0000	21.113
3	0.2663	1.282
4	0.3110	1.347

Table 4-4: Crypt-X results for 128-bit AES with new key schedule [30]

Instead of adding Round Keys to the cipher round in the key schedule, which would require a separate key schedule, the Master Key is used. This does not adversely affect the security. One potential worry might be that it adds a vulnerability to power analysis attacks (for a detailed description of these see section 4.3), but this is unfounded as these attacks require the interaction of key data with chosen or known data. In the case of the new key schedule it is only XORed with *KSPlaintext*, which is ultimately derived from the Master Key anyway.

The use of the cipher assures that the key generation is one way. Also as each of the sub keys are generated independently and the master key is not used as one of them, there is no bit leakage and knowledge of one sub-key does not give knowledge of the others.

Although the main aim of the work was to make a key schedule that fulfilled the properties outlined in section 4.2.2 May *et al* [30] also report that the new key

schedule improves the resistance to several reduced round cryptanalysis techniques such as differential cryptanalysis and the Square attack.

4.3 Power Analysis Attacks

Currently the vast majority of electronics are made using CMOS technology. This has the advantage of having low static power consumption; the dynamic power consumption is a much more significant component. This means that there is a relatively high amount of correlation between the power consumption and both the operations that the chip is performing and the data that is being operated on. This allows a cryptanalysis technique called power analysis, where by observing the power consumption of a device when performing encryption or decryption can yield information about the algorithm, implementation and the secret key. Most of the different techniques for performing power analysis involve measuring the power consumption while encrypting or decrypting a large number of known plaintexts or ciphertexts, combining the input with a guess at a byte of the key, and using the fact that there is a large data set to enable a statistical test as to the correctness of the guess. This is then repeated for all guesses of all bytes of the key, the values that appear to be the most correct are assumed to be the key. More detailed descriptions of specific power analysis techniques are given in the following sections.

4.3.1 Simple Power Analysis

Simple power analysis (SPA) involves directly interpreting power consumption measurements collected during a cryptographic process [2]. It is possible to identify which instruction is being executed by a microprocessor by inspecting the power consumption trace. This can provide an attacker with information about the key if the execution path is data dependent, for example if there are conditional branches based on key data, multiplication and exponentiation can also leak significant amounts of data via SPA. Some microprocessors also have heavily operand-dependent power consumption features. These systems can have serious SPA vulnerabilities, even if the execution path is not key dependent [2].

There are several techniques for the prevention of SPA that are fairly easy to implement. Avoiding the use of secret intermediate values or key values as the conditions for branching will remove a lot of the useful information that is leaked. If

such branches are inherent in the algorithm this can require a coding techniques that can negatively affect performance. Most ASIC implementations of symmetric cryptographic algorithms have sufficiently small variations in power consumption to not leak information about the key material via SPA [2].

4.3.2 Differential Power Analysis

Differential power analysis (DPA) is a statistical attack that uses power consumption data from a large number of encryptions to retrieve secret information about the key. DPA has proved to be a powerful cryptanalysis technique that has been able to extract the secret key from several DES implementations [2]. The DPA algorithm is presented below [2]:

1. A set of *N* plaintexts are randomly generated.
2. The power consumption during the encryption of the *N* plaintexts is measured. The attacker gets N traces each containing *n* values.
3. A hypothetical model of the chip is fed with the plaintexts (or ciphertexts) and a guess at one byte of the first (or last) sub-key.
4. A selection function, *D*, is applied to the output of the hypothetical model which separates the traces into two sets.
5. The average of both sets is computed and the difference between the averages is calculated.
6. Steps 3 to 5 are repeated for each sub-key guess. This will give 2^8 differential traces.
7. For each differential trace the peak and mean value is determined and the ratio between the two is calculated.
8. For a correct sub-key guess there will be large peaks seen in an otherwise flat differential trace.
9. To get all the sub-keys, steps 2 to 8 are repeated 16 times (for a 128-bit key).

The choice of hypothetical model determines the section of the algorithm that is being attacked. It takes the input or output to that section, generally a section of the plaintext or ciphertext, and a guess at one byte of the relevant sub-key and outputs

either the output or the input to section. The selection function separates the plain- or ciphertexts, and therefore their associated power traces, into two sets. Kocher's original hypothetical model and selection function D (C; b; K_s) [2] attacked the left hand intermediate at the beginning of the 16th round. It accepted the ciphertext, C, a 6-bit sub-key guess, K_s, to predict the output and a value between 0 and 31 representing which bit of the DES intermediate was being attacked, b, as inputs. The selection function applies the ciphertext and sub-key guess to an inverse DES algorithm and returns either a 1 or a 0 depending on the value of the b^{th} bit that would give these values. Varying the value of b modulus 4 targets different sub-bytes of the key, as in DES there are 8 s-boxes each with a 4-bit output. Kocher was using DPA to analyse DES; Schuster reported that while the original selection function used by Kocher on DES works with the AES power consumption model it was unsuccessful with real test data [3] and proposes a new one based on the Hamming weight of the output of the s-box, if it is greater than four then the trace is added to one set, if it is not then it is added to the other. Schuster uses this to successfully crack an AES implementation that is being run on an 8-bit microcontroller

4.3.2.1 *Leakage Based Differential Power Analysis*

As CMOS technology shrinks in size the leakage power becomes a more significant portion of overall power consumption. While leakage power is mainly dependent on physical parameters its dependence on input patterns becomes significant in sub-90 nm technology [65], therefore leakage power needs to be considered when evaluation a system for susceptibility to DPA. Lin and Burleson took this into account and developed "Leakage-based" DPA (LDPA) [66].

The LPDA algorithm is essentially the same as the regular DPA algorithm except the power traces that are recorded capture both the dynamic power and the leakage power. The attack was tested on a SPICE simulation of an implementation of DES and it revealed the correct key after 120 traces using 45 nm CMOS, compared to 200 traces for regular DPA using 180 nm CMOS.

4.3.2.2 *Correlation as the Statistical Test in DPA*

The DPA attack described in section 4.3.2 uses a statistical test called the difference-of-means. The distance-of-mean test simply takes the difference between the mean of two sets of data, it assumes that the variances of the two data sets are the

same and not much information from the model can be included. Other tests have been proposed, including analysis of variance (ANOVA), which can simultaneously compare the means of several sets of data and works better than the distance-of-mean test [67]. This section discusses the use of correlation in DPA using the Pearson correlation coefficient. It was first described by Brier *et al* in [68]. This coefficient reflects the degree of linear relationship between two random variables, it can be used to provide a direct comparison between the real and hypothetical model of the device. It is defined as the sum of the products of the standard scores of the two measures divided by the degrees of freedom. This is equivalent to dividing the covariance between the two variables by the product of their standard deviations as shown in equation (4-2).

$$\rho = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} \tag{4-2}$$

In order to calculate an estimate of the correlation from a number of samples the formula in equation (4-3) must be used.

$$\rho = \frac{N\sum_{i=1}^{N} X_i Y_i - \sum_{i=1}^{N} X_i \sum_{i=1}^{N} Y_i}{\sqrt{N\sum_{i=1}^{N} X_i^2 - (\sum_{i=1}^{N} X_i)^2}\sqrt{N\sum_{i=1}^{N} Y_i^2 - (\sum_{i=1}^{N} Y_i)^2}} \tag{4-3}$$

The coefficient ranges from -1 to 1, the sign indicating the direction of the relationship. If the coefficient has the value 1 then a linear equation describes the relationship perfectly and positively, all data points lie on the same line and Y increases with X. A value of -1 means a linear equation describes the relationship perfectly but negatively, i.e. all data points lie on a single line but Y increases as X decreases. A correlation value of 0 means that there is no linear relationship between the variables.

The technique described by Kocher in [2] attacks an algorithm by predicting the value of one bit and partitions the traces accordingly. The method proposed by Brier is a multi-bit attack; it predicts the number of bits that change in a byte of registers. This means that the technique involved is slightly different from regular DPA. It has three stages, prediction, measurement and correlation, a description is given below [61]:

1. Prediction Stage

a. Predict the number of bit changes inside a number of targeted registers in a specific clock cycle.

b. Repeat this for all 2^8 possible values of a byte of the key and for N different randomly chosen plaintexts.

c. Put them in $N * 2^8$ matrix. This is called the Prediction Matrix

2. Measurement Stage

a. Measure the power consumption over all (C) clock cycles in the encryption process

b. Record the highest power consumption in each clock cycle in an $N * C$ matrix. This is called the Consumption Matrix

3. Correlation Stage

a. Calculate the correlation between the column representing the clock cycle that was targeted in the prediction phase in the Consumption Matrix and each column in the Prediction Matrix.

b. The column of the Prediction Matrix that shows the greatest correlation is the one that represents a correct key guess.

It is possible to perform this type of attack using purely simulated data. This requires using a more detailed hypothetical model of the device that can be used to predict the bit changes in all of the registers in the device for all cycles and entering the data into an $N * C$ Prediction Matrix. This is then used instead of the Consumption Matrix in the Correlation Stage.

4.3.2.3 *Choice of Target in Differential Power Analysis*

Both forms of power analysis attack a specific point in an algorithm. In DPA the position of this is selected by the choice selection function and in a correlation attack the choice of which register to target is explicitly made. This section defines the properties that determine whether a particular register is an appropriate target for the attack. Figure 4-2 shows a diagram of the AES algorithm with all the possible positions of registers between the stages. It shows which of the registers in the design have the properties that make them suitable for the target of a DPA attack.

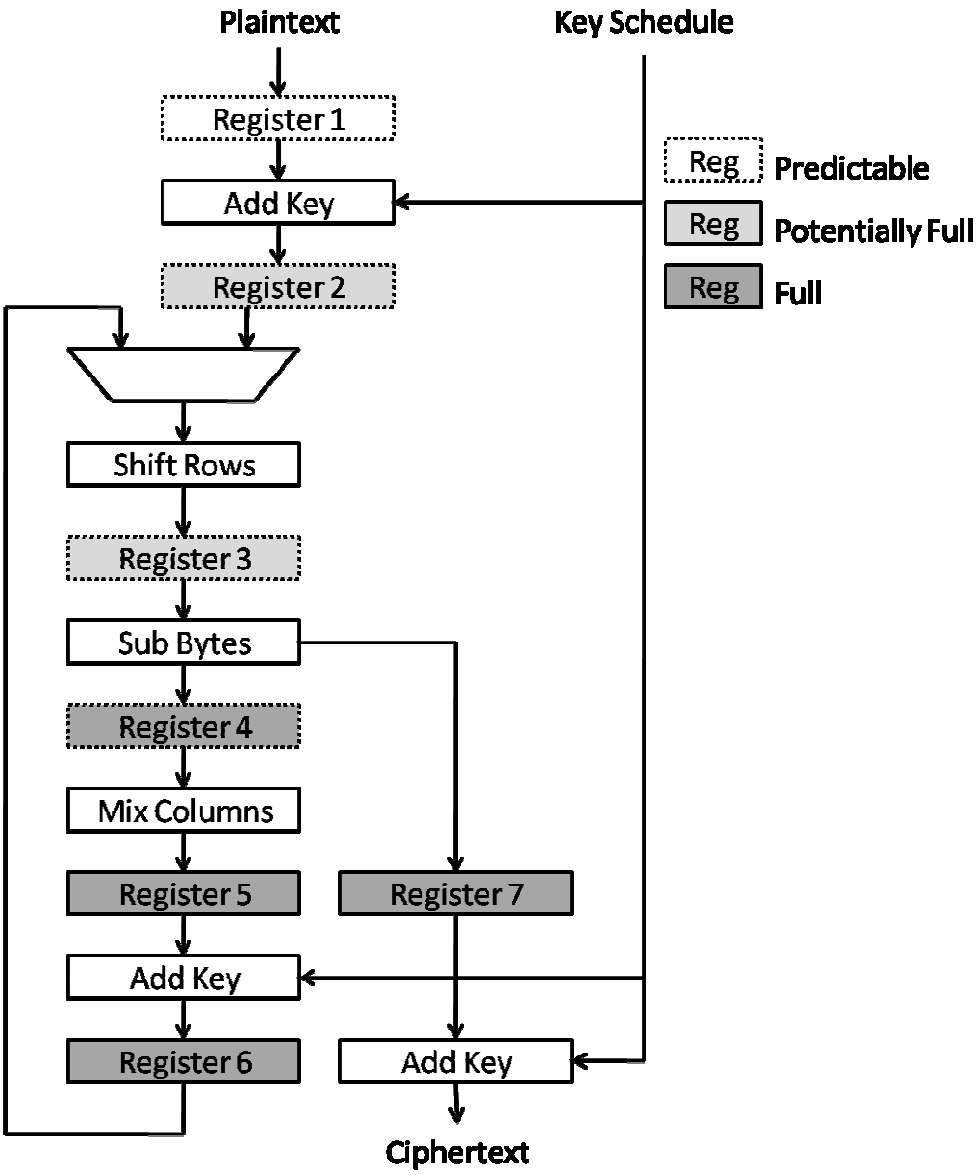


Figure 4-2: Diagram showing the predictability and fullness of registers at different points in AES.

Both forms of power analysis find the correct key value by testing all possible key values and finding the value whose result best fulfils the attack's selection criterion, the target must therefore be determined by a small enough number of key bits for this to be computationally feasible. In practise this limit is assumed to be 16 bits [60], below this a register is said to be *predictable*. In AES the s-boxes are 8 bits wide; this gives 256 different key values to test which is easily performed. The Mix Columns operation mixes the data from 4 bytes; this means the output depends on 32 key bits, above the predictability limit.

A register is described as *full* if it leaks information about the key via its transitions. This is also a property required in order to make a register a valid target. As seen in Figure 4-2 register 1 does not leak information as it only contains plaintext data. Interestingly, registers 2 and 3 do not necessarily leak information either as the influence of the key on the transition cancels out over two successive plaintexts as illustrated in equation (4-4). They can be made to be full by resetting the contents to 0s between plaintexts. Also they can be full in smart card implementations where there is a constant instruction address loaded.

$$Reg2_1 \oplus Reg2_2 = (plaintext_1 \oplus key) \oplus (plaintext_2 \oplus key) \qquad (4\text{-}4)$$

$$= plaintext_1 \oplus plaintext_2$$

Registers after the s-box will all be full as the non-linearity of the substitution stops the influence of the key on the transition value over 2 successive plaintexts cancelling.

4.3.3 Inferential Power Analysis

Fahn and Pearson have also developed a type of power attack that is similar to DPA [69]. It is called inferential power analysis (IPA) and consists of two stages, a long, computationally intensive profiling stage and a shorter key extraction stage. It has the advantage over DPA that the attacker does not need to know the plain or ciphertexts relating to the recorded power traces in order to perform the attack. The first step in the profiling stage is to record a large number of power consumption traces, between 100 and 1000 are generally required. The traces do not need to have the same key, although for simplicity when it was performed in [69] the key was kept constant.

1. The traces are aligned so that the power consumptions are all matched.
2. These matched traces are averaged to create a Mean Trace.
3. The Mean Trace is chopped into rounds to give Mean Rounds.
4. The Mean Rounds are averaged to give a Super-Average Round.
5. The difference between each Mean Round and the Super-Average Round is computed, this gives the Differential Traces.
6. The mean squares of the Differential Traces are calculated.

The first averaging of all the traces that have been collected removes the effect of the plaintext, but, in the case of the constant key example, leaves the key bits. Averaging the different rounds removes the effect of the key bits on the data; this will leave only the code features. These are cancelled out by calculating the difference between the average round and the super-average. After this, only the effects of the specific sub-key k_i remain. The mean square of the differential traces contains peaks at the locations of the key bits. The number of peaks that can be seen in the final traces can differ from the number of bits in the sub-key, but there should be a simple mathematical relationship determined by the specific implementation details of the device, for example, the binary compliment of the value.

After the locations of the key bits have been identified each key bit has to be connected to its specific position. How this is actually achieved depends on the algorithm that is used and can be quite complicated. For an algorithm like DES where there are no fixed rules about the order inherent in the algorithm it can get very complicated if the most obvious guesses as to the order have failed. In situations like that it can be useful to examine the specification of the key scheduling in the algorithm to gain additional information that can help, although this obviously negates the advantage that details of the algorithm do not need to be known.

It is useful to observe the distribution of recorded power levels at the peaks that are indicated by the first stage of the profiling. If the peak is in the correct place and it represents the manipulation of a single bit of key data, which will be either 0 or 1 with a probability of ½, then it should have a bimodal distribution, with each mode representing either a 0 or a 1. If more than one bit is being handled then there should be a binomial distribution, the shape of which indicates how the bits are being handled and gives information relating to the Hamming weight of the key bit grouping.

IPA has several advantages over DPA, the attacker does not need to know the plaintext (or ciphertext), removing the possibility of simply shielding this data in order to prevent the attack. DPA is restricted to examining points where the plaintext and key interact directly, generally limiting analysis to the first few rounds whereas IPA can probe all rounds of an algorithm. After a lengthy profiling stage IPA can simply perform a fast key extraction phase on all similar hardware, greatly reducing

the computational overhead when attacking data that has been encrypted with several different keys.

There are a few countermeasures that can make performing IPA harder. Avoiding handling key data one bit at a time will remove some of the data. Randomising the order of execution of the code and adding random delays to the system will cause problems with alignment and creating a system with an offline key scheduler may offer some resistance to IPA.

4.3.4 High-Order DPA

High order differential power analysis (HODPA) is a variation of DPA in which instead of finding the statistical properties of the signal at each sample time the attacker can use the joint statistics across several sample times to use data from multiple intermediate values [70].

This can be used in order to defeat whitening DPA countermeasures. In order to try and defeat DPA intermediate values can be masked by XORing them with randomly generated numbers, this de-correlates the Hamming weight from any key data and so information about it is not leaked. Obviously in order to still give the correct result for the calculation the data must be unmasked, this is again achieved by XORing the data with the previously generated number. By combining information gathered from the power consumption trace at the point where mask is generated and the point where it is removed it is possible to compensate for the masking and uncover information about the key. If a duplication countermeasure with k shares is used then the attacker needs to mount a k^{th} order attack.

Although using a higher order approach to DPA has its advantages it also has a number of disadvantages, if the standard deviation of the noise is the same at all of the n sample positions then the product has the standard deviation of the original raised to the power of n, this increases the amount of noise and hence the number of traces that is required to recover information. In order to extract information using DPA it is important to know which point in the samples relates to the intermediate value that is being attacked. In HODPA the effect of the intermediate value that is being attacked exerts influence on several points in the traces, but to take advantage of this the positions of all of these correlated points must be known. In first order DPA this problem is avoided by calculating the entire differential trace, a computationally un-

intensive operation. The natural higher order generalisation of this technique, to calculate the differential traces with each sample correlated to every other sample in turn, can quickly become prohibitively expensive.

Waddle and Wagner proposed two methods for second order DPA, one for where the correlation time is zero, or known, and one for when it is non-zero and known [71]. Zero-Offset 2^{nd} Order DPA is based on the assumption that the intermediate values that are the point of attack occur at the same time. This is not necessarily an unrealistic assumption, for example a parallel processor that calculates both the random and the masked bits simultaneously. It works by squaring the values of the samples in the power traces before performing regular DPA. This leads to a related attack, Known-Offset 2^{nd} Order DPA [71], where instead of calculating the square the lagged product is calculated, i.e. the sample multiplied by the value of the sample one offset later.

If the offset is non-zero and is not known then a Fourier transform can be used to auto-correlate the trace. This is achieved by calculating the squared L2-norm of the Fourier transform of the DPA trace; this involves multiplying the complex value generated by the FFT by its complex conjugate. The inverse Fourier transform of this data set is then calculated. This is repeated for all traces and this is summed for all of the traces within a particular bit-guess group. Values are only non-zero for correct correlations. The noise from the other traces significantly contributes to the standard deviation, so this attack is only practical for short traces.

4.3.5 Mathematics of Differential Power Analysis

DPA uses statistical techniques to gain information about the encryption key. This section discusses the mathematics behind the way secrets are leaked. Section 4.3.5.1 defines the leakage model for DPA and discusses the way XORing known data with an unknown constant reveals data. Also the correlation for a system with a given signal to noise ratio is derived and it is shown that, assuming the correlations are worked out to a high enough level of accuracy, the attack will always give the correct answer. Section 4.3.5.3 defines a new property of s-boxes called the t*ransparency order* it is the degree that an s-box leaks information about the key, then it is shown that an s-box that prevents linear and differential cryptanalysis in an optimal way has a very poor transparency order.

4.3.5.1 *Statistics of Secret Leakage*

Power analysis attacks use statistical techniques to exploit the leakage of secret data via the power consumption. As they are statistical attacks it is important to understand the statistics of the secret leakage model which is shown in equation (4-5) and was investigated by Brier *et al.* in [68].

$$W = aH(D \oplus R) + b \tag{4-5}$$

Where R is the information the attacker is trying to extract, D is the state the target register was in from the previous clock cycle, H represents the Hamming distance function, a is the linear gain between the Hamming distance and the power consumption of the register, b is the noise and W is the power consumption of the device. As the noise is, by definition, uncorrelated with the data dependent power consumption, and the variance of the sum of two independent variables is the sum of the component variances we get equation (4-6).

$$\sigma_W{}^2 = a^2\sigma_H{}^2 + \sigma_b^2 \tag{4-6}$$

The Brier *et al.*'s model requires a number of assumptions:

- The same amount of energy is required for the transition from 0 to 1 and 1 to 0.
- All bits in the target register are balanced and require the same amount of energy for transitions.

These are reasonable assumptions to make. If these assumptions are incorrect then this will reduce the linear correlation between the registers' power consumption and the total power consumption, which is analogous to there being more noise. An important point is that ($D \oplus R$) is a uniform variable; this means that the Hamming weight is binomially distributed, the binomial distribution being a discrete approximation to the normal distribution, with an average value of $m/2$ and a variance ($\sigma_H{}^2$) of $m/4$, where m is the number of bits in R.

From the signal to noise ratio of the power consumption it is possible to calculate the population correlation of the leakage and power consumption. As the number of traces used in a correlation attack increases the correlation of the power consumption with the predictions made using the correct value of the key becomes a

more accurate estimate of the population correlation of the system. The correlation between two variables is defined as the ratio of their covariance to the product of their standard deviations (equation (4-2)), hence:

$$\rho_{WH} = \frac{\mathrm{cov}(W,H)}{\sigma_W \sigma_H} \tag{4-7}$$

$$\rho_{WH} = \frac{\mathrm{cov}(aH+B,H)}{\sigma_W \sigma_H} = \frac{\mathrm{cov}(aH,H)+\mathrm{cov}(B,H)}{\sigma_W \sigma_H} \tag{4-8}$$

The noise is assumed to be uncorrelated to the Hamming distance, leading to:

$$\rho_{WH} = \frac{a * \mathrm{var}(H)+0}{\sigma_W \sigma_H} = \frac{a\sigma_H{}^2}{\sigma_W \sigma_H} \tag{4-9}$$

$$\rho_{WH} = \frac{a\sigma_H}{\sigma_W} \tag{4-10}$$

The SNR is given by the ratio of the standard deviations of the signal and the noise as defined in equation (4-11).

$$SNR = \frac{a\sigma_H}{\sigma_b} \tag{4-11}$$

Combining with equations (4-6) and (4-10) the relationship between correlation and SNR can be derived.

$$\rho_{WH} = \frac{a\sigma_H}{\sqrt{\sigma_w{}^2}} = \frac{a\sigma_H}{\sqrt{a^2\sigma_H{}^2+\sigma_b{}^2}} \tag{4-12}$$

$$\rho_{WH}{}^2 = \frac{a^2\sigma_H{}^2}{a^2\sigma_H{}^2+\sigma_b{}^2} \tag{4-13}$$

$$\frac{1}{\rho_{WH}{}^2} = \frac{a^2\sigma_H{}^2+\sigma_b{}^2}{a^2\sigma_H{}^2} = \frac{a^2\sigma_H{}^2}{a^2\sigma_H{}^2} + \frac{\sigma_b{}^2}{a^2\sigma_H{}^2} = 1 + \frac{1}{SNR^2} \tag{4-14}$$

$$\rho_{WH} = \frac{1}{\sqrt{1+1/SNR^2}} \tag{4-15}$$

Correlation based DPA analysis is based on the assumption that the correlation of the correct key value with the power consumption will have the greatest value. This

can be shown to be true by examining the correlation of an incorrect value denoted by H' with the power consumption.

$$\rho_{WH'} = \frac{\text{cov}(aH + b, H')}{\sigma_W \sigma_{H'}} \tag{4-16}$$

$$\text{cov}(aH + b, H') = \text{cov}(aH, H') + \text{cov}(b, H') \tag{4-17}$$

As H' and b are independent:

$$\text{cov}(aH, H') + \text{cov}(b, H') = a * \text{cov}(H, H') \tag{4-18}$$

$$\rho_{WH'} = \frac{\sigma_H}{\sigma_H} \frac{a * \text{cov}(H, H')}{\sigma_W \sigma_{H'}} = \frac{a\sigma_H}{\sigma_W} \frac{\text{cov}(H, H')}{\sigma_H \sigma_{H'}} = \rho_{WH} \rho_{HH'} \tag{4-19}$$

As the correlation between an incorrect key guess and the power consumption ($\rho_{WH'}$) is equal to the correlation for a correct key guess (ρ_{WH}) scaled by the correlation between the correct and incorrect guess ($\rho_{HH'}$), which is necessarily less than 1, the correct guess will always have the highest correlation. Assuming that value that gives H' has the same value as the one that gives H except for k bits, e.g. for H (0xE4 $\oplus$ R) and H' (0xE3 $\oplus$ R), k is 3, then the Hamming weights of the two values are given by equations (4-20) and (4-21).

$$H = H_{m-k} + H_k \tag{4-20}$$

$$H' = H'_{m-k} + H'_k = H_{m-k} - H_k + k \tag{4-21}$$

Where H_{m-k} is the Hamming weight of the bits that are the same in both and H_k is the Hamming weight of the bits that are different. As k is constant:

$$\text{cov}(H, H') = \text{cov}(H, H' - k) = \text{cov}(H_{m-k} + H_k, H_{m-k} - H_k) \tag{4-22}$$

From the expected values this gives

$$\text{cov}(H, H') = \left\langle H_{m-k}^2 \right\rangle - \left\langle H_k^2 \right\rangle - \left\langle H_{m-k} \right\rangle^2 + \left\langle H_k \right\rangle^2 \tag{4-23}$$

We know that $\left\langle H_k \right\rangle = \frac{k}{2}, \left\langle H_{m-k} \right\rangle = \frac{m-k}{2}$ as the mean of the Hamming weight of a word will always be half of the number of bits in the word assuming the values are evenly distributed. Additionally, $\left\langle H_k^2 \right\rangle = \sigma_{H_k}^2 + \left\langle H_k \right\rangle^2$ is a standard result that can

be derived from the definition of variance. Substituting these values into equation (4-23) gives:

$$\mathrm{cov}(H, H') = \frac{m - 2k}{4} \tag{4-24}$$

And hence:

$$\rho_{WH'} = \rho_{WH}\left(\frac{m - 2k}{m}\right) \tag{4-25}$$

As stated earlier with an increasing number of traces the estimate of the population correlation becomes more accurate. The sampling distribution of correlation, how much the correlation will vary with the number of samples, is approximately normally distributed when the correlation is close to zero. As the value of the correlation is bounded between -1 and 1 there is a skew. If the value is positive then it can extend further in the negative direction than in the positive and *vice versa*. After Fisher's transform is applied it becomes normal with a standard error of $1/\sqrt{N-3}$ where N is the number of traces. Fisher's transform is given in (4-26) [72].

$$z = 0.5\ln\left(\frac{1+\rho}{1-\rho}\right) \tag{4-26}$$

As the population correlation decreases with the addition of more noise, the margin between a correct and an incorrect key guess decreases and it is more likely that the variation due to the random nature of the variables will overshadow it. Therefore the more noise there is in the system the greater the number of traces that must be used in order to get the same level of confidence in the accuracy of a result.

4.3.5.2 *Lower Bound for the Number of Traces Needed to Perform DPA*

In [73] Mangard uses a statistical model of DPA to determine a lower bound for the number of traces required to successfully identify the key from a system that implements DPA countermeasures. He considers countermeasures that reduce the SNR and countermeasures that change the time the intermediate result is processed. The maximum correlation between the correct key hypothesis and the power consumption is defined in equation (4-27):

$$\rho_{\max} = \frac{\rho(H, H')}{\sqrt{1 + 1/SNR^2}} * p' * \sqrt{\frac{\mathrm{var}(P)}{\mathrm{var}(P')}} \tag{4-27}$$

Where *p'* is the probability that the power consumption at the sampling point is due to the processing of an attacked intermediate, *P* is the power consumption due to the processing of an attacked intermediate and *P'* is the power consumption of the device at the sample time.

Mangard reasoned that the number of samples required to correctly identify the key is determined by the distance between the sampling distributions with means of 0 and ρ_{max} as all of the values will be taken from one of these distributions, the greater the overlap between the two distributions greater the chance of the incorrect correlation appearing higher than the correct one. The amount of overlap can be reduced by increasing the number of traces used as this will reduce the standard deviation of the distributions. Using equation (4-28) Mangard calculated the probability of value drawn from $\rho = \rho_{max}$ distribution being higher than one from the distribution $\rho = 0$.

$$\alpha = \Phi\left(\frac{1/2 \ln\left(\frac{1+\rho_{\max}}{1-\rho_{\max}}\right) - 1/2 \ln\left(\frac{1+0}{1-0}\right)}{\sqrt{\frac{2}{N-3}}} \right) \tag{4-28}$$

$$N = 3 + 8\left(\frac{Z_\alpha}{\left(\frac{1+\rho_{\max}}{1-\rho_{\max}}\right)} \right)^2 \tag{4-29}$$

Equation (4-28) can be transformed into (4-29) to directly calculate the number of samples required, where Z_α is the quantile that determines the distance between the distributions. Quantiles are evenly spread points in a cumulative probability distribution, this marks the boundaries between consecutive sub-sets. There is a probability of *k* / *n* that a value drawn from a distribution is lower that its *k*th *n*-tile. In actual DPA several values are drawn from the distributions, for AES 255 are drawn from $\rho = 0$ for each $\rho = \rho_{max}$, these values are not independent so getting an exact

probability for a peak is difficult. Based on a series of experiments with different values of α it was determined that $\alpha = 0.9$ is a reasonable lower bound for the number of samples, $\alpha = 0.9999$ leads to a number of samples that has a high probability of revealing the attacked sub-key. Between those two values it is less clear.

4.3.5.3 *S-Boxes and DPA*

In [74] Prouff studied the effects the s-box has on the resistance of an algorithm to DPA. He defined a new property called the *transparency order* for an s-box. He showed that when s-boxes are optimally resistant to linear and differential cryptanalysis they perform inherently poorly in terms of their transparency order.

In order to derive the transparency order for a function first we must introduce some mathematical and notational preliminaries. F is an (n, m) function, that is a function that maps from GF (2^n) to GF (2^m), v is a vector in GF (2^m) and u is a vector in GF (2^n). The *sign function* is defined in equation (4-30), it is a Boolean function, the output of this function is either 0 or 1.

$$v \cdot F = \frac{1}{2} - \frac{1}{2}(-1)^{v \cdot F} \tag{4-30}$$

The Fourier transform of the sign function of F is defined by the *Walsh function* W.

$$W_F(u, v) = \sum_{x \in GF(2^n)} (-1)^{v \cdot F(x) + u \cdot x} \tag{4-31}$$

A mapping function F is *balanced* if the weight of the function, the sum of the outputs of the function across all inputs, equals 2^{n-1}, i.e. there is an equal number of 1s and 0s at the output. This is a requirement for a function to be a secure cryptographic primitive. A function is balanced if and only if W_F (0, v) equals zero for every vector $v \in$ GF (2^m). *Bent functions* are another set of Boolean functions, they are maximally non-linear and have only balanced non-zero derivatives. They are not balanced so they can not be used as cryptographic primitives but they do resist linear and differential cryptanalysis in an optimal way [74]. Another important concept is that of a *derivative*, they are used in differential attacks. The derivative of a function F with respect to the vector a is an (n, m)-function that maps x to $F(x) + F(x + a)$:

$$D_a F : x \rightarrow F(x) + F(x + a) \tag{4-32}$$

As stated in section 4.2.2.1 in order to have good levels of diffusion a function must satisfy SAC, it was generalised to the *Propagation Criterion* (PC) by Preneel in [75]. In order for a function F to satisfy $PC(l)$ at a high level D_aF must be balanced for every vector a of weight at most l. The correlation coefficient between two functions Boolean f and g is given by:

$$corr(f,g) = \sum_{x \in GF(2^n)} (-1)^{f(x)+g(x)} \tag{4-33}$$

It is also important to note that the correlation coefficient between the function that maps x to $v.F(x)$ and the function that maps x to $v.F(x+a)$ is the Walsh function of the derivative of F. This relationship is also expressed in equation (4-34).

$$corr(x \to v \cdot F(x), x \to v \cdot F(x+a)) = W_{D_aF}(0,v) \tag{4-34}$$

The rest of this section derives the transparency order for an s-box and describes the properties of the transparency of a few important functions. For the purposes of this section the power consumption of a cryptographic device is defined as:

$$C_K(X) = cH(\alpha(X,K) + F_K(X)) + b \tag{4-35}$$

Where b is the noise, c is the energy required to switch one bit from 0 to 1 or 1 to 0, α is the data on the device before the targeted transition and F_K is the data that replaces it K is the key and X is the plaintext.

We can define the single bit correlation attack as follows:

$$\Delta_{K,K}(K) = \frac{c}{2^n} \sum_{u \in GF2^n, H(u)=1} corr(v \cdot F_K, u \cdot (F_{K} + \alpha)) \tag{4-36}$$

In this case u is a vector where only 1 bit is 1 and the rest are 0, these serve to select a single bit in the vector $(F_k + \alpha)$, K is a round key guess and $\dot{K}$ is the correct value of the key. This can be generalised to a multi-bit attack by summing the contributions of each bit in v.

$$\delta_{K}(K) = \left| \sum_{v \in GF2^n, H(v)=1} \Delta_{K,K}(K) \right| \tag{4-37}$$

It becomes much more difficult to perform a successful DPA attack on an s-box when the peaks are not high enough for the correct values to be distinguished from the

incorrect ones. This is the case if the error in the computation of the correlation δ is larger than the average value given in equation (4-38).

$$D(\dot{K}) = \frac{1}{2^n - 1} \sum_{K \in GF\,2^n - \{\dot{K}\}} (\delta_{\cdot K}(\dot{K}) - \delta_{\cdot K}(K)) \tag{4-38}$$

In order to prevent differential and statistical attacks, cryptographic algorithms are designed so that round functions with different keys are as uncorrelated as possible, hence it is reasonable to assume that c*orr* ($v.F_{K,}\, u.\ F_{\cdot K}$) equals zero, unless $u = v$. Also if we assume that α and F are independent *corr* ($v.F + u.F_{\cdot K}$, $u.\alpha$) is equal to zero unless $v.F + u.F_{\cdot K}$ is constant. Taking equation (4-31) and (4-33) into account this leads us from equation (4-36) to equation (4-39)

$$\Delta_{\cdot K,K}(K) = \frac{c(-1)^{v.(F_K + F_{\cdot K})}}{2^n} W_\alpha(0,v) \tag{4-39}$$

Hence:

$$\Delta_{\cdot K,K}(\dot{K}) = \frac{c(-1)^{v.(F_{\dot{K}} + F_{\cdot K})}}{2^n} W_\alpha(0,v) = \frac{c}{2^n} W_\alpha(0,v) \tag{4-40}$$

This leads us from equation (4-38) to equation (4-41).

$$D(\dot{K}) = \frac{c}{2^n} \left| \sum_{v \in GF\,2^n, H(v)=1} W_\alpha(0,v) \right| - \frac{1}{2^n - 1} \sum_{K \in GF\,2^n - \{\dot{K}\}} \delta_{\cdot K}(K) \tag{4-41}$$

Now if we assume that the function α equals the constant value β, then due to equation (4-31), equation (4-42) is true. This is a realistic assumption if the device uses pre-charge logic, where the registers are cleared between operations, or the previous value represent the op-code for a microprocessor.

$$W_\beta(0,v) = (-1)^{v.\beta} * 2^n \tag{4-42}$$

Hence:

$$\sum_{v \in GF\,2^n, H(v)=1} W_\beta(0,v) = 2^n (n - 2H(\beta)) \tag{4-43}$$

As 1 is not added but subtracted for each non-zero bit in β, the total is has a maximum value of n. Also, if α is a constant, then from equation (4-36) we get equation (4-44), as constants make no contribution to the correlation.

$$\Delta_{\dot{K},\dot{K}}(v) = \frac{c}{2^n}(-1)^{v\cdot\beta} Cor(v \cdot F_K, v \cdot F_{\dot{K}}) \tag{4-44}$$

Due to equation (4-34), equation (4-44) becomes:

$$\Delta_{K,\dot{K}}(K) = \frac{c}{2^n}(-1)^{v\cdot\beta} W_{D_{K+\dot{K}}F}(0,v) \tag{4-45}$$

From equations (4-37), (4-41), (4-43) and (4-45) we can derive (4-46).

$$D(\dot{K}) = c\left|n - 2H(\beta)\right| - \frac{c}{2^{2n}-2^n}\sum_{\alpha\in GF2^n}\left|\sum_{v\in GF2^n, H(v)=1}(-1)^{v\cdot\beta} W_{D_aF}(0,v)\right| \tag{4-46}$$

Thus we come to the definition of the transparency order shown in equation (4-47), it is generalised for (n, m)-functions. This gives an idea of how susceptible an s-box is to DPA attacks. It is the highest value of $D(\dot{K})$ across all possible values of β. This is because the peak will have to be small enough not to be discernable for all values of both the round keys and β.

$$T_F = \max_{\beta\in GF2^m}\left(\left|m - 2H(\beta)\right| - \frac{c}{2^{2n}-2^n}\sum_{\alpha\in GF2^n}\left|\sum_{v\in GF2^n, H(v)=1}(-1)^{v\cdot\beta} W_{D_aF}(0,v)\right|\right) \tag{4-47}$$

The transparency order can vary from 0 to m. If the function F is bent then W_{DaF} will be 0 for all values of v and hence T_F will equal m. Bent functions are not balanced so are never used as cryptographic primitives, but they do resist linear and differential cryptanalysis in an optimal way. More generally if a function satisfies PC (l) at a high level it does not have a good transparency order. If F satisfies PC (l) then D_aF is balanced, and hence W_{DaF} (0, v), for every vector H (a) $\leq l$. This means:

$$\sum_{\alpha\in GF2^n}\left|\sum_{v\in GF2^n, H(v)=1}(-1)^{v\cdot\beta} W_{D_aF}(0,v)\right| = \sum_{\alpha\in GF2^n, H(a)>l}\left|\sum_{v\in GF2^n, H(v)=1}(-1)^{v\cdot\beta} W_{D_aF}(0,v)\right| \tag{4-48}$$

The number of values of a for which the Walsh function of the derivative of F with respect to a is not zero is given by: $2^n - \sum_{j=0}^{l} {}^nC_j$. As W_{DaF} is lower than 2^n then:

$$\sum_{v \in GF2^n, H(v)=1} (-1)^{v.\beta} W_{D_a F}(0, v) \leq m2^n \tag{4-49}$$

So:

$$\sum_{\alpha \in GF2^n} \left| \sum_{v \in GF2^n, H(v)=1} (-1)^{v.\beta} W_{D_a F}(0, v) \right| = m2^n \left(2^n - \sum_{j=0}^{l} {}^n C_j \right) \tag{4-50}$$

Finally:

$$T_F \geq m \left(1 - \frac{2^n - \sum_{j=0}^{l} {}^n C_j}{2^n - 1} \right) \tag{4-51}$$

It is also shown by Carlet in [76] that the inverse function, which is used as the basis of the AES s-box, the Gold functions and the Kasami functions also have large transparency orders. Carlet discovers that in the case of the AES s-box, for which $m = n = 8$, the transparency order is ≥ 7.8, this is close to the maximum, hence the AES s-box has a very poor transparency order.

4.3.6 Signal Processing Techniques

It is possible to combine power analysis with other signal processing techniques in order to improve the performance. Bohy *et al* [77] used principal component analysis (PCA) and *independent component analysis* (ICA), statistical pre-processing techniques more commonly associated with neural networks, to increase the signal to noise ratio and improve the performance of power analysis attacks. PCA searches for linear combinations of variables with the largest variances, when several linear combinations are needed it orders the variances in decreasing importance, thereby allowing the attacker to ignore less relevant measurements. This technique was used to remove noise that was added to the power consumption by masking countermeasures on a smart card. This enabled the Hamming weight to be read from power traces at the point when a PIN being entered was compared to the stored one. They reported results of approximately a 65% chance of being able to recover a PIN from a Microchip PIC 16F84 smart card using this technique with SPA. ICA is a more powerful technique that separates a complex data set into independent sub-parts. The aim was to use it to separate the effects of different parts of the chip from the power

trace, thereby reducing the noise. It was able to unmask the power traces as long as the added noise was independent of the power consumption. It also allowed the recovery of the clock pulse, which would be useful when performing attacks like differential fault analysis.

4.3.7 Other Uses for Power Analysis

The most widely used cryptographic algorithms are open to public review, so any insecurities in algorithms are more likely to be identified and fixed. Some algorithms are kept secret under the assumption that if the details are not publicly known then any insecurities cannot be exploited. This is not necessarily true, as power analysis techniques can be used to reverse engineer algorithms.

Quisquater and Samyde [78] used the analysis of power consumption and electromagnetic emissions to determine the instructions that were being executed on a smart card processor. The processor that was being analysed contained a four stage pipeline; this means that each instruction influences the power trace of the following three clock cycles after it starts. Each instruction gives a different power analysis trace and it is a function of its address in memory, the data that is handled and, if relevant, the address where that data will be stored and the Hamming weight of the instruction is clearly visible. Additional data can also be recovered by measuring the electric field. The concept was shown to be workable by creating a dictionary of the power consumption for various instructions and then recording the power consumption and electric field data for a set of instructions and correlating them with the dictionary entries. A success rate of higher that 87% was reported, it was better for CISC processors that for RISC ones. On the Z80 95% of the software was recovered. Neural networks were then employed in order to automate the process.

When performing DPA the sign of the peaks in the differential power traces are not given any significance, Novak proposed *Sign-Based Differential Power Analysis* (SDPA), which can be used to reverse engineer secret algorithms [79]. The basic method is to perform DPA and to record the signs of the power bias in a SDPA vector. A SDPA vector is a vector with n elements, where n is equal to the number of peaks in the DPA trace. Each element has either a 1, representing a positive bias, or a 0, representing a negative bias. This can then be converted into a SDPA value by calculating the vector dot product of the SDPA vector with a vector with elements

numbered 0 to *m*, each containing the value 2^m. It is difficult to directly interpret the sign data, as there are several different possible explanations, so cross-iteration analysis is used. This is where the SDPA data from several iterations of the algorithm are combined and stored in a SDPA matrix, which can be more conveniently written in the form of a vector containing SDPA values. This makes it easier to interpret how the data relates to each other. New intermediate values can then be identified. These intermediates can then be subjected to SDPA, this new information can then be combined with other methods such as SPA and the algorithm is gradually revealed. Novak successfully applied this attack and reverse engineered an unknown GSM authentication algorithm.

4.3.8 Countermeasures

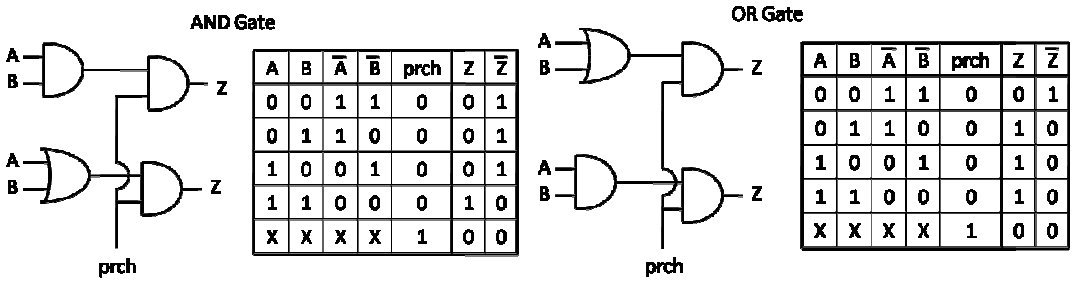
Various countermeasures have been proposed for power analysis attacks with varying degrees of efficacy. These countermeasures can be broadly separated into several groups: balanced logic styles, these seek to avoid leaking information by making the power consumption of transitions the same as non-transitions, masking techniques, these seek to hide details of the internal variables from an attacker and *Dynamic Voltage and Frequency Switching* (DVFS) that confuses the attacker by randomly changing the clock frequency and supply voltage of the chip. Sections 4.3.8.1 – 4.3.8.3 outline these techniques and describe the effectiveness of the countermeasures as well as the cost in terms of area and performance, the performance costs for the various countermeasures are also summarised in section 4.3.8.5. Section 4.3.8.4 discusses countermeasures to higher order DPA.

4.3.8.1 *Balanced Logic*

Section 4.3.8.1.1 discusses the use of *Sense Amplifier Based Logic* (SABL), this is a logic style that has exactly one transition per clock cycle irrespective of the data that is being processed. Tiri *et al* report an AES processor made using this technique where the full key cannot be retrieved even after 1.5 million traces [5]. Section 4.3.8.1.2 discusses Yu and Bree's attempts to prevent DPA by using an asynchronous design that has no clock [80]. This approach offers no DPA resistance unless it also uses a dual rail balanced logic approach. Section 4.3.8.1.3 discusses the effectiveness of the countermeasure and methods for defeating it. Section 4.3.8.1.4 details the effect on performance.

4.3.8.1.1 Sense Amplifier Based Logic

Tiri and Verbauwhede have investigated the possibility of using a different logic style that does not leak information in order to defeat DPA [25, 81-84]. The style they suggested is *Sense Amplifier Based Logic* (SABL). In every cycle SABL charges a capacitance with a constant value and uses this constant amount of charge for every transition, including those where the inputs of a gate do not change in value. This is because SABL is based on the *Dynamic and Differential Logic* (DDL), where there is exactly one switching event irrespective of the input pattern. This is achieved by using DeMorgan's law to create a gate with two halves, one that calculates the result and one that calculates the complement, this assures that there is always one output high and one low. Additionally there is an AND gate on each output and a pre-charge signal that sets the outputs of the gate to 0 for half of each cycle. The design of the AND and OR gates and their truth tables are shown in Figure 4-3.



A	B	$\overline{A}$	$\overline{B}$	prch	Z	$\overline{Z}$
0	0	1	1	0	0	1
0	1	1	0	0	0	1
1	0	0	1	0	0	1
1	1	0	0	0	1	0
X	X	X	X	1	0	0

A	B	$\overline{A}$	$\overline{B}$	prch	Z	$\overline{Z}$
0	0	1	1	0	0	1
0	1	1	0	0	1	0
1	0	0	1	0	1	0
1	1	0	0	0	1	0
X	X	X	X	1	0	0

Figure 4-3: The basic design of DDL AND and OR gates and their respective truth tables.

There is no guarantee that there will only be one switching event per cycle if this approach is used to build DDL versions of compound gates, e.g. XOR gates. Fortunately all logic functions can be made with AND, OR and NOT gates, additionally the inverter is unnecessary in this style as both the result and its complement are calculated, so to invert a signal the Z and $\overline{Z}$connections are switched.

During the pre-charge phase of the cycle the output of both halves of the gate are 0, due to the lack of inverters this means the output of any gates connected to them are 0, so there is no need to pre-charge them. This means that the pre-charge signal propagates as a wave through the block, this reduces the load on the pre-charge signal. This style is called *Wave Dynamic Differential Logic* (WDDL). In WDDL the pre-charge signal is added to the input of the block. This is shown in Figure 4-4, the output of the pre-charge inputs is 0 when the *prch* signal is high. Figure 4-4 also

shows a WDDL flip-flop, during the evaluation phase the registers at the output to the block store the pre-charged 0s and launch the pre-charge wave in successive blocks. It is important to note that a clock speed of twice the data rate is required for this scheme as the pre-charge and evaluation phases happen on different clock cycles.

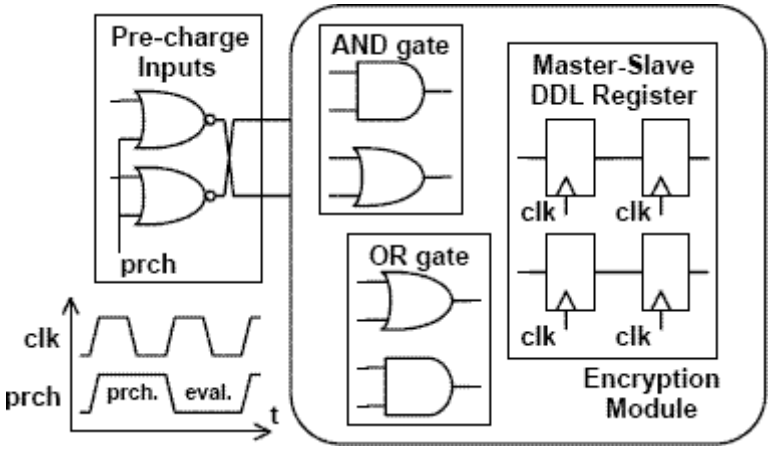


Figure 4-4: A WDDL Flip-Flop with pre-charge inputs. [25]

If there is no inversion in a block of logic then the gates that calculate the result and the ones that calculate the inverted result are on distinct paths, and hence can be separated into two blocks. This can make it easier for the router to match the paths.

In order for this technique to be truly useful it would have to be easily integrated into a design flow. This was achieved by Tiri and Verbauwhede by using the following technique. First the design is synthesised using a subset of a standard cell library using only AND, OR and NOT gates. A script then converts the AND and OR gates into WDDL form and replaces the NOT gates with the appropriate connecting of gates. The placement of the logic proceeds as normal, and the router matches the output lines of the two halves of each gate. A designer does not need any specialised knowledge of the underlying principle of the countermeasure, normal Verilog or VHDL can be used as an input to the automated design flow. This approach was tested by simulation and an ASIC was developed that contained two AES cores, one that used a WDDL approach and one that did not.

4.3.8.1.2 Clock-less AES Design

Yu and Brée proposed a countermeasure that involved creating a completely asynchronous AES chip [80]. This was hoped to provide added security as the clock in a synchronous design guarantees the timing of each operation and so aids the detection of small differences in power consumption. At first they developed an AES

chip using a single-rail asynchronous style. The chip was a 128-bit encryptor, no pipelining was used and computations were performed at the byte level. The s-boxes were implemented as LUTs using dynamic ROM, and RAM was used to store the plaintext, ciphertext and round keys. The asynchronous design language Balsa was used to synthesise the core.

The new design was simulated and the results showed a strong data dependence in the power consumption levels. One of the weaknesses was the ROM; it has two modes, charge and read. When the ROM is in charge mode the output is set to zero, in the read mode the ROM is discharged and the data is loaded onto the output, hence only '1' bits consume current. A new approach was adopted that used dual rail balanced logic instead. This approach was hoped to be more secure against DPA. Everything including the ROM, ROM controller and the RAM had to be made secure using the dual rail approach. This had the unfortunate side-effect of doubling the size, therefore an online key-scheduler was used.

4.3.8.1.3 Efficacy

The clock-less dual rail design developed by Yu and Brée was not directly evaluated for DPA susceptibility, but instead the power consumption was simulated using 500 encryptions with the same key but different plaintexts, and the amount of energy that was consumed was recorded. The mean energy consumed was 764.81 mJ, there were variations of 2.85 mJ around this; the standard deviation was 0.79 mJ. These values were not compared to a single-rail design, so no details of the level of improvement that their technique offered is available. While these variations are small, the standard deviation being just over 0.1% of the mean, they could still be exploited by a determined attacker. The source of the variations was reported to be that some of the dual rail buses were not routed as pair and the logic gates did not have balanced loads.

Tiri and Verbauwhede tested the DPA resistance of the WDDL logic style in simulation [82] and found that if layout parasitics were ignored then it gave perfect security. As stated in section 4.3.8.1.1 Tiri and *et al* also developed an ASIC with both single and dual rail AES cores on it so the gains in security due to the WDDL logic style could be fully evaluated [5]. They recorded 15,000 power traces for the unprotected AES core and found that all 16 key bytes could be determined using

between 320 and 8,168 traces, the average being 2,133. They also recorded 1.5 million power traces for the WDDL AES core. 11 out of the 16 key bytes could be retrieved using between 21,185 and 1,276,186 traces, with an average of 255,391. The remaining five key bytes could not be retrieved, even using the full 1.5 million traces.

As the key can sometimes be retrieved there is still some data leaking from the WDDL implementation. This can be attributed to two main factors: differences in the loading capacitances of two complementary logic gates and differences in the delay time between the input signals [85]. Improving the placement and routing could even out the capacitance and may help with some of the difference in delay, but as some difference in delay is due to the two inputs travelling through a different number of gates it will not be possible to completely eliminate it.

As only dynamic power has been considered when designing the WDDL logic gates it is still susceptible to Leakage-Based DPA. Lin and Burleson used normal DPA and the approach described in section 4.3.2.1 to attack an implementation of DES protected by WDDL simulated in SPICE. The key could be retrieved using 5000 traces with regular DPA, this fell to 2000 with LDPA [66].

4.3.8.1.4 Efficiency

In order to balance the logic it needs to be replicated so that there is always one transition, this requires doubling the amount of logic and hence the area of the design. Additionally, balanced flip-flops require four normal ones, so the overall area of a design will more than double.

The ASIC that Tiri *et al* developed containing both single and dual rail AES cores shows an increase in area of a factor of 3, going from 0.79 to 2.45 mm^2, the maximum clock speed falls by nearly a factor of 4, going from 330 to 85.5 MHz and the power consumption rises from 0.054 to 0.200 W, again nearly a factor of 4. Clearly this is an expensive DPA countermeasure.

4.3.8.1.5 Conclusion

Dual rail designs can significantly reduce the information leakage from a crypto device. In ideal conditions, when parasitics and path length are ignored they give perfect security in simulation. Clearly these are unrealistic assumptions and when these are included information is leaked. There are more complex place-and-route

algorithms that can match them in the two paths and reduce the correlation between power consumption and data, but there are some sources of the correlation and hence leakage that are currently unavoidable, inputs to a logic block travelling through a different number of gates for example. Even though some unrealistic assumptions were made for some simulations of dual rail designs, these criticisms clearly cannot be made against the WDDL ASIC made by Tiri and Verbauwehede. The ASIC clearly demonstrated that WDDL is capable of significantly reducing the amount of information leaked through power consumption. Dual rail designs do come at a high cost in terms of performance, for the WDDL ASIC the area increased by a factor of 4, the speed fell by a factor of 4 and the power consumption increased by a factor of 4.

4.3.8.2 *Hiding Intermediate Values*

Section 4.3.8.2.2 discusses the use of masking, this is where the plaintext is masked with a random value and encrypted; the mask is removed after the encryption. This hides all the intermediate values from the attacker. Section 4.3.8.2.1 discusses the duplication method, which splits the intermediate values into a number of other variables using a secret splitting scheme. Section 4.3.8.2.3 discusses the effectiveness of the countermeasure and methods for defeating it. Section 4.3.8.2.4 details the effect on performance.

4.3.8.2.1 Duplication Method

A method proposed by Goubin and Patarin [86] and again later by Chari *et al* [10] called the duplication method involves replacing each intermediate value that depends on the input with k variables that form into a secret sharing scheme. An example of the secret sharing scheme that could be used is if the k variables were XORed together to form the actual intermediate value. Computations can then be performed securely on the shares using a modified algorithm and then recombining the data at the end.

In general operations in cryptographic algorithms will fall into one of five categories:

1. Permutation.
2. Expansion.
3. XOR with another intermediate variable.
4. XOR with key data.

5. Non-linear transform.

Permutation and expansion operations simply need to be performed on all variables in the secret share, the relationship between them before the operation will still be correct after it. With the two types of XOR operation, if it is between two variables dependent on the input then the corresponding section of each variable must be XORed together, if it is with key data then the key data must be XORed with each section of the intermediate variable. The non-linear transforms are slightly more complicated to implement, *k* different s-boxes are required, each with all of the sections of the secret share as an input and one as an output. Of these *k* s-boxes, *k*-1 of them implement randomly chosen secret transformations and the remaining one implements a transform that when combined with all the others will give the value that would have been given if they were combined before transformation. As tables are used for the substitutions there is no need to recombine the different sections of the secret share and hence it remains secure against DPA.

4.3.8.2.2 Masking

Masking involves ensuring the attacker cannot predict any full registers in the system without making run-specific assumptions that are independent of the inputs to the system. This is achieved by applying a reversible random mask to the plaintext data before encryption with a modified algorithm. This makes exploiting data from several encryptions impossible as it would require guessing the correct mask for each run, increasing the number of traces decreases the probability of this. The difficulty with this technique is that if a mask is added in a linear way it will be difficult to remove after the non-linear section of the algorithm, in AES the Sub Bytes operation.

Several different masking techniques have been designed for AES, the three main ones were developed by Akkar and Giraud [87], Oswald *et al* [4, 7, 88] and Trichina and Seta [8]. Akkar describes a masking technique for both AES and DES although only the AES method will be discussed here. It involves adding a mask to the plaintext, removing it before the Sub Bytes operation and replacing it with a multiplicative mask. After the byte inversion in Sub Byte the multiplicative mask is replaced with the original additive mask. Trichina's method uses a similar technique to that of Akkar's except that it re-uses the additive mask as the multiplicative mask, this requires fewer operations as it has to calculate fewer masks. The other difference

is that a new mask is generated every round. The third approach, by Oswald *et al,* adds a mask at the start and does not remove for the non-linear sections of the algorithm, but converts the calculated $(data + mask)^{-1}$ to the wanted $(data^{-1} + mask)$ by calculating a correction in parallel.

4.3.8.2.3 Efficacy

The duplication method does not offer complete security against DPA, it is possible to perform k^{th} order DPA on it. Generally the complexity of HODPA increases exponentially with order, as the traces must be combined with the correct time offset between them. This means DPA would probably become infeasible with a relatively small *k*. If the system is not carefully designed however, then it may be possible to spot the positions on the power trace where the data is being handled and hence reduce the complexity to something manageable [89].

In order to evaluate the security gains of the various masking techniques Pramstaller *et al* made a processor utilising the three types as described in section 4.3.8.2.2, they discuss this in [23, 90]. No attempts to perform DPA on these implementations was made, the standard DPA algorithm is not applicable to a masked implementation. This does not guarantee security however; both the Akkar and the Trichina approaches are vulnerable to a "zero value attack". This is where the partial data and the partial key have the same value after the additive mask is removed so the value will be zero and the multiplicative mask will have no effect [91]. Also it was shown in [92] that the Trichina method can be defeated by regular DPA. Akkar reports in [87] that HODPA, albeit with considerable effort, can defeat the masking countermeasure. Mangard *et al* implemented an AES ASIC using the Oswald approach, the Akkar approach and an unmasked implementation [93]. Despite the fact that these masking techniques were provably secure against first order DPA the implementations could still be attacked using results from simulations to make predictions about the outputs of logic gates rather than registers. For Akkar's technique this required 130,000 traces. For Oswald's technique only 30,000 were required, this was of a similar order to the 25,000 traces required for the unmasked s-box.

4.3.8.2.4 Efficiency

The effect that the duplication method has on the performance of the design depends on the value of *k*. For each additional secret share variable an additional data path needs to be added. Additionally, as well as needing more s-boxes they are all bigger, the size increasing to the power of *k*. The size of the design will have to increase by a factor greater than *k*. The duplication method significantly increases the amount of memory required for the s-boxes so it is not appropriate for smart card implementations without modification. In [86] Goubin and Patarin suggest ways to reduce the memory requirements so as to be able to fit a duplication protected DES implementation with a *k* of 2 on a smart card. If the same random transformation is used for all of the 8 DES s-boxes then the number of s-boxes that need to be stored in memory is 9 rather than 16. These s-boxes can be made smaller by combining the parts of the secret share in a secure way, by doing it inside a bijective masking function so the actual value never appears in registers. The two post-s-box intermediate values are: some randomly chosen secret transform of the securely combined value, and, that value XORed with what the output of the s-box would be in an unmodified implementation. The unmodified s-box output can be calculated securely from the altered combined value from a table that has been rearranged.

Pramstaller *et al* implemented an ASIC with AES s-boxes using the three masking techniques described in section 4.3.8.2.2 [9, 23], they used a 0.25 μm process. The effect of the countermeasures on the area and the critical path are shown in Table 4-6. Akkar's implementation is the largest, and Trichina's is the smallest, Oswald's is the slowest. Compared to the smallest AES s-box the area increases by between a factor of just under 3 to over 4, and the critical path increases by a factor of between approximately 1.5 and 2. While compared to the fastest AES s-box the area increases by a factor of between 1 and 2 and the critical path by between approximately 4 and 6. The final column of Table 4-6 gives the number of random bits per data bit required by the algorithm. Oswald's masking algorithm requires a 128-bit mask for each 128-bit input block. Akkar's masking algorithm required both a multiplicative and additive mask and hence required 256 random bits for each input block. Trichina's masking algorithm reuses the additive mask as a multiplicative mask but requires one for each round so requires 1,280 random bits for each input block. This will put additional strain on the efficiency of the algorithm as the random numbers have to be generated by additional circuitry and potentially in additional

time. Pramstaller *et al* also developed a 128-bit implementation of AES Oswald's masking method, it was designed as a high throughput chip and they compared it to Fastcore [94]. They reported an similar area, Fastcore had 45,325 gates, their design had 42,408, this did not include the circuitry required to generate the random masks and the Fastcore datapath includes some additional functionality [9]. The throughput fell from 2.12 Gb/s for Fastcore to 1.15Gb/s for Pramstaller *et al*'s design. Both chips used a 0.25 μm CMOS process.

S-box Implementation	Area (mm^2)	Critical Path (ns)	Random Bits / Data Bit
AES	0.0075 – 0.0125	4 – 5.8	0
AES LUT	0.015 – 0.037	1 – 3	0
Oswald	0.025 – 0.033	8.3 – 14.4	1
Akkar	0.034 – 0.054	6.5 – 12.2	2
Trichina	0.020 – 0.035	6 – 9.2	10

Table 4-6: A table comparing the area, speed and random bit requirements for masked and unmasked implementations of the AES s-box [23].

4.3.8.2.5 Conclusion

Clearly masking techniques are not a solution to the problem of DPA, the costs on performance are significant, increasing the size of an s-box by a factor of between 3 and 6 [9, 23] and in an full implementation of AES decreasing the speed by a factor of 2 [94]. That doesn't even include the penalties for generating the massive amount of random bits required to make the masks. Much more significant that the cost to performance is the inability of any of the proposed masking techniques to actually protect a system from DPA, the Trichina technique is vulnerable to regular DPA [92], and the Akkar and Oswald techniques are vulnerable to DPA against logic gates rather than registers, using simulations to predict the values of the logic gates [93]. This pretty much rules masking out as an effective DPA countermeasure.

The duplication method can protect implementations against DPA but it can be defeated with HODPA. The order of HODPA to which it is vulnerable is determined by the number of duplications that are used to protect it. HODPA is computationally expensive as it requires the attacker to combine the power traces in an iterative way to

determine the correct time offset. One possible weakness of the duplication method is the random transformations that are used to make the s-boxes need to be kept secret, if an attacker was able to determine what they were then the implementation may be vulnerable.

4.3.8.3 *Dynamic Voltage and Frequency Switching*

Dynamic Voltage and Frequency Switching (DVFS) was originally proposed as a technique to reduce power consumption. Yang *et al* realised that is would also frustrate an attacker who was trying to perform DPA on a cryptosystem as they would typically assume that the device is operating at a constant frequency and hence take power samples at constant intervals [11].

When DVFS is used as a DPA countermeasure it is composed of three parts, the processor core, the DVFS feedback loop and the DVFS scheduler. The DVFS scheduler randomly generates a voltage or a frequency value, the feedback loop then implements the frequency and voltage using a phase locked loop and a ring oscillator and supplies it to the processor core.

4.3.8.3.1 Efficacy

Yang *et al* [11] created a simulated DES implementation with DVFS but did not perform actual DPA on it. Instead, to measure the effectiveness of their countermeasure they simulated 1000 of encryptions, collected statistics about the variation in the power and timing, and defined two performance metrics, *Power Traces Entropy* (PTE) and *Time Trace Entropy* (TTE). These represent uncertainty in the power and consumption and clock period traces. They found that in their design with DVFS the PTE was 7.5% higher than their design without. The TTE was ∞% higher as there wasn't any uncertainty in the timing in designs without DVFS.

Baddam and Zwolinski [95] attempted to perform DPA on a simulation of a cryptosystem using this technique and discovered that they could not retrieve the key after 10,000 traces. As any circuit with a clock frequency is necessarily sequential, lots of values will change in the circuit on the rising edge of the clock, this will create a detectible spike in the power consumption, which the attacker is already measuring. This information can be used to determine the altered frequency, this in turn give information about the new supply voltage. Applying this new data to their DPA technique Baddam and Zwolinski were able to retrieve the key from a DVFS

protected cryptosystem. As an improvement to the technique they suggested only modifying supply voltage and keeping the frequency constant so as not to give the attacker the tools to defeat the countermeasure. When tested this reduced the correlation with power consumption by a factor of 5, although DPA could be performed on a single AES s-box using 2,500 traces it could not be performed on a full implementation of AES with 10,000 traces.

4.3.8.3.2 Efficiency

The area overheads for DVFS are not particularly large, as adding the countermeasure simply involves including the DVFS scheduler and feedback loop. In [11] it was assumed that the underlying hardware was already available in the design essentially reducing the area cost to nothing. The accuracy of that assumption depends on the exact nature of the device in question and it will not always be true. Changing the frequency and the voltage affects the amount of power and time required to perform encryption. When Yang *et al* implemented DES using DVFS they reported a speed overhead of 16% but the amount of power used fell by 27% [11].

4.3.8.3.3 Conclusion

In terms of the performance cost to an implementation that a countermeasure incurs DVFS is cheap. The size of the design will not increase very much, especially if the chip already has a phase-locked loop and ring oscillator. The speed decrease is a moderate 16% and the power consumption was even reduced by 27%.

If the operating frequency is altered as well as the voltage then it is possible to use the large increase in power consumption that accompanies the rising edge of a clock pulse to deduce the new clock frequency and hence the supply voltage and defeat the countermeasure [11]. If only the power consumption is altered then this is not possible and the countermeasure can achieve a factor of five reduction in the correlation with power consumption [95], making it infeasible to attack a simulation of full AES with 10,000 traces.

4.3.8.4 *High Order DPA Countermeasures*

Designing *ad hoc* countermeasures for DPA does not assure security against HODPA. It has been shown that masking techniques are not secure against HODPA [92]. The Duplication method, using k shares, does not provide security against k^{th}

order DPA attacks. However, while the complexity of the implementation increases with order *k*, the complexity of the attack increases exponentially with *k*, so an attack would quickly become infeasible.

A modification to the masking technique was suggested by Chang and Kim that would make it secure against 2nd order DPA [96]. It involves generating two masks and selecting one at random and applying it to the data. This provides security as the attacker is not able to determine the point at which the mask is loaded.

Another masking technique that is not vulnerable to HODPA was proposed by Goubin and Akkar [97] and applied to DES. A 32-bit number is generated randomly and this is used to create 2 new s-boxes, one that masks the value and one that unmasks it. The intermediate values in DES have variable vulnerability to DPA, that is to say that information about some of the rounds does not gives more information about the key than others. The middle 2 two rounds are not vulnerable, at this point the data is unmasked and a new set of secure s-boxes is used. If only one mask was used then it would be susceptible to HODPA in the same way that other masking techniques are.

4.3.8.5 *Summary of Power Analysis Countermeasures*

Countermeasure	Penalty			Effectiveness
	Speed	Area	Power	
Balanced logic				
WDDL	4	3	4	11 out of 16 key bytes identified with average of 255,391 traces, others not identified with 1.5 million traces.
Masking				
Oswald [23]	2.1	2.5	-	Susceptible to DPA targeting logic gates with back annotated netlist. [93].

Trichina [23]	1.5	2.8	-	Susceptible to DPA [92].
Akkar [23]	1.7	4.2	-	Susceptible to DPA targeting logic gates with back annotated netlist. [93].
Duplication	-	$> k$	-	Susceptible to HODPA [22].
DVFS				
Frequency and supply voltage	1.2	-	.73	Can be defeated using power surges to find new rising edges of clock pulse [95].
Supply Voltage	-	-	-	Correlation strength decreased by a factor of 5 [95].

Table 4-7: Summary of DPA countermeasures.

4.3.8.6 *Conclusion*

As discussed in section 4.3.8.5 adding hardware countermeasures to DPA is expensive in terms of area and time requirements, increasing them by a factor of 1.2 to 4 and 1.8 to 4 respectively.

The effectiveness of the countermeasures is often debatable, even if it has been demonstrated that they work. Some masking techniques that were proposed were shown to be susceptible to higher order DPA attacks [87] and one was even vulnerable to first order DPA [92]. Even techniques that were shown to be theoretically provably secure [6, 7, 46], was susceptible to DPA using predictions based on simulations and a back-annotated netlist [93].

Most other countermeasures were generally shown to be secure in simulation with unrealistic assumptions about parasitics. Even if the countermeasures are implemented it is still difficult to show that they are definitely secure by attempting DPA on them. However many traces are recorded in the experiments it is still possible that if more were taken enough information would leak to enable an attacker to retrieve the key. It is only possible to show that countermeasures are not secure, or are effective up to a certain limit. In order to assure security this limit has to be greater than the number of traces realistically available to the attacker. This is still not a

definite figure, and is largely dependent on the application for which the encryption is used. Hwang *et al* collected 1.5 million traces and could not retrieve the key from their WDDL AES ASIC [98], they claimed that this demonstrates that their method is secure. 1.5 million traces is equivalent to the encryption of 12 MB of data.

4.4 Conclusion

Cryptography is a constant battle between cryptanalysts and code makers. Algorithms are developed, weaknesses are found and techniques to overcome these vulnerabilities are discovered and included in the next generation of algorithms. As processing power becomes cheaper algorithms have to become more complex, with longer keys to withstand the ever increasing brute force attacks. The current standard is AES which has no known mathematical attacks and with key lengths of 128, 192 and 256 bits cannot be feasibly brute forced with current technology. Even the most modern algorithms however fall to side channels, until an effective countermeasure is developed the cryptanalysts have the upper hand.

There are no truly successful techniques to protect AES against DPA. Hardware countermeasures are expensive in terms of area and speed and cannot be guaranteed to work. Masking techniques have been shown to vulnerable to a variety of DPA based attacks such as targeting logic gates [93] and the duplication method is vulnerable to high order attacks [22]. Clock frequency based countermeasures can be compensated for and defeated [95]. Balanced logic styles can protect implementations up to a point, but any data dependence in power consumption, no matter how small, can be exploited by an attacker if they have access to enough power traces, and the logic can never be perfectly balanced.

The countermeasures that have partial success, balanced logic and randomly varying the supply voltage, both reduce the effectiveness of DPA. This does not prevent it, but does increase the number of traces required to successfully perform an attack. This leads to the question: how many traces are required for an attack before it is considered secure against DPA? There is no general answer to this, it is determined by the specific use of the system in question, if it is not feasible to get enough power traces to retrieve the key before the key is changed then it is essentially secure. If the key is never changed then the attacker has a theoretically infinite number of power traces available to them and even the slightest data dependence in the power

consumption will eventually betray the key. Tiri *et al* tested their WDDL system with 1.5 million traces and found that they could not retrieve the entire key using DPA [5] but DES is considered vulnerable to differential cryptanalysis even though it requires 2^{47} chosen plaintexts. Additionally, even retrieving some key bits weakens the cipher. In the WDDL example 11 out of 16 key bytes were successfully determined, this just leaves 40 key bits unknown. That is well within the reach of an exhaustive search with current levels of available computing power. It is important not to forget the words of Robert Morris at Crypto 95:

"Never underestimate the time, expense, and effort an opponent will expend to break a code."[99]

Chapter 5 Recording and Analysing Power Data and Benchmark DPA Results

5.1 Introduction

In order to investigate potential countermeasures to differential power analysis it must first be possible to perform DPA so the efficacy of any modification can be determined. This requires both a design of a suitable cryptographic algorithm, in this case AES, the implementation of which is described in section 5.2, and a system to extract power consumption measurements. A system was developed that used an oscilloscope to measure the power consumption of an FPGA while performing encryption and is described in section 5.3.2. The data was then analysed using a program that is described in section 5.3.3. Also DPA was performed in simulation in two different ways. Firstly the power consumption was estimated using transitions in registers in a VHDL simulation as, this is described in section 5.3.1.1. Also, a model of DPA was made in Matlab that can be used to quickly perform lots of experiments, this is useful for performing Monte Carlo simulations and is described in section 5.3.1.2.

5.2 AES Core

This section describes the AES core that was implemented in order to have a platform to test the susceptibility and effectiveness of countermeasures to DPA.

Another advantage of developing a new AES implementation was that a greater understanding of the algorithm and issues regarding implementation was gained.

5.2.1 Modules

A highly modular design style was used to implement AES. This simplified the addition of optimisations and the creation of a variety of architectures.

5.2.1.1 *Sub Bytes*

For the Sub Bytes operation two different s-boxes have been produced, one that uses the LUT approach and one that converts the values from GF (2^8) to GF (2^4), also versions of these were made for pure encryptors, for further details see section 3.6.9.2. The designs were synthesised and details are given Table 5-1.

S-box	Slices	Delay ns
LUT Full AES	139	9.41
LUT Encryptor	68	8.44
GF (2^4) Full AES	58	20.33
GF (2^4) Encryptor	46	17.45

Table 5-1: Details of the various s-box implementations.

As the Shift Rows operation can simply by implemented as routing it was combined with Sub Bytes in the top level of the module, with the substitution of each input going to a different output.

5.2.1.2 *Mix Columns*

Originally the multiplication for the mix columns was performed by a series of generic GF (2^8) multiplication modules. One of the operands in each multiplication is constant, so this was then changed to a series of custom built constant value multipliers, the equations for which are given in Table 3-4. There is a degree of sharing of values in the equations so there is a further optimisation that can take place, the 4^{th} bit of the 03 multiplier is the 4^{th} bit of the 02 multiplier and x_4. When a purely encrypting implementation of AES was made the custom multiplier was changed so it only calculated the multiplications needed for encryption. The designs were

synthesised and details are given Table 5-2. The custom multipliers outperform the generic ones in both speed and area.

Multiplier	Slices	Delay ns
Generic	724	13.13
Custom	600	11.97
Custom enc	208	5.65

Table 5-2: Details of the various Mix Columns implementations.

5.2.1.3 *Key Scheduler*

Several different key schedulers were designed that could calculate an entire round key in 1 clock cycle. They were 128-bit online and offline schedulers and a full AES offline key scheduler. Online key schedules generate the round keys as they are required whereas offline ones pre-calculate them and store them for future use. The designs were synthesised and details are given Table 5-3.

Key Scheduler	Slices	DFFs	Delay
128-bit Online	428	128	31.03 ns
128-bit Offline	1,383	1,285	769 ns
Full AES Offline	7,688	1,824	84.78 ns

Table 5-3: Details of the various Key Scheduler implementations.

5.2.2 Architectures

Several different architectures were implemented, a 128-bit encryptor with online and offline key schedulers, a 128-bit encryptor / decryptor with online and offline key schedulers, a full AES encryptor / decryptor with an offline key scheduler and a fully pipelined 128-bit encryptor with online and offline key schedulers. The designs were synthesised for a Xilinx XCV100E with a speed grade of -6.

The area, speed and throughput in bits per second are given in Table 5-4. For implementations that have an offline key schedule additional clock cycles were required to pre-calculate the expanded key so the overall cycles per result and hence

the throughput would be dependent on how many blocks were encrypted with each key. For the implementation of full AES with three different possible key lengths, the number of rounds and the setup time of the expanded key is determined by the key length. It can be seen from Table 5-3 that although an online key scheduler is smaller for one round this is because the overhead in creating an offline version is very large, when the pipelined architectures were created the offline key schedule approach used nearly 2,000 less slices. The complexity involved in creating an implementation that supports multiple key lengths is so great that there is a significant increase in area and reduction in clock speed.

Implementation	Clock (MHz)	Cycles / Result	Throughput (Mb/s)	Slices	DFF
128-bit Enc online	33.4	10	427	2,446	516
128-bit Enc offline	25.2	10	323	3,099	1,702
128-bit Enc/Dec off.	28.1	10	360	4,364	1,575
Full AES Enc/Dec off	12.5	10–14	114-160	10,003	2,085
128-Bit Enc piped on.	43.9	1	5,620	11,709	2,309
128-Bit Enc piped off.	28.5	1	3,650	9,991	3,624

Table 5-4: The performance results from the various AES implementations.

5.3 Performing a Correlation Attack

In order to gain any real insight into power analysis attacks and their countermeasures such an attack must be performed. The ultimate aim of this is to develop a system where the resistance to power analysis attacks can be measured. This section describes the various methods that have been used to perform a correlation attack on AES and the tools that were developed in order to facilitate them.

5.3.1 Simulation

Before an attack was performed on a physical system it was first done in simulation. Section 5.3.1.1 describes a correlation attack on Modelsim simulation of a

VHDL design of an AES chip. Section 5.3.1.2 discusses the use of Matlab to simulate a realistic but simplified model of a device being subjected to power analysis. This allows investigation into properties of the attack that would otherwise take a prohibitively long time.

5.3.1.1 *FPGA Power Estimation*

As shown in [60] the number of bit transitions inside the registers of an FPGA gives a reasonable estimation of the power consumption at that time. For this reason a program was written that could accept an FPGA design file and use it to produce a file containing the number of bit changes within all registers in the design on each successive clock cycle. Details of the program are given below.

The program parses post-synthesis VHDL files and extracts the names of the registers in the design. The program then writes a test bench containing the key to be extracted and a list of plaintexts. Additionally a Modelsim script file is written that loads the design and a test bench, runs the simulation and records the values in the registers at each delta time into a file. This file is then read, and the number of transitions in a given clock cycle is counted. This information is used to perform DPA on the design using the method described in section 4.3.2.2, to extract the key that was specified in the test bench. In this example 742 traces were required to extract all 16 bytes of the key.

The first byte of the key had the decimal value 43. In Figure 5-1 the correlation between the consumption matrix and the prediction matrix for the first byte of the key is shown for all 256 possible values of the key, the value with the highest correlation is 43, this means that the correct value for the first key byte can be correctly identified.

In Figure 5-1 it can be seen that there are a series of distinct levels that the values of the correlation take. This is due to the effect discussed in section 4.3.5.1. Each time the Hamming distance between the key guess and the correct key is increased, the correlation falls by a fraction equal to the number of bits in the key guess, in this case 8. This can be seen in Figure 5-1 as the correlation when the key guess is 42 is approximately the same as when it is 47, both having a Hamming distance of 1 away from 43. When half the bits are incorrect there is a correlation of

approximately 0, and when all bits are incorrect the correlation is negatively correlated by the same amount as the largest peak.

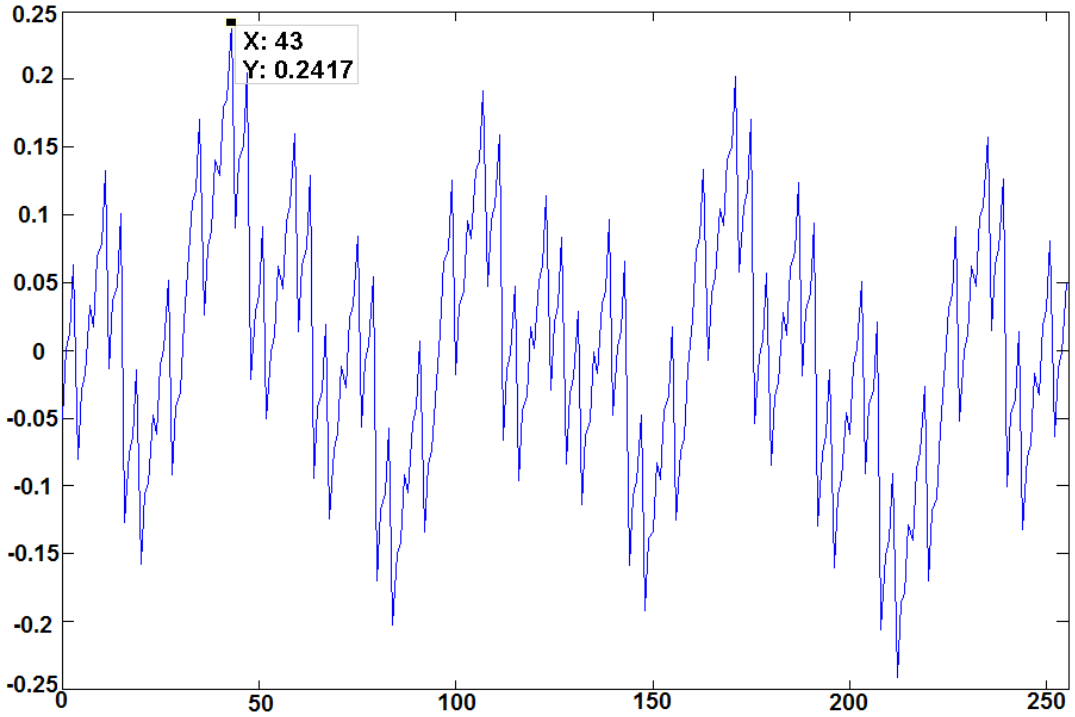


Figure 5-1: Graph showing the correlation of the 256 key guesses for a correlation attack on the power estimation of an AES FPGA with 1,000 traces.

5.3.1.2 *Matlab Simulations of the Consumption Model*

The simulated attack described in section 5.3.1.1 takes a significant amount of time. The majority of this is taken up by the Modelsim simulation, as to be sure that enough register transition data was collected 4,000 plaintexts were used, this took over 2.5 hours on a 3 GHz Pentium 4. Using Matlab it is possible to simulate a correlation attack on AES much faster and so investigate a wider variety of properties of the attack, such as the affect of the SNR and number of traces on the results of the correlation.

In the AES design that was attacked using a Modelsim simulation in section 5.3.1.1 there were 516 registers, 128 are used for storing the data relevant to the attack, the rest are not used at all during the targeted clock cycle of the encryption. This may seem to imply that there is no noise in the measurements, but this is not true. All of the bytes are calculated in parallel but each one is targeted individually and the data is independent so the data from one byte appears as noise when attacking another. This means that the signal-to-noise ratio of this system is 0.25. The Matlab

model to simulate an attack randomly generates 16 1-byte integers for the plaintext and XORs them with a 16 byte key and then sums the Hamming weight of each number. This value is entered into the consumption matrix. The prediction matrix is the Hamming weight of all 256 possible key values XORed with the randomly generated plaintext value of the target byte.

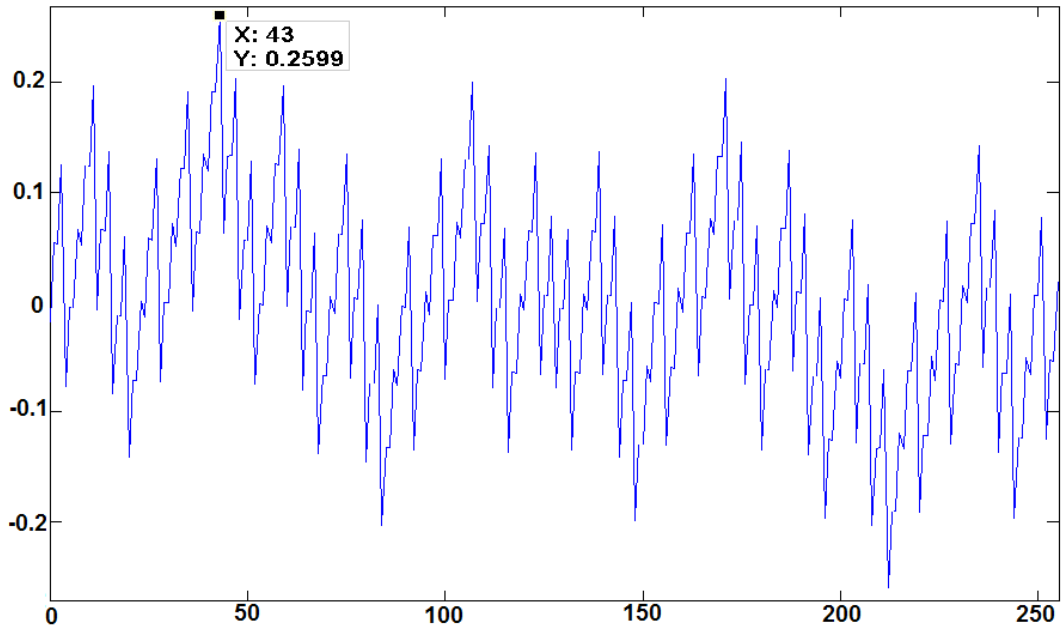


Figure 5-2: Graph showing the correlation of the 256 key guesses for the Matlab model of a correlation attack on AES.

The first byte of the key had the decimal value 43. In Figure 5-2 the correlation between the consumption matrix and the prediction matrix for the first byte of the key is shown for all 256 possible values of the key, the value with the highest correlation is 43, this means that the correct value for the first key byte can be correctly identified.

The signal to noise ratio can be improved by combining data from two key bytes. This does increase the size of the key-space that must be exhaustively searched from 2^8 to 2^{16}. The value of the 2-byte section of the key that was being targeted was 0x2B7E or 11,134 in decimal notation. In Figure 5-3 the correlation between the consumption matrix and the prediction matrix for the first byte of the key is shown for all 65,536 possible values of the key, the value with the highest correlation is 11,134, this means that the correct value for the first two key bytes has been correctly identified. As stated in section 4.3.5.1 the correlation of the correct key choice is related to the signal to noise ratio of the system, as this has been increased from

$\sqrt{1/16}$ to $\sqrt{1/8}$ the value of the maximum correlation as calculated by equation (4-15) becomes 1/3. Like Figure 5-1, Figure 5-3 also has a regular pattern in the values of the correlation, with key guesses that have the same Hamming distance from the correct value having the same correlation. The only difference is that in this example there are 2 bytes, and so 17 different possible values for the Hamming distance between the correct and incorrect values.

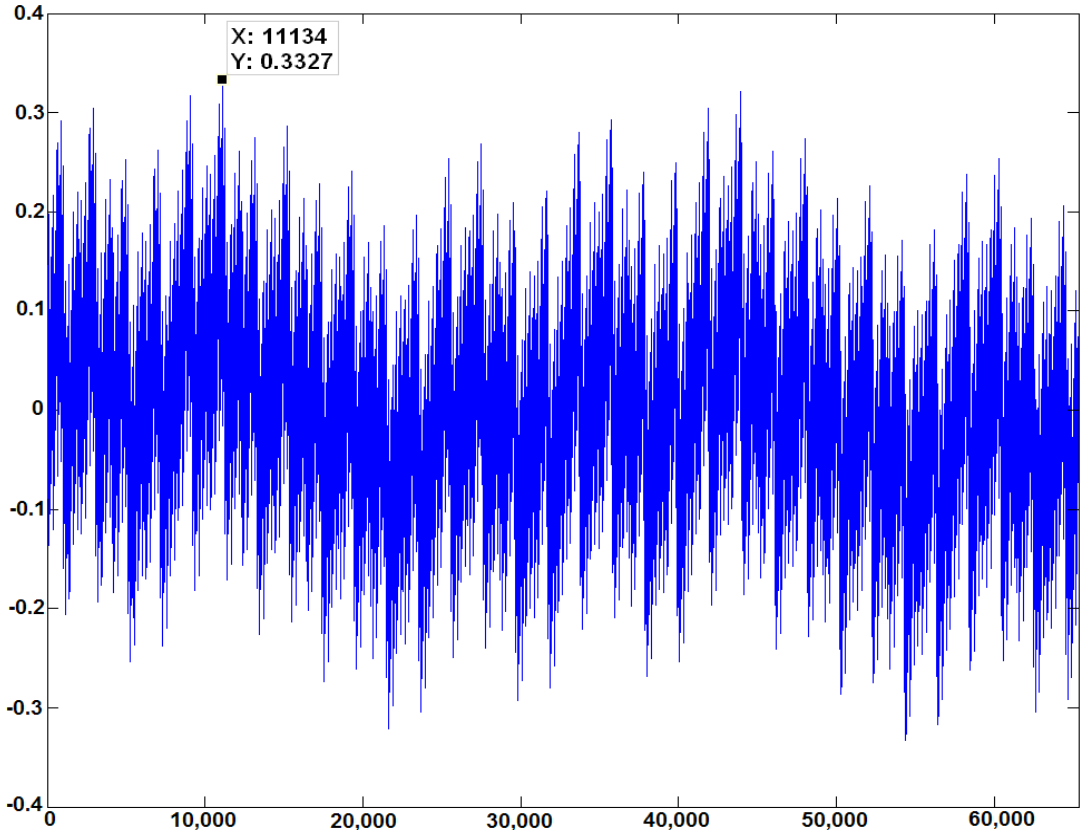


Figure 5-3: Graph showing the correlation of the 65,536 key guesses for the Matlab model of a correlation attack on 2 bytes of AES (2B 7E) with 1,000 traces.

5.3.2 Performing a Correlation Attack on an FPGA

An AES core was combined with an LFSR to provide the plaintexts and loaded onto a Xilinx XCV1000E FPGA. The FPGA was put into a Xilinx BG560 prototyping board. The board only contains wiring for the JTAG, sockets for oscillators and some LEDs. The power for the internal logic of the FPGA is supplied via separate power supply jacks, this means that the power consumed by other things on the board does not interfere with the power consumed by the chip itself, this will reduce the noise for power analysis attacks. The power consumption data was

captured using an Agilent Technologies 4 channel mixed signal oscilloscope with a maximum sample rate of 1GHz (MSO6104A). The power consumption of the FPGA can be deduced by measuring the current drawn from the power supply, the oscilloscope only measures voltage. A 0.5 Ω resistor was connected in serial with the FPGA and the voltage across it was measured by connecting an oscilloscope probe either side of it and subtracting one value from the other. The FPGA generated a *Doing* signal, one pin went high when an encryption was being performed in order to trigger the oscilloscope. In addition to the power consumption data the oscilloscope also captures the *Doing* signal and the clock pulse to aid the synchronisation of the power consumption traces. The output from the oscilloscope is an array of 1,000 floating point numbers signifying the values displayed on the screen.

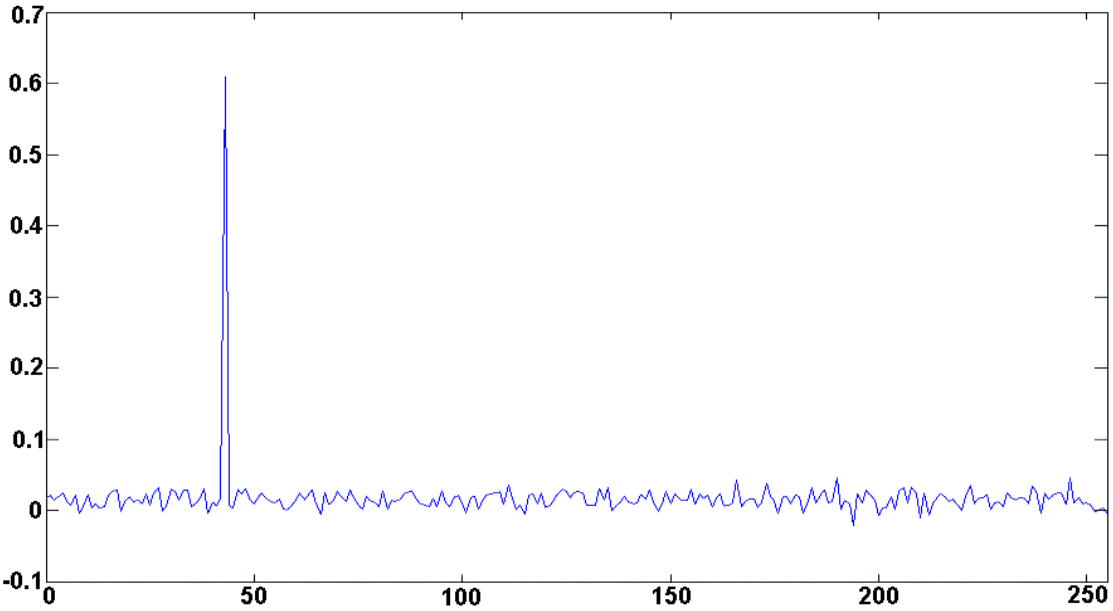


Figure 5-4: Correlation for all possible key values for an attack on a single AES s-box on an FPGA with 10,000 traces.

To test the setup, an attack was performed on a single AES s-box. A random 8-bit plaintext was generated using a 128-bit LFSR, this was XORed with a constant value (0x2B) to represent the key and this was fed into the s-box and the output was stored in a register. A graph showing the correlation of all possible values of the constant key for 10,000 traces is shown in Figure 5-4, there is a clear peak at the decimal value 43, showing the correct value of the constant that was XORed to the LFSR output. Figure 5-4 looks different to the previous graphs of the correlation of all possible key values, there is not the distinct set of levels for the correlation of

incorrect values, this is because the predictions are made after the s-box. This is discussed in more detail in section 5.4.

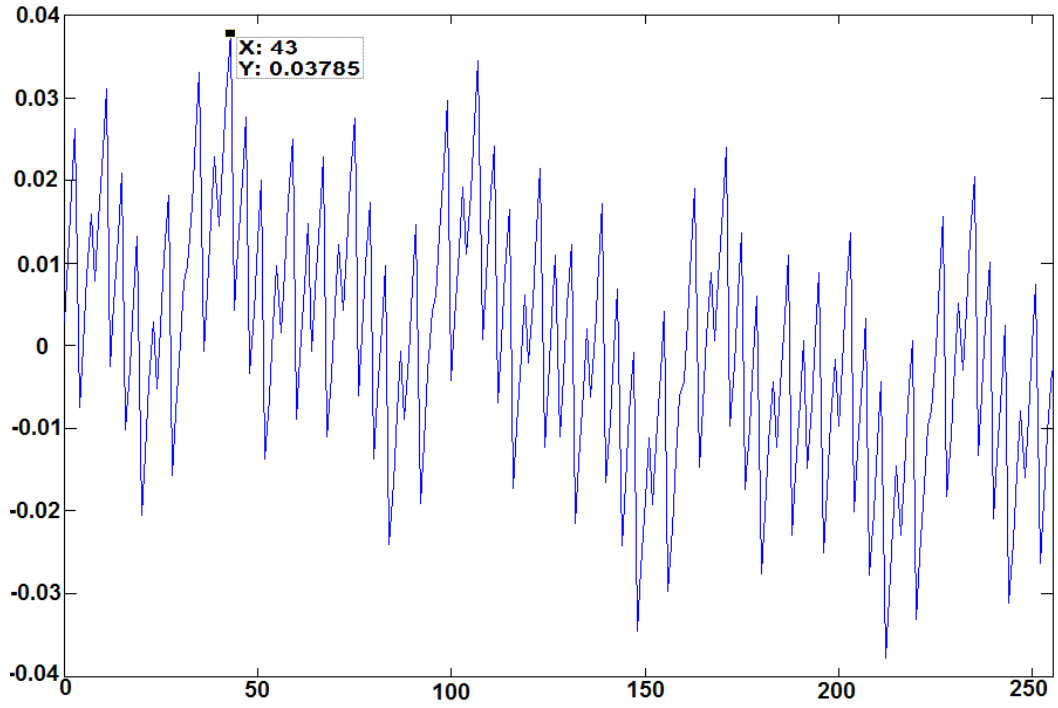


Figure 5-5: The correlation for all possible key values after 30,000 traces while attacking the first byte of the first sub-key of AES before the s-box.

After the correct function of the test-bed was verified a correlation attack was performed on an AES core. The correlation of all possible key values of the first byte of the first sub-key after 30,000 traces is shown in Figure 5-5, again the correct value is 43 and this is the largest peak in the graph. Even though the correct answer is still clearly visible in the graph the actual value for the correlation is much lower in this attack, at 0.03785, compared to the simulated attack, at 0.2599. This is because there is a lot more noise this system as the Modelsim simulation considers only register transitions whereas in the real system there is noise from all parts of the circuit.

5.3.3 DPA Software

As noted in section 4.3.2.2 a correlation attack is a three stage process involving predicting the values in registers based on the plaintext and the key, generating and capturing the power consumption information and performing the correlation. These can be grouped into two more general tasks of capturing power data and then processing it. Power consumption data from the FPGA was captured using the Agilent MSO6104A oscilloscope; in order to automate this, a program was written to control

it. The program has two main tasks, initialising the settings on the oscilloscope and transferring data from the oscilloscope to the PC. The settings for the capture, such as the number of traces that are going to be performed and the number of clock cycles of the traces that are to be captured, are entered into the GUI, shown in Figure 5-6, the program then calculates the appropriate time-base settings and sets up the relevant channels on the oscilloscope for capture. The program also sends requests for data to the oscilloscope. When the FPGA is performing an encryption a "Doing" signal is set high, the oscilloscope uses this as a trigger and captures the power consumption data. Now it has data in its buffers is can fulfil the program's request to send the data to the PC. For simplicity of design there is no communication between the PC and the FPGA, the plaintexts were generated by an on-chip LFSR and there was a counter that ensured a fixed period of time occurred between encryptions. If this was greater than the time taken to transfer data between the oscilloscope and the PC then the oscilloscope would receive another request for a transfer before the next encryption took place and the process would repeat. There was another counter on the FPGA that stopped it after a certain number of encryptions had been performed, there was also a counter in the program that counted the number of datasets that had been received. If the data transfer between the PC and oscilloscope took longer than the time between encryptions the program would not capture the expected number of traces and it would be apparent that the capturing process had failed. The output from this was stored in comma delimited files ready for processing by other programs.

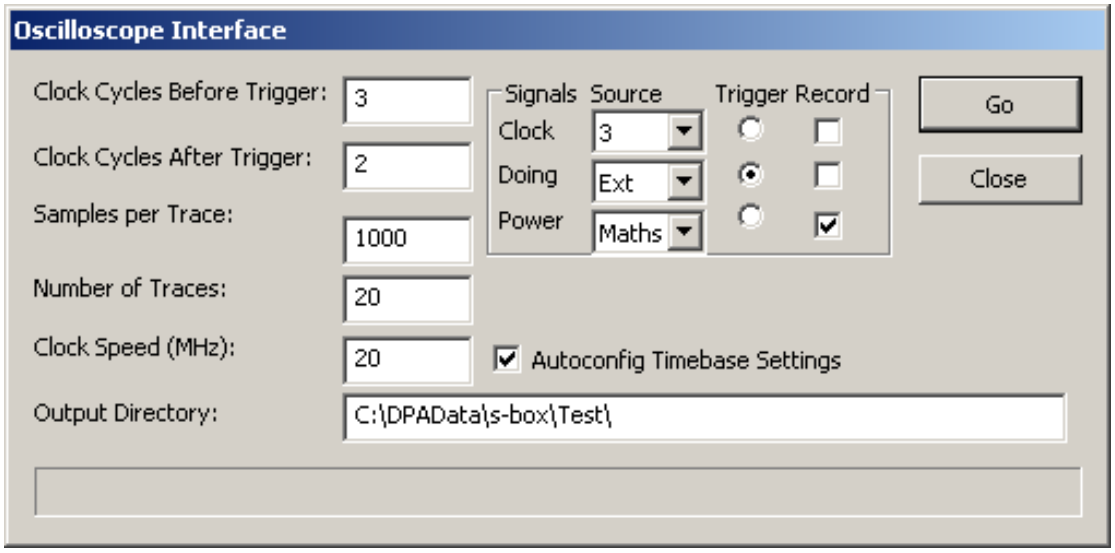


Figure 5-6: GUI for the program that controls the transferral of data between the oscilloscope and the PC.

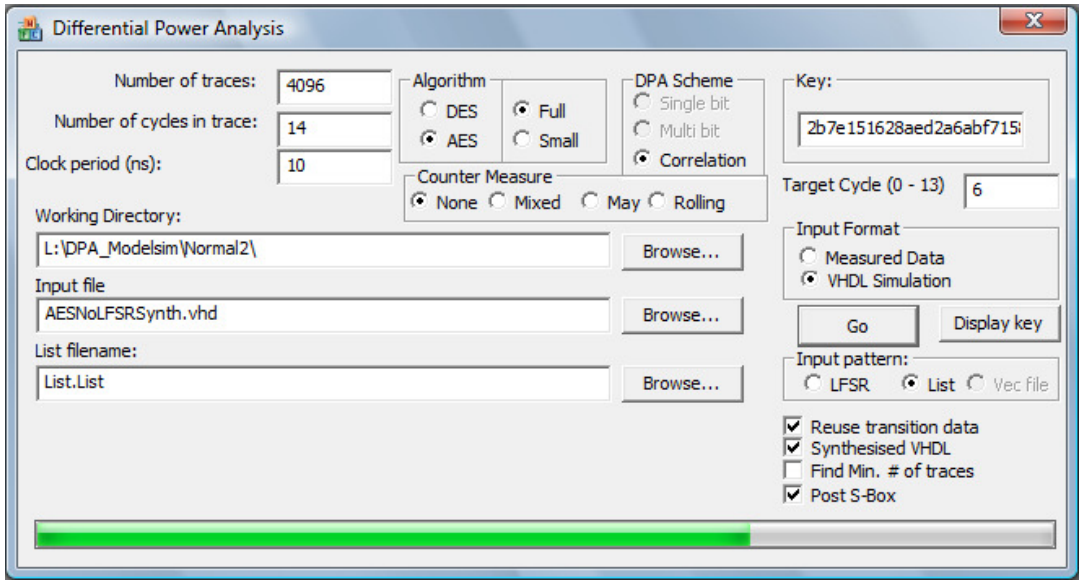


Figure 5-7: GUI for the DPA analysis program when attacking simulated power data.

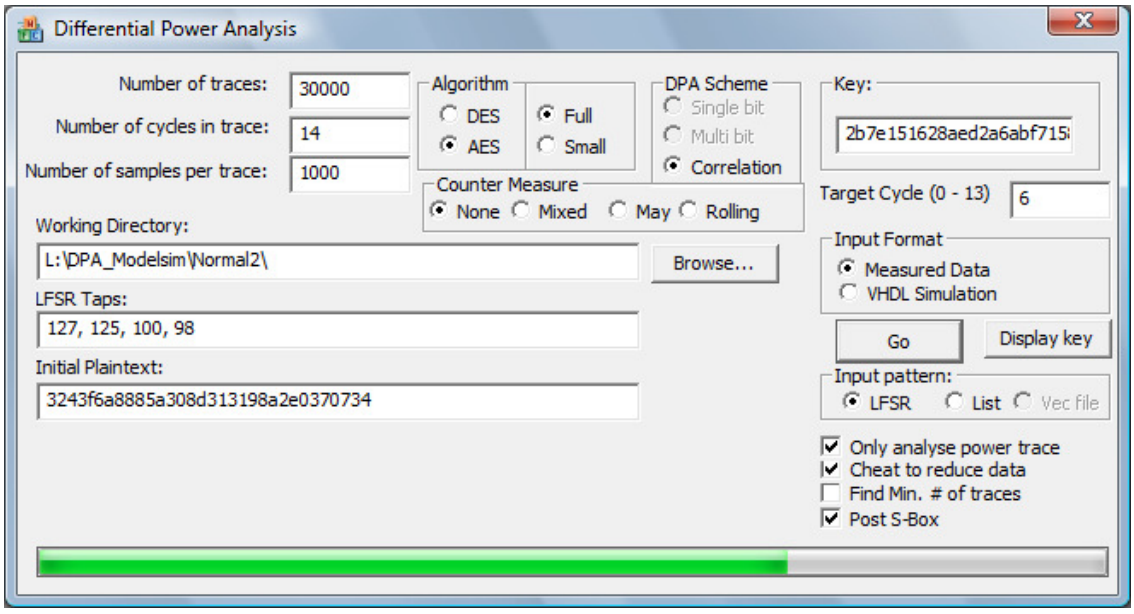


Figure 5-8: GUI for the DPA analysis program when attacking FPGA power data.

The majority of the basic level algorithms for performing the various types of correlation attacks were very similar, for this reason one program was written that incorporated the analysis of both the simulated results and the power traces captured from the FPGA. Different options in the GUI could be set to control aspects of the analysis such as the number of traces, how the plaintexts were generated, either a file with a list of them or an LFSR, and which, if any, of the countermeasures proposed in sections 7.2.1 - 7.2.3 were used. The GUI for the program when performing DPA on a simulated AES and FPGA data are shown in Figure 5-7 and Figure 5-8 respectively. There are several available settings displayed in Figure 5-8, most are self-explanatory.

In the Countermeasure section there are four options: None, for when there is no countermeasure; Mixed, for when the countermeasure is the additional Mix Columns operation (see section 7.2.2); May, for when the countermeasure was the strengthened AES key schedule developed by May *et al* [30] (see section 7.2.1); and Rolling, for when the key schedule continues expanding the key indefinitely (see section 7.2.3). It should also be noted that the option marked "cheat to reduce data" does not really cheat but performs the correlation using the correct value of the key byte (which is already known to the analysis program) to find the sample in the power consumption trace that gives the highest correlation. This is then the only sample that is used when calculating the correlation for the other 255 key-byte guesses which makes the data analysis stage much faster.

5.4 Effects of the Position of the Target Register on Correlation Attacks

The correlation attacks described in section 5.3 all target the register at the start of the first round after the initial Add Key operation (apart from Figure 5-4, where there is only an s-box and a register). This was chosen as in the AES design each round was performed in one clock cycle so there were no registers after the s-box and it was a simpler modification to reset the initial register at the start of each encryption than to alter the structure of the round. A simulation of power analysis was performed in Matlab where the target of the attack was a register that stored the results of the substitution. A graph of the correlation from all 256 key values from a simulated DPA attack is shown in Figure 5-9, and an attack on a real FPGA is shown in Figure 5-10, in both cases the correct key value was 43.

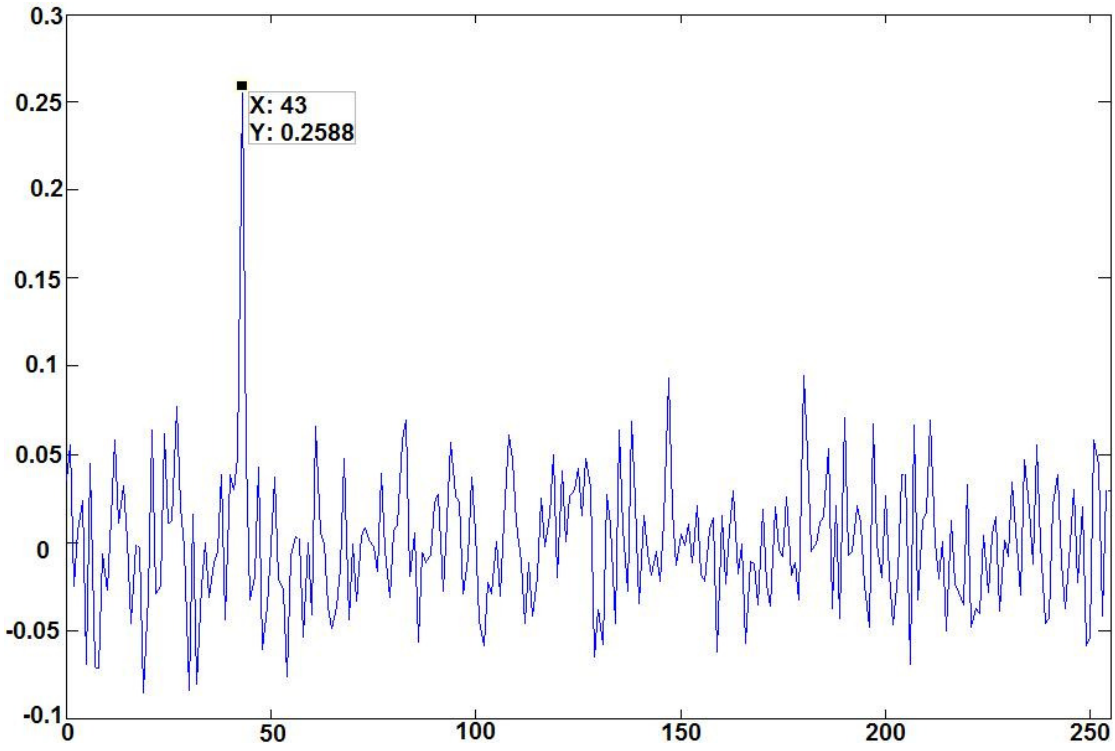


Figure 5-9: Graph showing the correlation of the 256 key guesses for the Matlab model of a 1,000 trace correlation attack on AES targeting the algorithm after the S-Box.

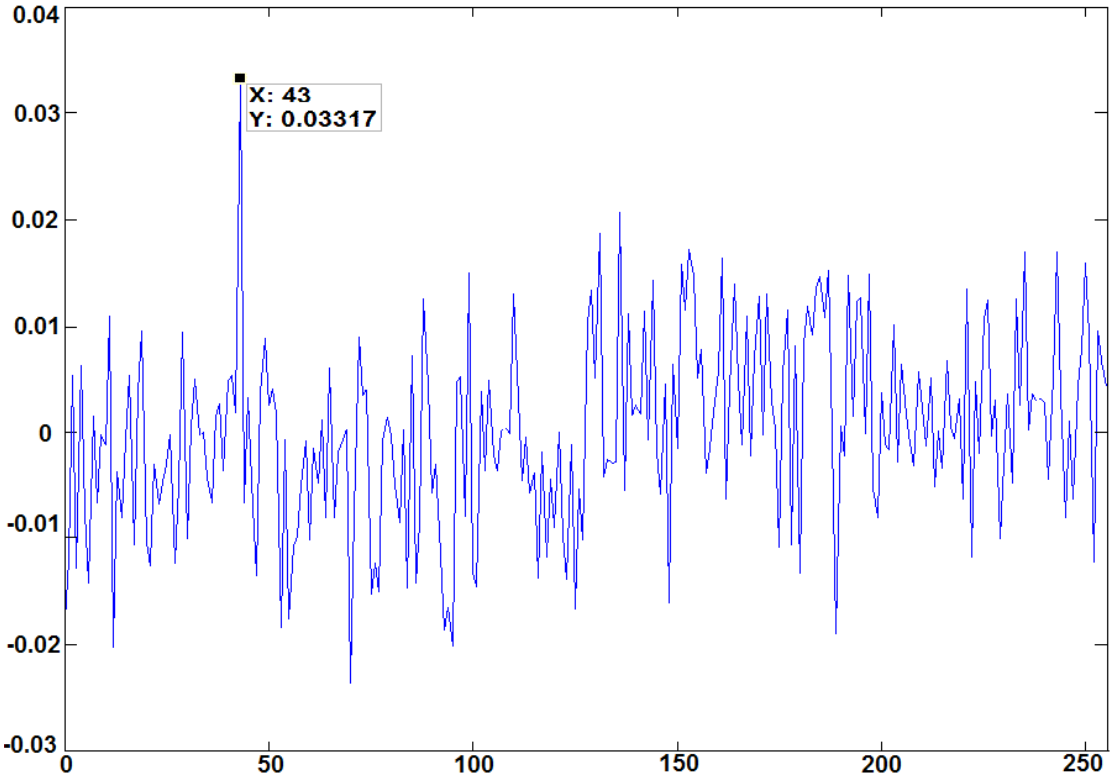


Figure 5-10: Correlation of the 256 key guesses for a 30,000 trace correlation attack on an FPGA AES implementation, targeting the algorithm after the s-box.

A number of differences become immediately apparent when comparing this graph to the one shown in Figure 5-2. In Figure 5-2 the variable plaintext byte is XORed with constant key, the statistics are like those described in section 4.3.5.1, with a reduction in the correlation by ¼, in an 8-bit attack, from the maximum for each incorrect bit in a guess. It is significantly easier to extract the correct value from a post s-box attack. This is because there is potentially a high correlation between the predictions for correct and incorrect key hypothesis when the target is (Plaintext XOR KeyGuess) as one bit difference in the key leads to only 1 bit difference in the output. In contrast, as the s-box is a complex, non-linear function then after it is applied a single bit difference in the key guess leads to a vastly different output and hence there is a much lower correlation for incorrect key guesses. As explained by Prouff in [74] the same properties that make an s-box satisfy the propagation criterion to give an algorithm resistance to linear and differential cryptanalysis also make the s-box fundamentally vulnerable to DPA.

Performing a 2-byte power analysis attack has an analogous effect on the correlation; an example is given in Figure 5-11.

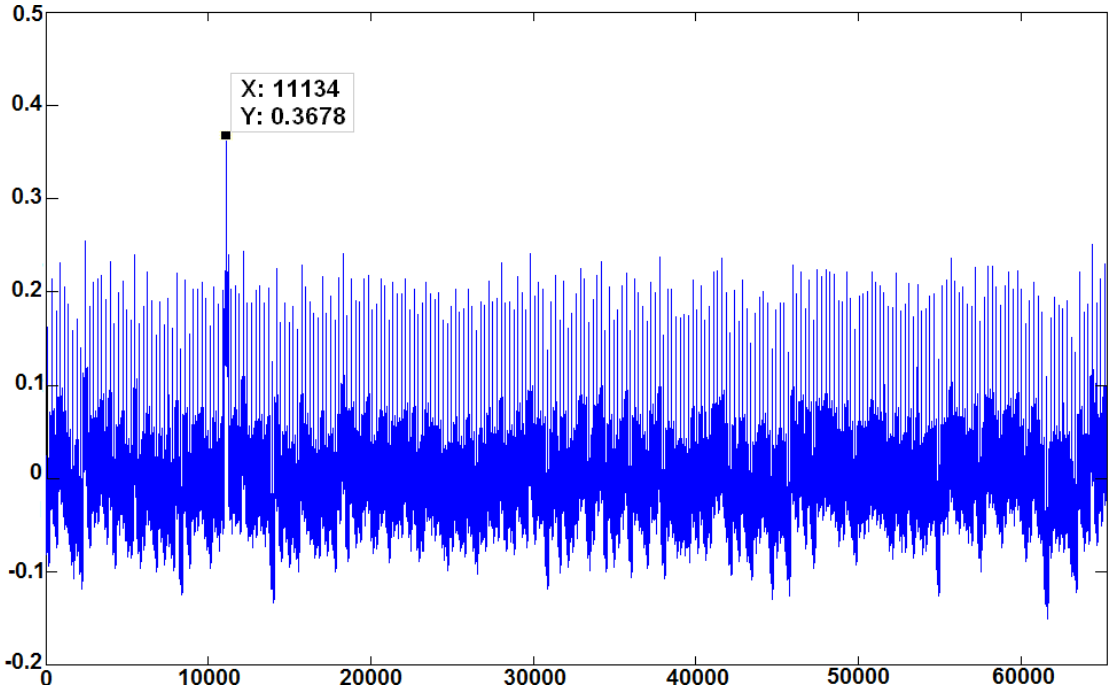


Figure 5-11: Graph showing the correlation of the 65,536 key guesses for the Matlab model of a post s-box correlation attack on 2 bytes of AES (2B 7E) with 1,000 traces.

Knowing the position of the registers in the AES design would not always be possible in a realistic situation. If the incorrect position is attacked with the prediction

function clearly this will not give the correct result. It is however unlikely that the result that is given will be confused with the correct one as they look significantly different. If the target is pre s-box when it should be after it then it looks similar to a correct attack, there is no large peak indicating the correct result but there is still the same characteristic shape in the graph, this is because there is still the same pattern in the values of the prediction matrix irrespective of whether any of the predictions are accurate. If the target is post s-box instead of before it then the graph looks significantly different, there are not the same number of levels for the correlation and the highest and lowest values are significantly reduced. This is illustrated in Figure 5-12.

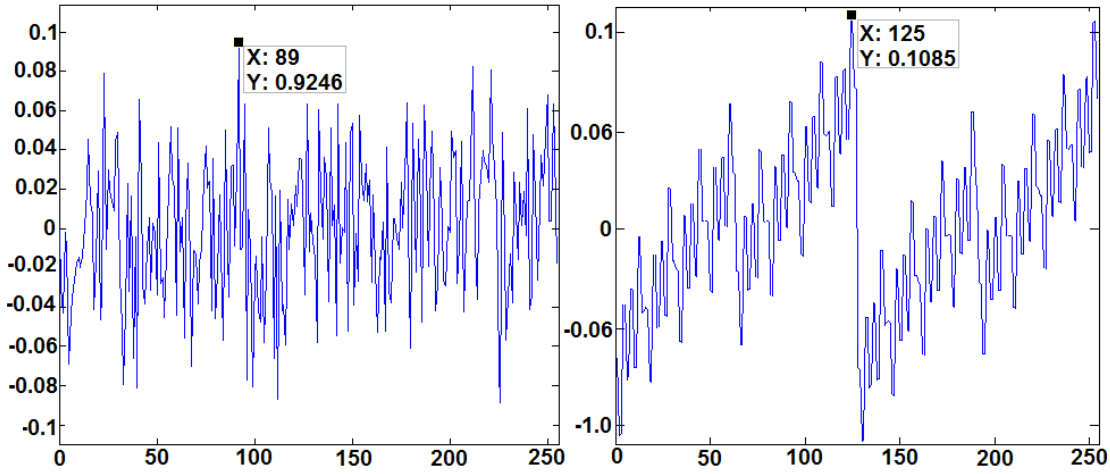


Figure 5-12: Graphs showing the results of a correlation when the attack targets the incorrect side of the s-box, the graph on the left targets post s-box and the graph on the right targets pre s-box, in both cases the correct key, 43, is not represented by the highest peak.

Figure 5-13 shows three DPA traces where the correlation values have been sorted into descending order. Each has an SNR of 0.25 but a different number of traces. As the number of traces increases the correlation line becomes flatter and it appears that the correlation for incorrect guesses will approach zero. This is however not quite the case, the amount of variability of the calculated values decreases as the number of traces increases. This is discussed further in Chapter 6.

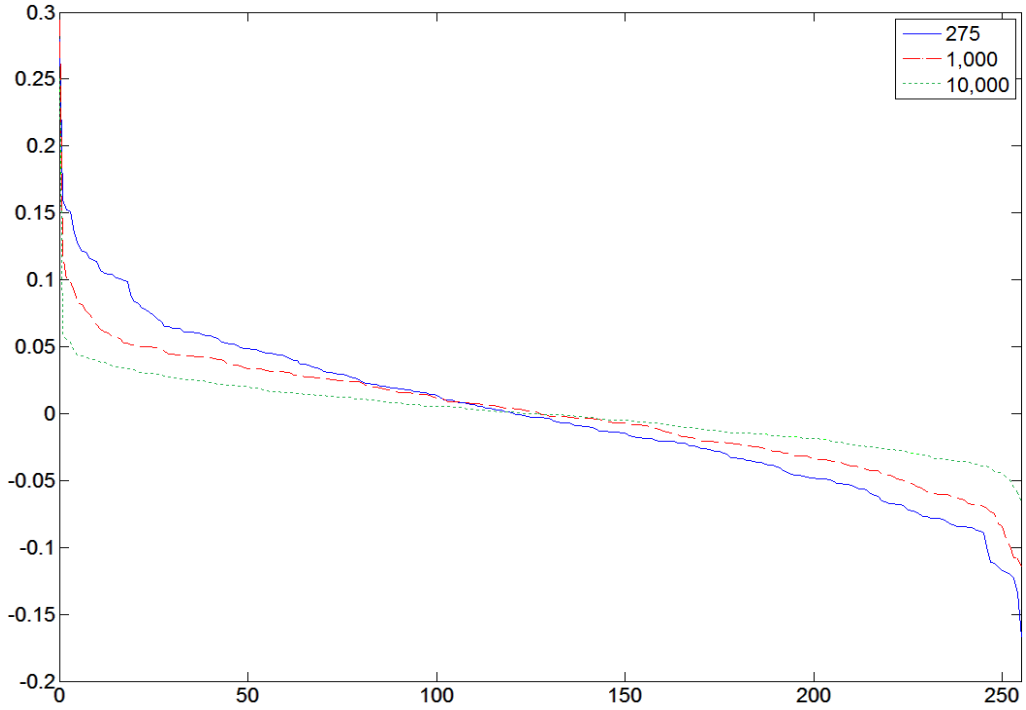


Figure 5-13: Correlation of all 256 key guesses for 3 different numbers of traces arranged in descending order.

5.5 Conclusion

A number of different methods for performing DPA have been developed. A system that records the power consumption of an FPGA configured as an AES core was made and programs that analyse the data were written. Also a method of performing DPA using Modelsim VHDL simulations was made which also proved effective, this creates measurements with much less noise than a real system which means that less power traces are required to perform DPA saving time collecting and processing data. It does only measure register transitions so any information leaked through other sources, such as logic gates, would not show up in these simulations. Additionally a model of DPA was made in Matlab, this allowed even faster experiments to be performed, which is important in order to perform Monte Carlo simulations of a DPA system so the properties of DPA can be better understood, this will be used extensively in Chapter 6.

After gaining experience of collecting and analysing DPA data it was discovered that the choice of the position in the algorithm that is attacked has a significant effect on the results and the number of traces required to successfully retrieve a the value of a byte of the key. If the value targeted is after the s-box then it

is much easier to perform DPA. This is because of the non-linear properties of the s-box, even if there is only a single bit error in the key guess this will lead a significantly different value after the s-box and hence the correlation for an incorrect key guess will be lower and therefore easier to distinguish from the correct value.

Chapter 6 The Statistics of Differential Power Analysis

6.1 Introduction

Differential Power Analysis is a statistical attack, understanding the statistical properties of it can yield techniques for evaluating the vulnerability of a system and the efficacy of countermeasures, as well as other insights into the attack. In previous sections the number of traces that were required to retrieve the key for different scenarios have been given as an indication as to the ease at which the system was cracked. Due to the statistical nature of the attack these are not definite values, but are product of the implementation being attacked, the choice of the inputs that were used and the noise in the system. Just because it required 1,000 traces to crack a system with on Monday there is no guarantee that 1,000 traces will succeed on Tuesday.

With a given number of traces and a specific level of random noise there is a fixed probability of success. In order to investigate this relationship further, a Monte Carlo simulation of DPA was performed with different numbers of traces each time using different random generated plaintexts and keys. To form the consumption matrix the plaintext was XORed with key and the Hamming weight was calculated. No additional noise was added but the contribution to the consumption model from the bytes in the plaintext that were not being attacked would act as noise. The number of times a byte of the key was successfully retrieved was recorded and the probability of success against the number of traces was plotted, this can be seen in Figure 6-1. The simulation was repeated with 16 and 32 bytes in the plaintext and key, this changes the amount of noise in the system.

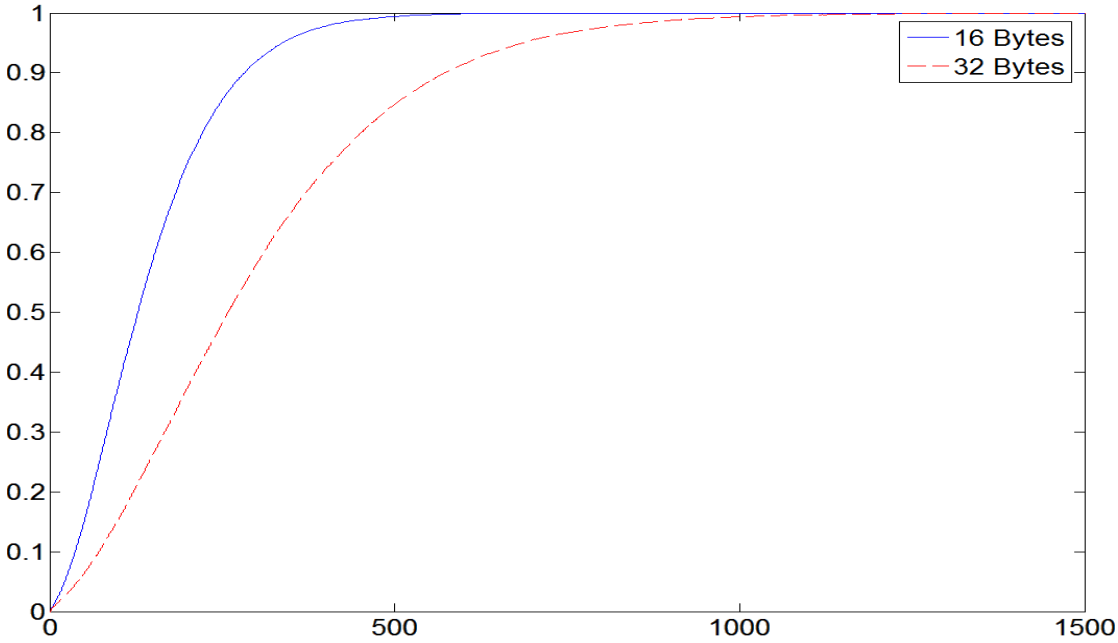


Figure 6-1: A graph showing the probability of successfully retrieving a key byte against the number of traces taken in simulation calculating 16 and 32 bytes concurrently.

It can be plainly seen from the graph that the more traces that have been used the greater the probability of success. Also, the more noise there is in the system the greater the number of traces that are required to achieve the same success rate. In order to determine the precise relationship between these variables a statistical model of DPA is derived and from this, a technique to calculate the probability of success from the amount of noise in the system and the number of traces that have been taken is established.

Detecting the effect of a particular pattern of register transitions can have other uses than divining a cryptographic key. This chapter also presents a method for using DPA to detect the presence of a particular pseudo-random sequence that has been added to a design as a sort of watermark to protect intellectual property.

6.2 Statistical Model of DPA

6.2.1 Introduction

In order for DPA to be successful the correlation relating to the set of predictions based on the correct key guess must be higher than the predictions relating to the 255 incorrect key values. The specific value of the correlation generated by the predictions based on the correct and incorrect key guesses can vary between

successive runs of DPA depending on the values of the plaintext and the key that are used and the noise in the rest of the circuit. This means the value obtained is just an *estimate* of the true correlation; this estimate is called the sample correlation as it is based on the samples taken. The true correlation is that calculated if the entire population was sampled (therefore this is referred to as the population correlation) and as the population is all potentially observable values, this implies an infinite number. This means that there is a random element to the values created and hence the chance of the guess with the largest correlation not being the correct key.

In order to investigate this relationship between the noise and variables, and to verify the accuracy of the model a Monte Carlo simulation of several DPA attacks was performed using a 16-bit key and putting it through the AES s-box. Each plaintext was then used to generate 256 prediction values for each byte. After a fixed number of plaintexts had been generated the power consumption values were correlated with each column of the prediction matrix and the correlation values for each key guess were recorded. The first byte of the key was kept constant and the rest were changed to new random values and the process was repeated a large number of times. One byte of the key was kept at a constant value so it was possible to investigate the statistics of the correlations generated by each incorrect key guess individually. There was no random noise added to the power consumption values as the 16 bytes were independent of each other so the *signal-to-noise ratio* (SNR) of each byte is given in equation (6-1) where the SNR in this case is the ratio of the standard deviations of the data dependent part of the signal and the noise.

$$SNR = \frac{1}{\sqrt{15}} \tag{6-1}$$

6.2.2 Statistical Model of DPA

In order to develop a statistical model of DPA it is important to understand the statistics of correlation. The distribution of the sample correlations around the population correlation is called the sampling distribution. Due to the fact that the value for the correlation between two variables is bounded between -1 and 1 the sampling distribution of it is not normal, as when the population correlation is positive the sample correlation can vary more in the negative direction than in the positive and

vice versa. In order to convert it to a normally distributed variable the Fisher transform [72] must be applied to the data as given in (6-2):

$$Fisher(r) = \frac{1}{2}\ln\left(\frac{1+r}{1-r}\right) \tag{6-2}$$

This means that after the Fisher transform has been applied the correlation can be modelled as a normally distributed random variable. This is only accurate when the number of samples is greater than 30, but this does not matter because in most practical examples of DPA the number of traces is much higher than 30. For the rest of this discussion all data is assumed to be after the Fisher transform unless specifically stated.

The standard deviation of a sampling distribution is called the standard error, for the post-Fisher correlation it is controlled by the number of traces that were used in the correlation and is given by (6-3):

$$StdError = \frac{1}{\sqrt{traces - 3}} \tag{6-3}$$

Correlation is a measure of how much of the variance of one variable is due to the variance of another. This is the same as how much of the total signal is made up of the information that we are interested in. This will be referred to as *PercentSignal* and is related to the SNR as shown in equation (6-4):

$$PercentSignal = \frac{1}{\sqrt{1 + {}^{1}\!/_{SNR^2}}} \tag{6-4}$$

This is the correlation of the correct prediction values with the power consumption and after the Fisher transform gives the mean of the sampling distribution for the correct key.

DPA in AES separates the key into groups of 8 bits and so there is 1 correct and 255 incorrect correlations, each taken from their own normal distribution. Section 6.2.2.1 deals with the distribution of the correlation between the power consumption and the predictions with the correct key guess; this is referred to as the *Correct* distribution. Section 6.2.2.2 describes the distributions of the correlation between the power consumption and the predictions with the 255 incorrect key guesses, referred to from now on as the *Error* distributions.

6.2.2.1 *Correlation with the Correct Key*

After the Fisher Transformation has been applied the correlation between the consumption matrix and the prediction matrix generated with the correct key forms a normal distribution. The standard deviation of the distribution is related to the number of traces and can be calculated from equation (6-5). The mean is the *PercentSignal* of the system. Figure 6-2 shows four distributions of correlation using a different number of traces.

$$StdCorrect = \frac{1}{\sqrt{traces - 3}} \tag{6-5}$$

$$MeanCorrect = Fisher(\text{PercentSignal}) \tag{6-6}$$

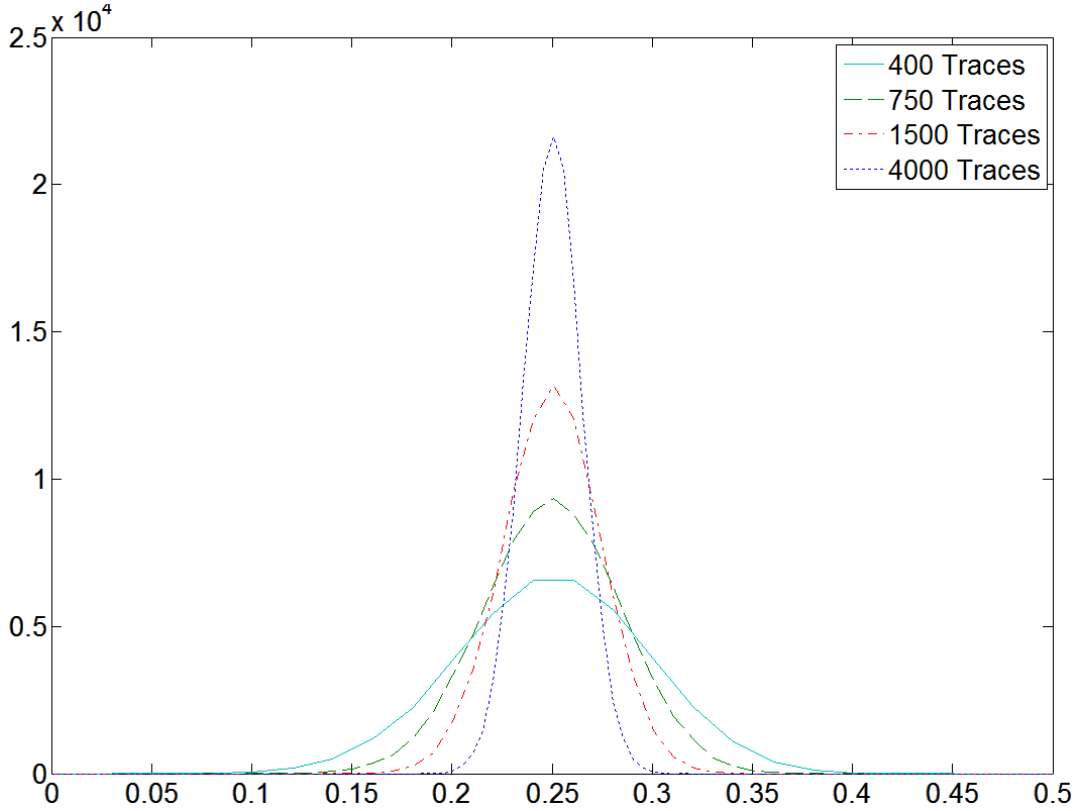


Figure 6-2: Distribution of correlations from the correct key guess using different numbers of traces.

6.2.2.2 *Correlation with the Incorrect Key*

As there are 255 different incorrect key values and hence 255 prediction matrices, clearly there have to be 255 different correlation distributions. Each will have properties controlled by signal SNR of the crypto system, the number of traces and particular details of the implementation.

6.2.2.2.1 Mean

Each of the distributions has its own mean. The values of each mean is controlled by two factors, the SNR of the system and constant that is determined by the correlation between the prediction matrix for the correct key guess and the prediction matrix for incorrect key guess relating to the distribution.

$$MeanError(i) = Fisher(corr(Error\ \Pr ed, Consumption)) \tag{6-7}$$

$$MeanError(i) = Fisher(corr(Error\ \Pr ed, Correct\ \Pr ed + Noise)) \tag{6-8}$$

$$MeanError(i) = Fisher\left(\frac{\operatorname{cov}(Error\ \Pr ed, Correct\ \Pr ed + Noise)}{\sigma(Error\ \Pr ed)\sigma(Correct\ \Pr ed + Noise)}\right) \tag{6-9}$$

$$MeanError(i) = Fisher \left(\frac{\operatorname{cov}(Error \Pr ed, Correct \Pr ed) + \operatorname{cov}(Error \Pr ed, Noise)}{\sigma(Error \Pr ed)\sqrt{\sigma^2(Correct \Pr ed) + 2\operatorname{cov}(Error \Pr ed, Noise) + \sigma^2(Noise)}}\right) \tag{6-10}$$

By definition the noise and the prediction matrices are independent, therefore this reduces to:

$$MeanError(i) = Fisher\left(\frac{\operatorname{cov}(Error\ \Pr ed, Correct\ \Pr ed)}{\sigma(Error\ \Pr ed)\sqrt{\sigma^2(Correct\ \Pr ed) + \sigma^2(Noise)}}\right) \tag{6-11}$$

$$MeanError(i) = Fisher\left(\frac{corr(Error\ \Pr ed, Correct\ \Pr ed)\sigma(Correct\ \Pr ed)}{\sqrt{\sigma^2(Correct\ \Pr ed) + \sigma^2(Noise)}}\right) \tag{6-12}$$

$$MeanError(i) = Fisher\left(\frac{corr(Error\ \Pr ed, Correct\ \Pr ed)}{\sqrt{1 + \sigma^2(Noise) \Big/ \sigma^2(Correct\ \Pr ed)}}\right) \tag{6-13}$$

From equation (6-13) and (6-4) it can been seen that this is:

$$MeanError(i) = Fisher(corr(Error\ \Pr ed, Correct\ \Pr ed) * PercentSignal) \tag{6-14}$$

The correlation between the *Correct* and *Error* prediction matrices can be estimated to a reasonable degree of accuracy by randomly generating large prediction matrices for the 256 key values and calculating the correlation between them. As this is a relatively quick calculation (compared to simulating an entire DPA system) a

large number of samples can be used, this will reduce the standard error of the estimate which is given in (6-3).

Clearly the *Correct* prediction matrix, and therefore the *MeanError*, is affected by the choice of key. This means that in the general case, where the key is not specified, there are 256 different possible sets of arrays of *MeanError*. In order to investigate this all 256 * 255 values were estimated using a large number of samples. This is discussed further in section 6.4.

6.2.2.2.2 Standard Deviation

The standard deviation of the distribution of correlations is controlled by the number of samples in the correlation, this is the same for both the *Correct* and *Error* distributions.

$$StdError = \frac{1}{\sqrt{traces - 3}} \tag{6-15}$$

6.2.2.2.3 Correlation between the Correct and Error distributions

Now both the mean and standard deviation have been found it would be easy to think that the model is complete, unfortunately the relationship is not quite that simple. There is a correlation between the *Correct* and *Error* distributions. This means that if a sample from the *Correct* distribution is above the mean it affects the probability of a sample from each of the *Error* distributions being above their mean. This also has to be modelled.

Like *MeanError* the correlation between *Correct* and *Error*, referred to as *CorrCorr*, is controlled by the correlation between their respective prediction matrices and the SNR of the system. Figure 6-3 shows the variation of *CorrCorr* with *PercentSignal* when the correct key is 0 and the key guess is 1. When *PercentSignal* approaches 0, when there is a large amount of noise, *CorrCorr* tends towards the correlation between the *Correct* and *Error* prediction matrices. As the *PercentSignal* approaches 1, when the noise drops off to 0, the *CorrCorr* tends towards another constant that can be determined by simulation. The relationship between *CorrCorr* and *S* can be modelled using the inverse tan function, as shown in (6-16):

$$CorrCorr(i) = \frac{(Last - (C * (\tan^{-1}(\text{PercentSignal} * 4.5 - 2.25)) + (C + Last)}{2} \quad (6\text{-}16)$$

$$where \quad C = corr(ErrP(i), CorP)$$

Where *Last* is the value that *CorrCorr* tends towards when *PercentSignal* is approaches 1. The modelled curve is plotted next to the actual curve in Figure 6-3**.**

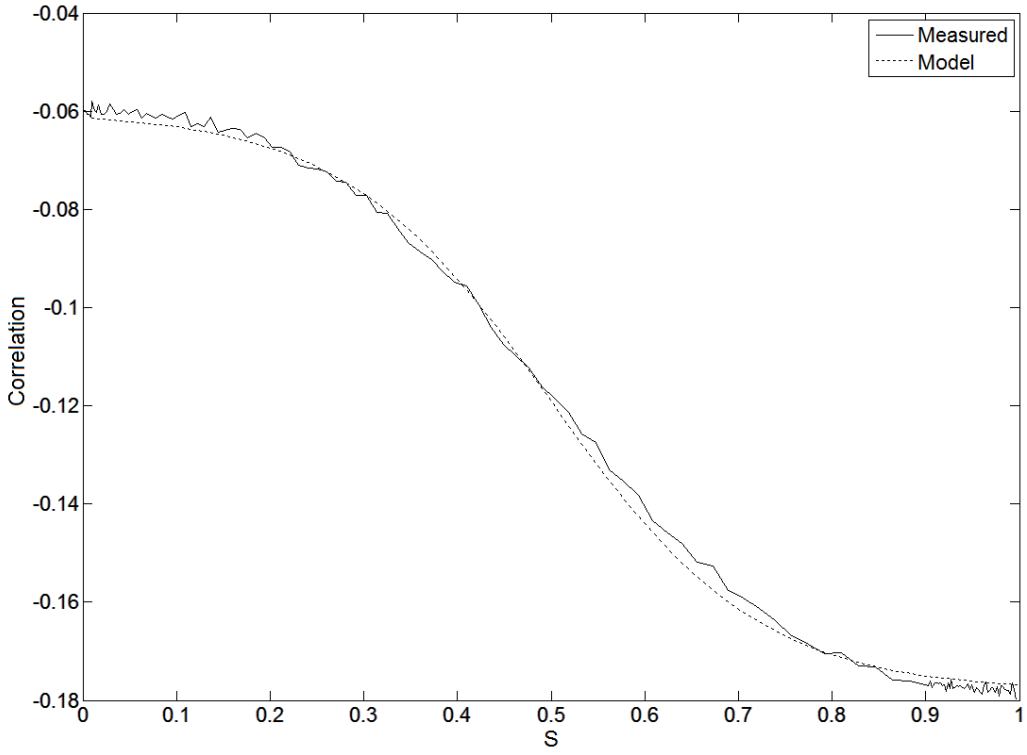


Figure 6-3: The correlation of Correct and Error and the model curve versus *PercentSignal*

6.2.2.3 *Summary of the Model*

The *Correct* correlation can be modelled by the normal distribution shown in equation (6-17) where $N(\mu, \sigma)$ is a normally distributed random variable with a mean of μ and a standard deviation of σ. The *Error* correlations can be modelled as the normal distributions described in (6-18), where $i \in 0 \ldots 254$.

$$Correct = N(MeanCorrect, StdCorrect) \quad (6\text{-}17)$$

$$Error(i) = N\left(MeanError(i) - CorrCorr(i) * MeanCorrect, \sqrt{(Std^2 - (Std * CorrCorr(i))^2}\right) + CorrCorr(i) * Correct \quad (6\text{-}18)$$

6.3 Predicting Success

From the model developed in section 6.2 we can generate a formula to determine the probability of successfully determining the correct key using DPA on a system. This is achieved by calculating the probability of the highest of the 255 *Error* values being greater than a particular value, denoted by *t*, and the *Correct* value equalling *t*, then integrating across all possible values of *t*.

$$P(Success) = \int_{-\infty}^{\infty} P(\max(Error) \le t \cap Correct = t)dt \tag{6-19}$$

$$P(Error_i \le t \cap Correct = t) = P(Error_i \le t \mid Correct = t) * P(Correct = t) \tag{6-20}$$

$$P(\max(Error_i \le t) \mid Correct = t) = \prod_{i=0}^{254} P(Error_i \le t \mid Correct = t) \tag{6-21}$$

$$P(Success) = \int_{-\infty}^{\infty} \prod_{i=0}^{254} P(Error_i \le t \mid Correct = t) * P(Correct = t)dt \tag{6-22}$$

$$P(Correct = t) = \varphi_{MeanCorrect,Std}(t) \tag{6-23}$$

The probability of an *Error* distribution being less than the *Correct* when it is at *t* is the probability of the distribution that was used to model the lack of perfect correlation between the two variables being less than *t*.

$$P(Error_i \le t \mid Correct = t) = \Phi_{\frac{MeanError(i) - CorrCorr(i)*MeanCorrect}{\sqrt{Std^2 - (Std*CorrCorr(i))^2}}, \frac{1 - CorrCorr(i)}{\sqrt{Std^2 - (Std*CorrCorr(i))^2}}}(t) \tag{6-24}$$

$$P(Success) = \int_{-\infty}^{\infty} \prod_{i=0}^{254} \Phi_{\frac{MeanError(i) - CorrCorr(i)*MeanCorrect}{\sqrt{Std^2 - (Std*CorrCorr(i))^2}}, \frac{1 - CorrCorr(i)}{\sqrt{Std^2 - (Std*CorrCorr(i))^2}}}(t) * \varphi_{MeanCorrect,Std}(t)dt \tag{6-25}$$

Where $\varphi_{\mu,\sigma}$ is the probability density function (PDF), and $\Phi_{\mu,\sigma}$ is the cumulative probability density function (CDF) for the normal distribution. Unfortunately this cannot be directly evaluated as the CDF for the normal function does not have any elementary primitives and so certainly cannot be integrated. There are techniques to approximate it, however, so the integration in (6-25) can then be approximated using numerical integration.

It is important to note that the formula calculates the probability of successfully extracting 1 byte of the key. In order to calculate the probability of successfully retrieving the entire 16-byte AES key the resultant value would need to be raised to the 16^{th} power.

6.3.1 Calculating the Other Variables

While it is useful to calculate the probability of success from the number of traces for a given signal to noise ratio, it would be more useful for an attacker to be able to calculate the number of traces required to ensure a particular probability of success on a system with a given SNR, and more useful to a designer to be able to determine the amount of noise in a system that would require the attacker to take a particular number of traces if they wanted to have a given probability of success.

Due to the formula not being able to be evaluated directly, it is not possible to rearrange it for these purposes. It is however possible to use the original technique to perform an iterative search for the value of the desired variable. This entails either the SNR or the number of traces and making an initial estimate of the value of the other one that will give the desired probability of success. The probability of success is then determined and if the initial guess was too low it is increased, if it was too high then a binary search can take place to efficiently determine the correct value.

6.3.2 Testing the Formula

In order to verify the efficacy of the formula an FPGA implementation of an AES s-box was fed 50,000 random plaintexts XORed with a constant byte and the power consumption was recorded, DPA was performed for the correct key using all 50,000 traces to get a accurate estimate of the population correlation, the standard error of the estimate was 0.0045 and the correlation was estimated to be 0.8642. DPA was then performed on 20 traces from the 50,000 taken at random and it was recorded whether the correct key was retrieved, this was repeated 100,000 times. The success rate was 76.24%. The estimated *PercentSignal* was used to evaluate the probability of success when 20 traces were used, the result returned was 72%. The process was repeated on a similar design of an AES s-box, this time with a correlation estimate of 0.4874, the predicted success rate for 20 traces was 26.54% and the actual success rate was 29.358%. This is clear evidence that the method described above is a good

indicator of the probability of successfully retrieving a key using DPA given the SNR of the system and the number of traces to be used in an attack. This also implies that the statistical model of the attack is accurate.

6.4 Relative DPA Susceptibility of Keys

The result of the correlation in DPA is affected by the values that are used to compute it and therefore the inputs that are used when performing the encryption. In DPA it is assumed that the plaintexts are random and as normally a large number are used their effect will average out. Each byte of the key is considered independently and only has one value for the entire attack. It is conceivable that the choice of that value can influence the probability of success, this section investigates that possibility.

The probability of successfully retrieving a key is related to the relative position of the mean of the incorrect distributions compared to that of the correct distribution. This is controlled by the SNR of the system and the correlation between the prediction matrices for the Correct and Error distributions. The correlations between prediction matrices is controlled by the structure of the s-box so this analysis is only valid for algorithms that use the AES s-box. As stated in section 6.2 there is a set of 255 *MeanError* values for each possible key value. As they are different it is possible that different keys have different levels of susceptibility to DPA. To investigate this all 65,280 different *MeanError* values were estimated for a particular prediction function using a large number of samples. Each row has similar values, a KS test is not able to reject the null hypothesis that the values are drawn from different distributions.

These values are just estimates, the possibility that the true values for different keys are the same was tested by calculating the lowest value for all different keys with two different numbers of samples and the Fisher transformation was applied to the results. If the variation in the values was due to an estimation error then the values would be normally distributed around the true value with a standard deviation related to the number of samples used in the estimate of the correlation. The standard deviation for both sets of values was 0.0035; the standard deviations predicted by the number of samples for the two estimates were 0.001 and 0.00057. Additionally the distribution of values was not normal, the Lilliefors test, a version of the KS test optimised for normality testing, showed this. This refutes the possibility that the

variance in the values of the correlation between the *Correct* and *Error* prediction using different keys is due to the inaccuracy of the estimates.

This leaves the problem of how to select which of the 256 sets of *MeanError* to use. The most accurate choice would be to calculate the probability of success for all 256 key values and take the average, this would be rather computationally intensive as it would require calculating 65,280 mean values in addition to performing 256 integrations. How worthwhile this is, is determined by the overall effect the difference in key value has on the calculated probability.

In order to investigate the variability in probability of success due to the difference in key, the probabilities for all keys were calculated for a particular system, these are shown in Figure 6-4. While there is a difference it is very small, the standard deviation of the probabilities is 0.000091. The variation is much lower than other errors in the system and so it makes little difference which of the 256 sets of values is used.

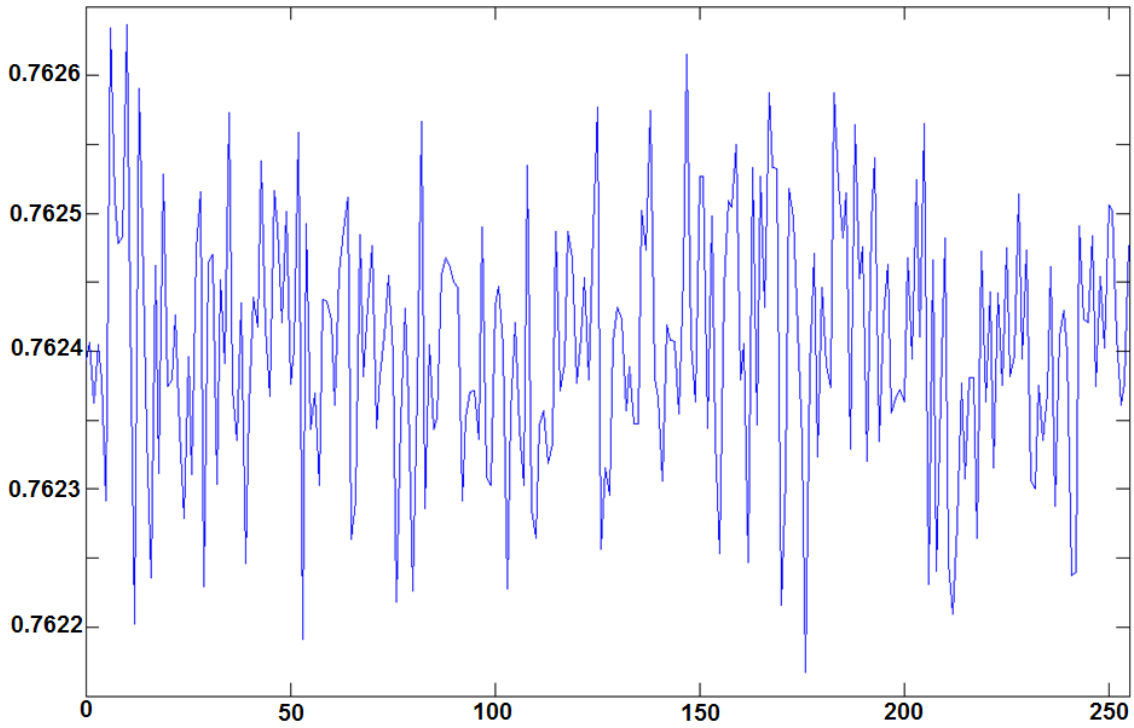


Figure 6-4: The probability of successfully retrieving a key using DPA for the different possible values of the key.

6.5 Protecting Intellectual Property Using a DPA Detectable Watermark

6.5.1 Introduction

Intellectual Property (IP) is a valuable commodity and can form the main source of income for a company. It is, therefore, important to protect it. There have been several proposed methods for achieving this. In [100] Alkabani and Koushanfar propose adding a series of initial states to a state machine that require a unique and unpredictable set of inputs that only the designer knows to bring the device into its functional state. Koushanfar, Hong and Potkonjak developed techniques for adding a signature to the structural properties of designs [101].

If the design of an integrated circuit contains a small section that produces a known bit pattern in a set of registers, like a pseudo random bit generator (PRBG), then it would be possible to use DPA to detect this. This would act like a watermark and would be useful for determining if a piece of hardware contains the relevant piece of intellectual property (IP).

6.5.2 Power Consumption Watermarks

6.5.2.1 *Adding a Watermark*

In order to detect a watermark in the power consumption of a device a characteristic fingerprint needs to be added to it. This can be achieved quite simply by the addition of a pseudo random bit generator (PRBG). The pseudo random, but deterministic, values generated in the registers of the PRBG will add a specific pattern to the power consumption.

6.5.2.2 *Measuring Power Consumption*

In order to determine whether a watermark is present in the power consumption of a design the power consumption must be recorded. Assuming the PRBG generates a new multi-bit value each clock cycle then the power consumption needs to be sampled each clock cycle. The correlation between the power values and the Hamming distance in the registers of the PRBG in successive values is then calculated.

6.5.3 Detecting the Watermark

If the watermark is present in the power consumption then the population correlation will have the value of the ratio between the standard deviation of the power consumption of the watermarking hardware and the standard deviation of the total power consumption. If the IP is included within a larger design it may not be possible for the rights holder to know the value of this. They could measure the standard deviation of the total power consumption but the standard deviation of the power consumption of the watermarking hardware would be dependent on the technology that it was implemented on, so would not necessarily be the same as their reference version.

What is known is that if the watermarking hardware is there then the population correlation will be positive and if it isn't then it will be 0. After the sample correlation has been calculated it can be determined whether it is reasonable to reject the null hypothesis that the correlation was drawn from a distribution with a mean that is not greater than zero and hence there is no watermarking hardware present. In order to do this a p-value is calculated using a z-table. The p-value is the probability of observing by chance a result that is at least as extreme as the one being tested. A z-table contains the probabilities of a standard normal distribution, one with a mean of 0 and standard deviation of 1, being greater than a set of values.

$$z = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}} = \frac{F - 0}{1/\sqrt{N-3} * 1/\sqrt{1}} = F\sqrt{N-3} \tag{6-26}$$

It is determined by looking up on a z-table the associated probability for the value of z which can be calculated with (6-26) where $\bar{x}$ is the mean sample correlation, in this case the Fisher transform of the sample correlation, μ_0 is the value of the mean in the null hypothesis, in this case 0, σ is the standard deviation of the sampling distribution, given by (6-2) and n is the number of samples in the mean of the sample correlation, as only one correlation is being calculated this is 1.

6.5.3.1 *Summary of method*

In order to tell whether the power consumption data supports the presence of a watermark the following steps must be taken:

1) The power consumption (P) of the device is measured from its reset state and the Hamming distance (H) of the registers in the PRBG for the same number of samples (T) is recorded.

2) The correlation between the two is calculated.

$$\rho = \text{Corr}(P, H) \tag{6-27}$$

3) The Fisher transform is applied to the correlation.

$$F = \text{Fisher}(\rho) \tag{6-28}$$

4) The confidence level must be decided. This is the probability incorrectly detecting a watermark when there is none. A typical value is 0.05.

$$C = 0.05 \tag{6-29}$$

5) The p-value is calculated.

$$P = Z\left(F\left(T\text{-}3\right)^{1/2}\right) \tag{6-30}$$

6) If the p-value is lower than the confidence level then the null hypothesis can be rejected and the watermark has been detected.

6.5.3.2 *Experimental Results*

Watermark	Yes	No	Yes	No	Yes	No
Samples	5000	5000	1000	1000	1000	1000
σ_{Total}	1	1	1	1	1	1
$\sigma_{\text{Watermark}}$	0.05	-	0.1	-	0.05	-
Null Hypotheses rejected (%)	97	4.8	93.9	5.0	46.6	4.8

Table 6-1: Summary of the simulation results.

A series of simulations was performed in order to verify the method. The power consumption of the watermarking hardware was modelled by generating a series of random numbers between 0 and 255 and calculating the Hamming distance between them, giving values between 0 and 8. Noise was included by adding a series of normally distributed random numbers to the model. The noise represents both the power consumption from the rest of the circuit and any non-linearity in the power consumption vs. Hamming distance. The correlation was performed between the Hamming distance values and the power consumption model. This was repeated

10,000 times for different numbers of samples and with different amounts of noise. Simulations of hardware power consumption with no watermark were also performed. In these, the Hamming distance for a watermark was calculated in the same way and they were correlated with normally distributed random numbers that had the same standard deviation as the total power consumption for the watermarked simulations. The results are summarised in Table 6-1.

6.5.3.3 *Type I and II Errors*

There are two types of errors when trying to detect a watermark: detecting one that is not there and not detecting one that is. The probability of incorrectly rejecting the null hypothesis and falsely claiming there is a watermark is the significance level chosen for the p-value test. This is why the number of times a watermark was detected in the simulations when there was none was always approximately 5% irrespective of the number of samples taken and the population correlation.

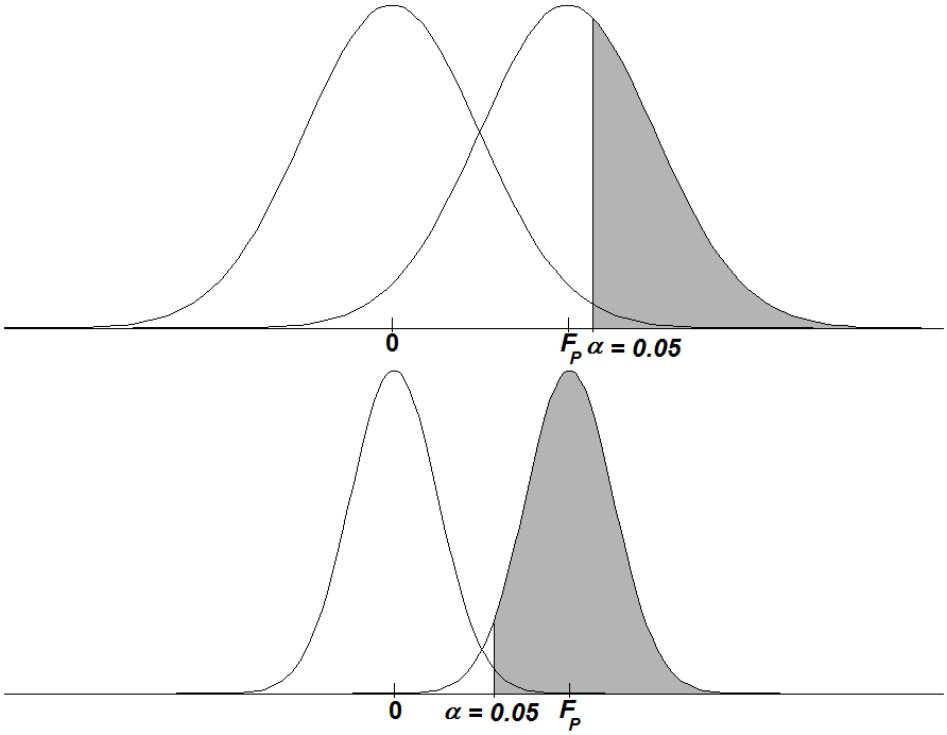


Figure 6-5 : A graph illustrating the effect of increasing the number of samples of the ease of detecting a watermark

The probability of not detecting a watermark that is there is controlled by two factors: the population correlation and the number of samples. The smaller the population correlation the greater the number of samples that must be taken to ensure the same probability of detecting the watermark. This is because increasing the

number of samples reduces the standard deviation of the sampling distributions making it easier to differentiate between the two. This is illustrated in Figure 6-5, the curves in both plots are normal distributions with the same mean but different standard deviations, the shaded area represents the amount of the sampling distribution that the correlation can come from in order to reject the null hypothesis with a confidence of 0.95. It is clearly more likely to successfully detect the watermark from the lower graph.

6.5.4 Calculating the Number of Traces if the Population Correlation is Known

In the previous section it was assumed that the population correlation could not be reasonably estimated before trying to detect the watermark. While this would most likely be the case if the IP in question is an entire chip design then the population correlation could be estimated.

If this is the case then it is possible to use this information to calculate the number of power consumption samples that need to be recorded in order to give a particular probability of successfully detecting the watermark. The following method can be used to calculate the number of samples required to give a 90% chance that the null hypothesis will be rejected at the 0.05 level. First the z-table is consulted to find the value that the standard normal distribution has a 95% chance of being lower than and 90% chance of being higher than, these will be referred to as $z_{<95}$ and $z_{>90}$.

To reject the null hypothesis at 0.05 the correlation must be higher than equation (6-31).

$$\frac{z_{<95}}{\sqrt{N-3}} \tag{6-31}$$

Also, there is a 90% chance that the correlation will be greater than equation (6-32).

$$\frac{z_{>90}}{\sqrt{N-3}} + F_P \tag{6-32}$$

In order for there to be a 90% chance of rejecting the null hypothesis at the 0.05 level these two values must be the same, this is demonstrated in Figure 6-6. The dark grey shaded area represents 90% of the area under the right hand curve which is the sampling distribution of the correlation. The light grey shading represents 95% of the area under the left hand curve which is what the sampling distribution of the correlation would be if the null hypothesis was correct.

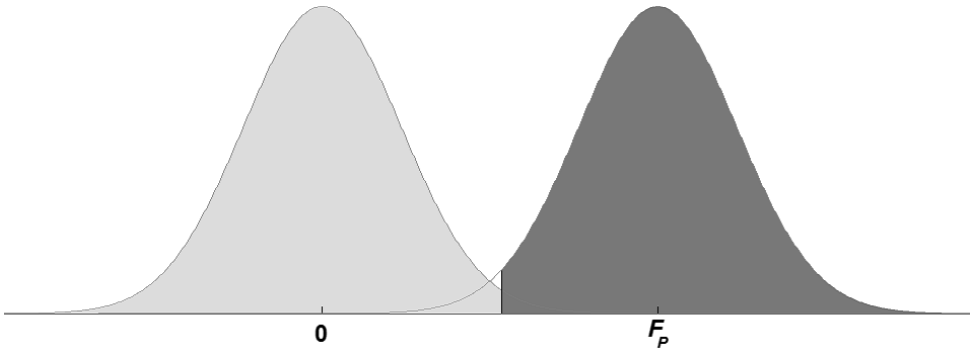


Figure 6-6: A graph illustrating the requirements for rejecting the null hypothesis at the 0.05 level 90% of the time when the population correlation is known.

The number of traces required to achieve this is given by equation (6-35).

$$\frac{z_{>90}}{\sqrt{N-3}} + F_P = \frac{z_{<95}}{\sqrt{N-3}} \tag{6-33}$$

$$F_P = \frac{z_{<95} - z_{>90}}{\sqrt{N-3}} \tag{6-34}$$

$$N = \left(\frac{z_{<95} - z_{>90}}{F_P} \right)^2 + 3 \tag{6-35}$$

In order to verify this method a simulation was performed, 100,000 correlations were performed with 10,000 samples where the population correlation was 1/34. A population correlation of 1/34 would require approximately 10,000 samples to reject the null hypothesis at the 0.05 level 90% of the time and with 10,000 traces the sample correlation would have to be 0.0165 or greater as shown by equations (6-36) and (6-37) respectively.

$$N = \left(\frac{1.6449 - (-1.2816)}{Fisher(1/34)} \right)^2 + 3 = 9{,}897 \approx 10{,}000 \tag{6-36}$$

$$\frac{1.6449}{\sqrt{10{,}000 - 3}} = 0.0165 \tag{6-37}$$

Out of the 100,000 correlations generated in the simulation 90,231 were higher than this value so would have rejected the null hypothesis at the 0.05 level, this is approximately 90%.

6.5.5 How much area should be given to the watermarking hardware?

The probability of successfully detecting the watermark is related to the population correlation, which is determined by the power consumption of the

watermarking hardware. The greater the amount of area that is dedicated to watermarking hardware the easier it will be to detect but the larger the overhead involved. Assuming that power consumption is directly proportional to area the population correlation of the watermark can be estimated using (6-38) where W is the percent increase in area due to the watermarking hardware.

$$\rho_P = \frac{W}{\sqrt{1+W^2}} \tag{6-38}$$

Using this and equation (6-36) it is possible to calculate the number of samples that would be required to give a chosen probability of successfully detecting a watermark with a given significance level. Table 6-2 gives the number of samples required to successfully detect a watermark 90% of the time with a significance level of 0.05 after different amounts of area have been dedicated to the watermarking hardware.

When performing DPA to retrieve cryptographic keys the samples are relatively difficult to collect, the attacker has no control over when they are generated or how many are generated before a new key is used. When using DPA to detect watermarks this is not true and it is easy to collect however many are deemed necessary. It is not unlikely that 10 million samples could be quite easily taken, this would give a good chance of detecting a watermark that added less than 0.1% to the area of a design.

Increase in area	Samples	Increase in area	Samples
10%	862	0.5%	342,560
5%	3,431	0.2%	2,141,000
2%	21,415	0.1%	8,563,900
1%	85,644		

Table 6-2: The number of samples required to detect a watermark using a given percentage of the hardware with a 90% accuracy.

6.5.6 Conclusion

It is possible for designers to add hardware to their designs that will create a known pattern of register transitions as a way of using DPA to detect whether their IP has been used without their permission. As DPA is a statistical technique it does not give a definite answer, but if the correlation between the watermarking register

transitions and the power consumption is calculated it is easy to calculate the probability that the measured result would have been observed assuming the watermark was not there. If this is sufficiently unlikely it gives reasonable confidence in the falseness of the null hypothesis and by extension the hypothesis that the watermarking is present. The overhead incurred by this protection can be chosen by the designer and a method of calculating the amount of effort they would need to go to in order to detect it based on their choice is also presented. It has been shown that the overhead can be very low (< 0.1%) and still produce a signal that can be most likely detected with a realistic amount of data collecting.

It is important to note that the p-value is not the probability that the null hypothesis is true, but the probability of getting the observed result given that the null hypothesis is true. Bayes' theorem could be used to convert between these two probabilities but it would involve knowing the probability that the watermark is present (without having performed any tests to see if it is) and the probability of getting the observed correlation (without any knowledge or assumptions about the sampling distribution that it was drawn from). It is not practical to estimate these values.

6.6 Conclusion

As DPA is a statistical attack it is important to understand the statistical properties. The analysis of these properties has led to a technique for calculating the probability of key retrieval with a given number of traces for a particular system with a known SNR. This technique can be easily adapted to calculate one of the other variables, each being arguably more useful to either an attacker or a designer of a crypto-system. A designer may wish to calculate the amount of noise that must be present in a system in order to reduce the probability of a successful attack to a given level assuming the attacker has access to a known and finite number of traces. An attacker, having previously analysed the power consumption for the SNR of the data dependence may wish to know how many traces he is required to take to give him a good chance at retrieving the entire key. Additionally it has been determined that although there is a slight variation in the susceptibility of different key values to DPA the overall effect is negligible, so there are no particular key values that it would be best to avoid if concerned about DPA susceptibility. Improving the understanding of

the mathematics of the attack was one of the main aims of the thesis. This has been achieved.

The ability of DPA to discern a characteristic pattern of processed data in the power consumption of a device can be put to more benign uses than cracking encryption. If a section is added to a design to generate a known pattern of register transitions then it would be possible to use this as a watermark to detect theft of intellectual property.

Chapter 7 Novel Algorithmic-Based Power Analysis Countermeasures

7.1 Introduction

All the countermeasures to power analysis attacks described previously have been added to implementations of a cryptographic algorithm. As summarised in section 4.3.8.5, they come at a large cost in terms of either the speed of the implementation or its requirements in memory or area, and when their effectiveness is evaluated they, at best, simply frustrate the attacker, forcing him to collect more power consumption data, or perform High-Order DPA, rather than stopping the attack completely. It is unlikely that any countermeasure that involves attempting to eliminate the leakage of information through the power consumption will ever be completely effective. In can be seen from the relationship between the SNR of the data dependence in the power consumption and the number of power traces required to give a particular probability of successfully that was derived in Chapter 6 that any correlation, no matter how small, can be exploited by an attacker to discern the key if they have enough traces. Any attempts to remove the correlation will be imperfect and will leave the device vulnerable.

If DPA is ever going to be completely eliminated as a potential avenue of attack a new method will be required. Rather than adding ad-hoc and expensive countermeasures a better alternative would be if algorithms were secured against this type of attack when they were designed. This chapter describes the investigation into

some possible algorithmic countermeasures. It is important to note that in this thesis only the resistance to power analysis attacks is examined. There has been no investigation into the effects these changes have on the general security of the algorithm, although all the modifications involve measures that ought to strengthen algorithms, either adding additional layers of existing cryptographic primitives or increasing the confusion and diffusion properties of the key schedule.

Section 7.2 describes the evaluation of a set of different ideas on how to modify AES to protect it from DPA. Only one of the ideas, a perpetually expanding key schedule (section 7.2.3) is effective. In section 7.3 TDES is modified to be DPA resistant, however the algorithm isn't well suited to the countermeasure and it comes at a significant cost in terms of area. Other modern cryptographic algorithms were investigated to determine their suitability to the countermeasure, this is described in section 7.4 and one of them, ARIA, is shown to perform well with the modification in section 7.5. This leads to a second attempt at protecting TDES in section 7.6 with much better results.

7.2 AES Algorithm Alterations

This section describes the investigation into the resistance to power analysis attacks imparted to a modified version of the AES algorithm by several techniques, and their effect on the resources required to implement it. The countermeasures were added to the same AES design that was used in section 5.3.1.1 and 5.3.2; a 128-bit encryptor with an online key schedule that calculates one round per clock cycle. When the design was synthesised for a Virtex-E 1000 it used 2,446 slices and 516 flip flops, and it had a clock speed of 33.4 MHz, giving a throughput of 427 MB/s. The first modification is based on the strengthened key schedule described in section 4.2.2.2. The next uses an additional Mix Columns operation before the first round to decrease the predictability of the target registers. The final one uses a constantly changing key to remove the ability of the attacker to exploit data across a large number of encryptions.

7.2.1 Strengthened Key Schedule

The AES key schedule has a number of weaknesses as described in section 4.2.2.1. A version of AES was implemented with the strengthened key schedule

described in 4.2.2.2 in order to test whether the new key schedule adds any resistance to power analysis attacks. As the first sub-key is no longer the master key and the key schedule is a one way function in order to fully crack this algorithm all sub-keys must be extracted. After the first sub-key is extracted it is possible to predict values in the first round up to the Add Key operation. From here the attack can be performed again to extract the next sub-key, this will increase the amount of computation an attacker has to do.

7.2.1.1 *Effects on the Efficiency of the Algorithm*

There were significant changes to the design of the AES core. The key schedule used the same hardware as the encryption so they had to be calculated consecutively rather than concurrently. This meant that a more complicated controller was needed, so the number of registers increased to 521 and the number of clock cycles increased to 54. The clock speed was 31.2 MHz, so the throughput fell by 83% to 74 MB/s. Although the design had a more complicated controller the datapath was reused by the key schedule, this reduced the total size of the design to 1,699 slices, 69% of the size of the original AES implementation, fulfilling the third property in May *et al*'s list of requirements for a good key schedule [30].

The throughput of the new algorithm is significantly reduced because each round requires a round key which takes four clock cycles to generate. If the design was changed from an online key schedule to an offline one this would make significant savings in time, the round keys would still take 40 clock cycles to calculate but it would only have to happen once so the average number of clock cycles required to encrypt one plaintext would return to 10. Clearly the offline key schedule would require more area than an online one, but as the datapath is reused in this scheme it would still require less area than a standard AES offline key schedule.

7.2.1.2 *Attack on a Simulated System*

In order to determine if the new key schedule afforded any protection from DPA an attack was performed using a Modelsim simulation as described in 5.3.1.1. The attack was successful. A graph of the correlation for 256 possible key values of the first byte of the first round key is shown in Figure 7-1. The key byte had the value 35, or 0x23, it can be seen as the largest peak in Figure 7-1. Although the number of registers in the design increased compared to normal AES none of the new ones store

new data during the target clock cycle so there are no additional transitions and hence the signal to noise ratio remains the same. The first sub-key was retrieved using 761 traces.

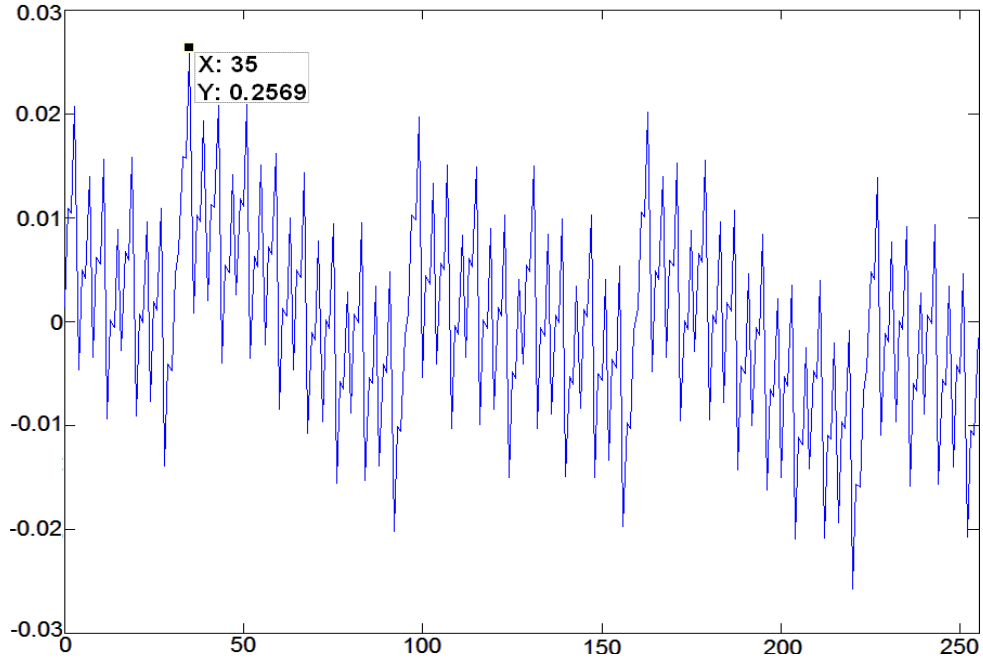


Figure 7-1: Graph showing the correlation after 1000 traces of the 256 key guesses for a DPA attack on a Modelsim simulation of an FPGA running AES with a strengthened key schedule targeting the first byte of the first round key before the s-box.

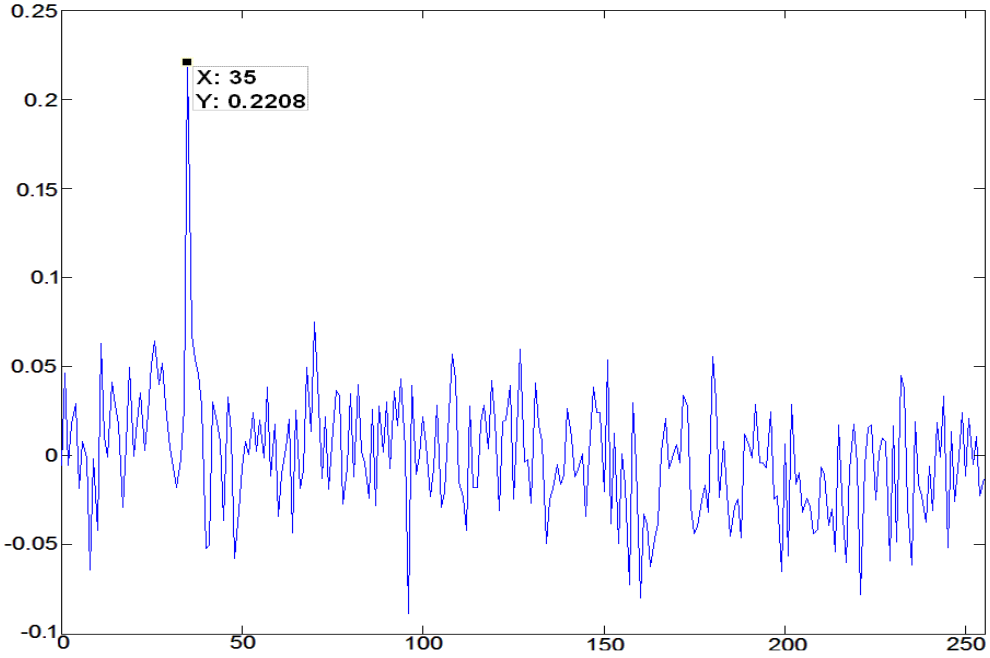


Figure 7-2: Graph showing the correlation after 1000 traces of the 256 key guesses for a DPA attack on a Modelsim simulation of an FPGA running AES with a strengthened key schedule targeting the first byte of the first round key after the s-box.

When the target was changed to after the Sub Bytes operation the number of traces required to crack it fell to 291. A graph of the correlation for 256 possible key values of the first byte of the first round key when the DPA attack targets the data after the s-box is shown in Figure 7-2, again the correct value of the key is clearly shown as the largest peak.

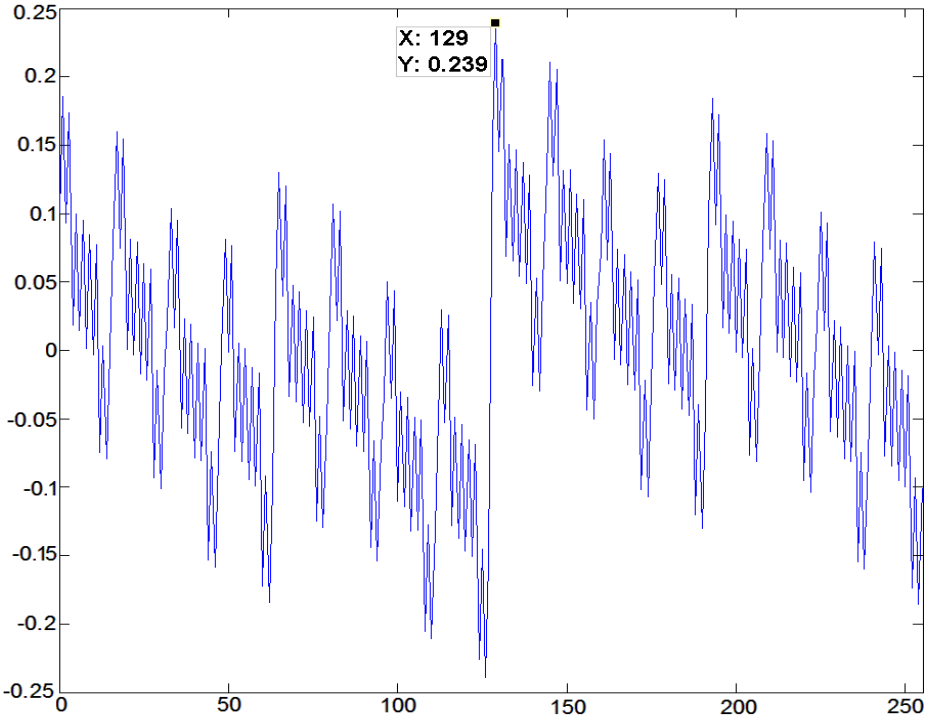


Figure 7-3: Graph showing the correlation after 1000 traces of the 256 key guesses for a DPA attack on a Modelsim simulation of an FPGA running AES with a strengthened key schedule targeting the first byte of the second round key before the s-box.

As mentioned in section 4.2.2.2 the round keys are created using a one way function and the master key is no longer used in the first round. This means that all round keys must be cracked to allow decryption of the ciphertext. Once the first round key has been retrieved this can be fed into the algorithm and the 128-bit state just after the first Mix Columns operation can be calculated, this is the data that interacts directly with the key data in the Add Key operation in the first round. The DPA algorithm can be repeated using this value instead of the plaintext to retrieve the second round key. To do this on the improved AES algorithm required 999 traces if the target was before the s-box and 332 if it was after. Graphs of the correlations for the 256 key values for these two attacks, before and after the s-box, on the first byte of the second round key are shown in Figure 7-3 and Figure 7-4 respectively. The

value of the first byte of the second round key was 0x81, or 129 in decimal notation, and is revealed as the largest peak in both graphs.

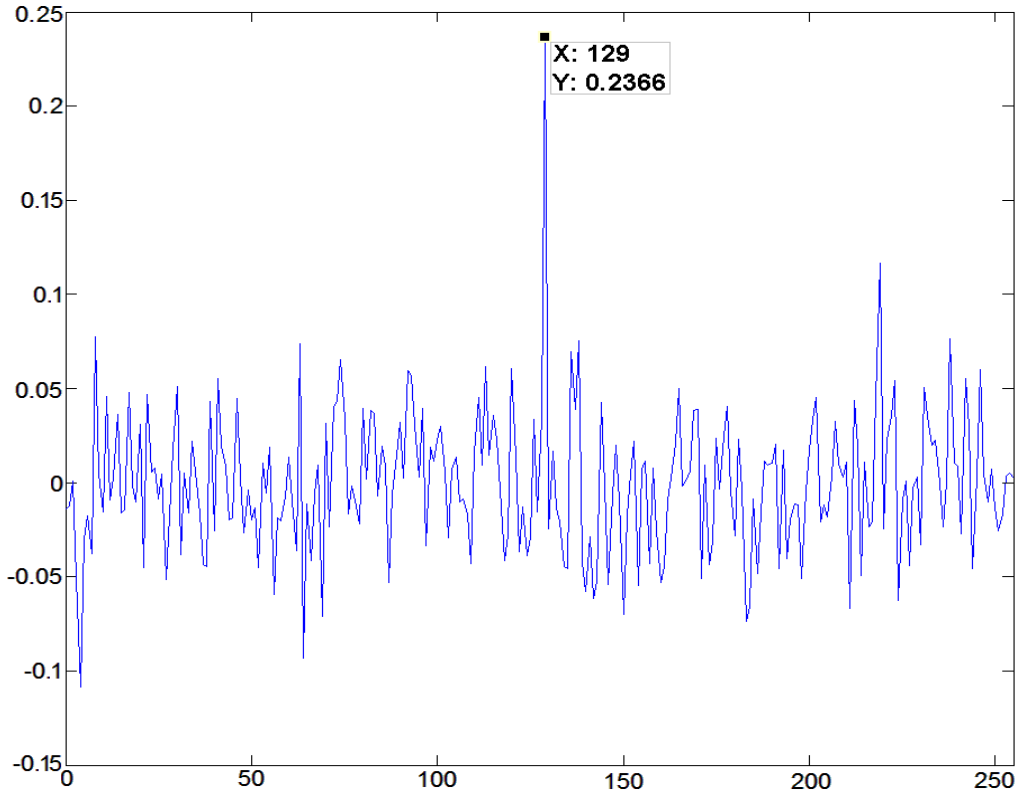


Figure 7-4: Graph showing the correlation after 1000 traces of the 256 key guesses for a DPA attack on a Modelsim simulation of an FPGA running AES with a strengthened key schedule targeting the first byte of the second round key after the s-box.

7.2.1.3 *Conclusion*

While the modifications to the key schedule that were proposed in [30] arguably increase the security of the algorithm against some attacks, DPA is not one of them. While it is more irritating for the attacker as they have to run the DPA analysis once for each round key, the attack is no harder and they can still use the traces from the same encryptions used to crack the first key for the second and hence no additional traces are required compared to standard AES.

7.2.2 Addition of Initial Diffusion

As explained in section 4.3.2.3, attacks like DPA and correlation attacks cannot target all positions in an algorithm but can only yield useful information when the target registers are full and predictable. The efficiency of these attacks is related to the fact that one byte of the target is related to one byte of the plaintext and one byte of

the key. If an extra Mix Columns operation is added to the algorithm after the initial Add Key but before the first set of registers then all bits of the key and plaintext in 1 column have an effect on the value and therefore the Hamming weight of the register. This means that instead of checking the correlation of the 256 columns of the prediction matrix corresponding to all possible values in one byte of the key all 2^{32} must be checked. It would be possible to increase this to 2^{128} by adding an analogous mix rows operation.

It is important to note that, unlike later on in the algorithm, the Mix Columns comes after the Add Key; this is because the unpredictable element, the key, has to be mixed with the known plaintext. A block diagram of the modified algorithm is shown in Figure 7-5. It is important to note that the Mix Columns operation must be between the Add Key and the first register. If there are any registers between the two then this can be the target for the attack and the countermeasure is rendered useless. For this reason this is only suitable for an FPGA or ASIC implementation rather than a microprocessor as each byte of the Add Key would be calculated and stored in registers on different clock cycles.

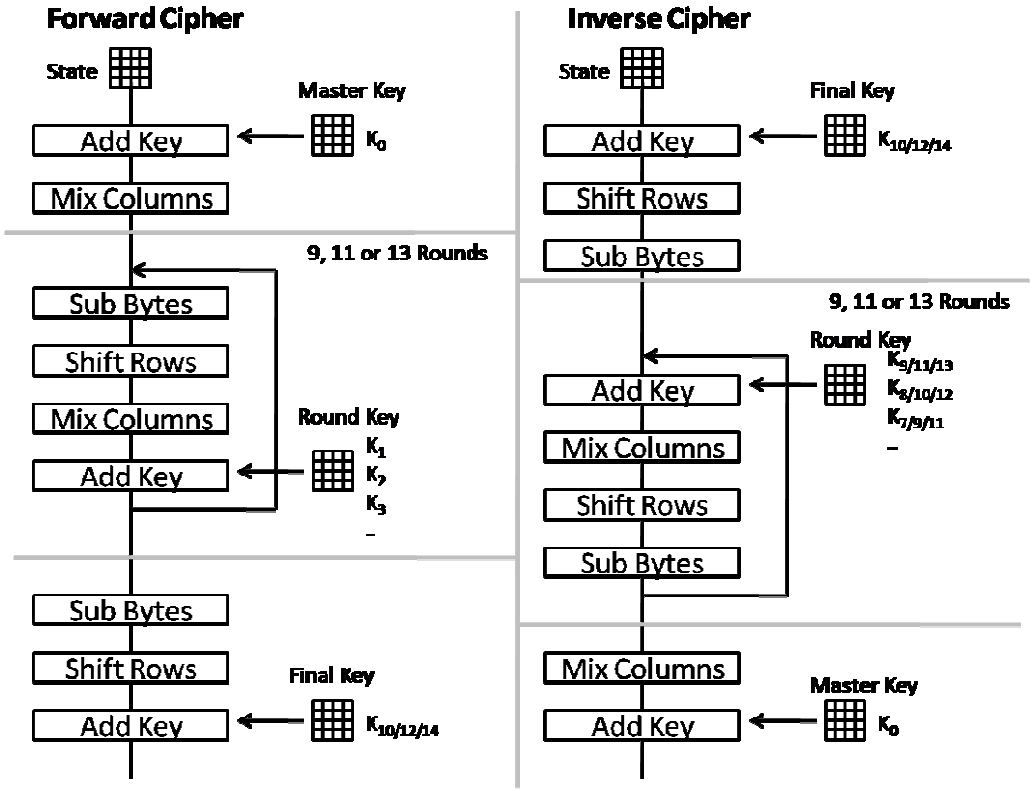


Figure 7-5: Diagram showing the structure of the algorithm with extra initial diffusion.

7.2.2.1 *Effects on the Efficiency of the Algorithm*

There is very little difference between an implementation of an encryptor of this algorithm and one of normal AES. There are still 516 registers, a small combinational section was added, but due to some slight differences in implementation the number of slices used in the design fell to 2,395 when synthesised for a Xilinx Virtex-E. The clock speed fell to 27 MHz, reducing the throughput by 20% to 346 MB/s.

7.2.2.2 *Attack on a Simulated System*

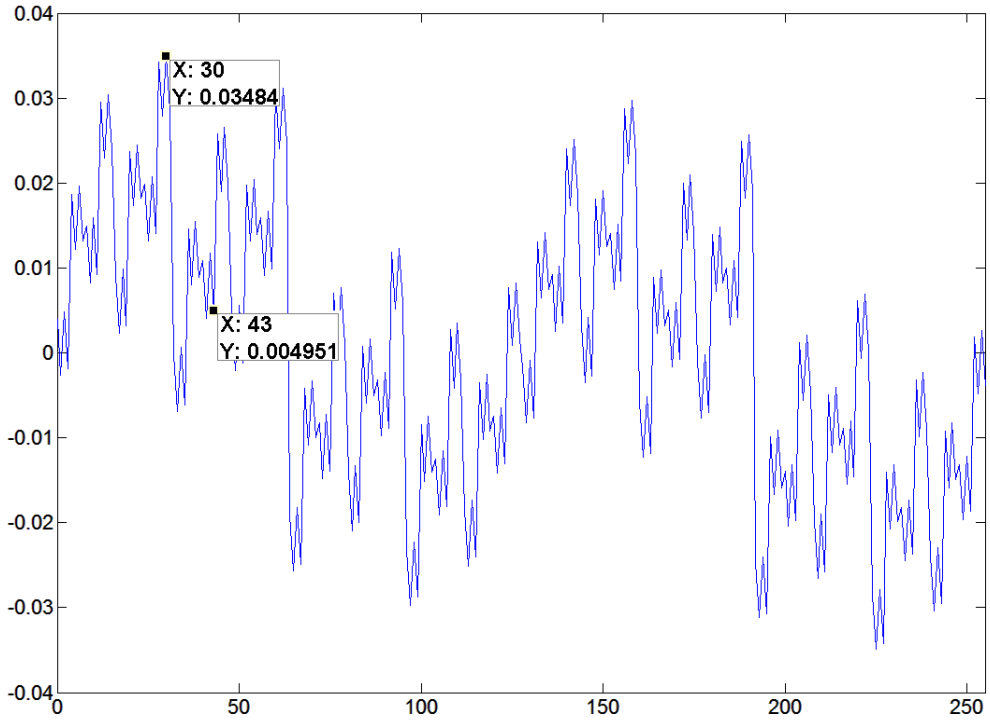


Figure 7-6: Graph of the correlation for each key guess when attacking the first byte of the sub-key of AES with additional diffusion using the normal DPA algorithm.

There is the same number of registers in this design and the design of regular AES that was attacked in section 5.3.1.1. This means that there is the same amount of noise in the system and hence if DPA was successful it should identify the correct key with a correlation of approximately 0.2. The first byte of the key was targeted, it had the value 43 and 4,096 traces were used. Using normal DPA the key could not be retrieved. The correlation from each key guess is shown in Figure 7-6. The largest peak is at 30, with a value of 0.03484, much lower than what would be expected from a series of correct prediction about the transitions in the target register. The correct value of 43 has a correlation of 0.004951. It is not feasible that an attacker could determine the correct key from this data. Another attempt at DPA was made, this time

it was assumed that the attacker knew the other values in the column of sub-key bytes. The correlation from each key guess is shown in Figure 7-7 correct value of 43 had the highest peak at 0.2398. Using this approach required 674 traces to successfully retrieve all 16 bytes of the key.

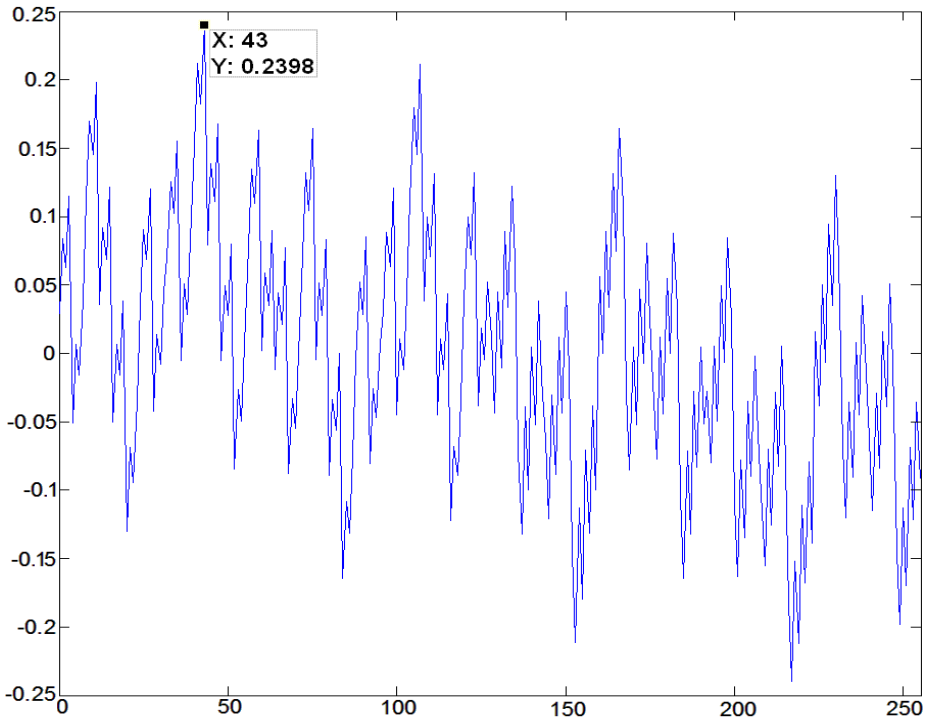


Figure 7-7: Graph of the correlation for each key guess when attacking the first byte of the sub-key of AES with additional diffusion using a DPA algorithm that assumes the attacker known the other sub-key bytes in the column.

7.2.2.3 *Conclusion*

This countermeasure increases the key space that must be searched in order to determine the correct key as each byte that is stored in the registers is dependent on an entire column of the key. As stated previously this technique relies on the fact that the Mix Columns operation is performed before any values are stored in registers, this means that the technique is unsuitable for use with microcontrollers as they would not be able to perform it atomically. Even if algorithms are protected with this system it would be possible for a naive designer to put registers in the wrong place in the design and open up DPA vulnerability. Also, if the attacker knows the values of some of the bytes in the key then DPA can be used to discover the values of other bytes.

It is possible that a technique similar to that used in [93], where the target for the DPA is not the contents of a register but the output of logic gates, would still be

able to retrieve the key from this algorithm. This would require more detailed knowledge of the design and, as the logic gates consume less power than registers, more traces, so it would make DPA harder, but not impossible.

7.2.3 Perpetually Expanding Key Schedule

The reason DPA is so effective is that it is able to combine the information from several encryptions, combining the small variations in power consumption into usable information. This is only possible because the attacker is able to make predictions of the values inside registers based on a hypothesis of the value of a byte of the key. In AES the key is expanded once and this data is used for every block, this means the key hypothesis is valid across all of the encryptions using the same key.

The proposed modification would continue to expand the key so the same round keys are not reused. This undermines the attacker's ability to make predictions about the values inside registers. Due to the nature of the AES key schedule, after each round of key expansion the value of each byte becomes dependent on the value of an additional byte in the original key. This means that after encrypting two plaintexts with the modified algorithm the value of each byte in the round key is dependent on all bits in the original key. This means that to make any accurate predictions of the value of any hypothesis about the value of one byte assumptions about all bytes would need to be made. The attack would therefore offer no advantage over brute force.

7.2.3.1 *Effects on the Efficiency of the Algorithm*

Other than potentially providing resistance to DPA this countermeasure is that there can no longer be random access of the encrypted data as the key for each block is different and has to be calculated in sequence. This is not necessarily that much of a disadvantage as ECB is the only mode of operation for block ciphers that allows this and using this method can lead to some insecurities, like replay attacks.

Ultimately the effect of this modification on the efficiency of the implementation depends on what features the implementation requires. For an implementation that is only an encryptor it is not practical to have an offline key schedule, as the round keys will not be the same for the next encryption so have to be re-calculated anyway. As the round keys are used in reverse for decryption any implementation must store them when decrypting, for normal AES an offline key

schedule would be the obvious choice. With the new approach the implementation would have the disadvantages both the online and offline styles, the key would have to be calculated each time and additional memory would still be required.

Implementation	Clock (MHz)	Cycles / Result	Throughput (Mb/s)	Slices	DFFs
128-bit Enc	29.1	12	310	2,576	651
128-bit Enc/Dec off.	28.1	21	171	4,575	2,074
128-bit Enc/Dec Unroll.	6.3	11	73	6,395	780
128-bit Enc/Dec Piped	17.5	11 - 13	203-172	6,394	1,164
128-bit Enc/Dec 2*Mem.	26.4	11	307	5,616	3,209

Table 7-1: A summary of the performance of different implementation s of modified AES with a perpetually expanding key schedule.

In order to further investigate the performance impact of the modification 4 different versions of the algorithm were made and synthesised for a Xilinx Virtex-E with a speed grade of -6. The performance of these designs is summarised in Table 7-1. A standard 128-bit encryptor was made, this was very similar to the implementation of the unmodified algorithm.

Next an implementation that could also decrypt was made. As decryption requires the keys in reverse order the keys have to be calculated first and stored, this doubles the number of clock cycles required for processing a block. In order to try and make the implementation more efficient the key schedule was unrolled, this means that the entire expanded key is calculated combinationally, it does increase the area requirements as there are four s-boxes required for the generation of each round key, also the critical path becomes significantly longer, reducing the maximum clock speed. In order to increase the clock speed of the unrolled implementation registers were added in the key schedule, this means the key expansion is performed across several clock cycles, increasing the number of cycles per result but the critical path falls. With three blocks of registers in the unrolled key schedule the clock cycle increased to 17.5 MHz and the throughput increased to 172 Mbits/s for decryption,

this is significantly higher than the unrolled version but there is only a marginal increase in throughput compared to the offline version.

An alternative way of increasing the throughput was then investigated that involved doubling the amount of memory the key schedule has and while reading the round keys for the processing of one block it calculates and stores the keys for the next. Using this technique the throughput was returned to nearly that of the encryptor, there is a significant increase in the amount of area used, over 1,000 more slices compared to the simple decryptor, and clearly it involves nearly twice the amount of flip flops for storing the round keys.

7.2.3.2 *Attack on a Simulated System*

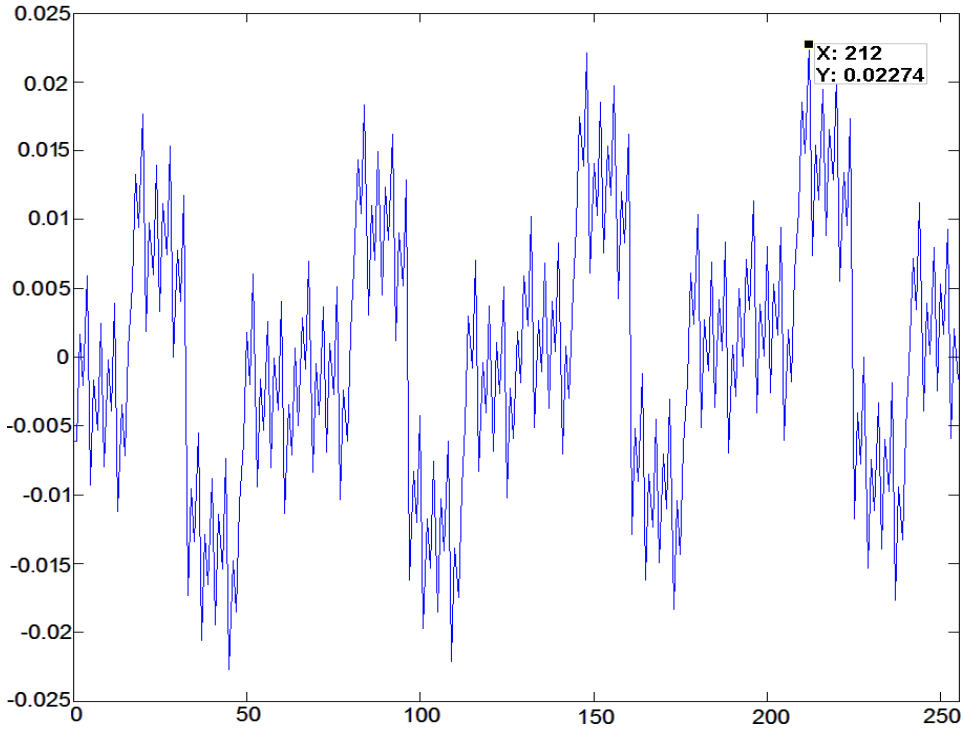


Figure 7-8: Graph of the correlation for each key guess when attacking the first byte of the sub-key of AES with a perpetually expanding key schedule before the s-box using the normal DPA algorithm with 4096 traces.

A DPA was performed on a simulation of a device running the modified algorithm. The initial value for the first byte of the first sub-key was 43, 0x2B. A graph of the correlation of the key guesses after 4,096 traces when the target register is before the s-box is shown in Figure 7-8, the key guess with the largest peak was 212 giving a correlation of 0.02274. This is not the correct key value and the correlation value is much lower than what would be expected for a successful attack

given the signal-to-noise ratio of the system. The correlation after 4,096 traces when the target register is after the s-box is shown in Figure 7-9, the key guess with the largest peak was 111 giving a correlation of 0.04413. This is also not the correct key value and the correlation value is again much lower than what would be expected for a successful attack, the highest peak does not really distinguish itself from the rest of the results.

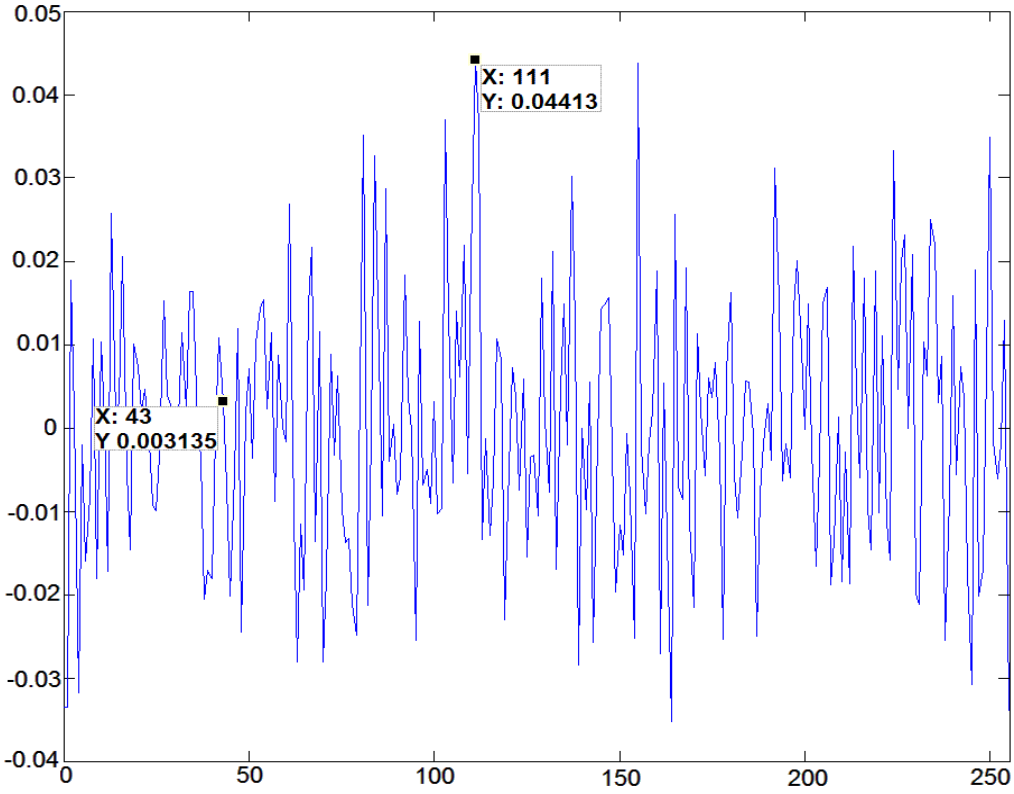


Figure 7-9: Graph of the correlation for each key guess when attacking the first byte of the sub-key of AES with a perpetually expanding key schedule after the s-box using the normal DPA algorithm with 4096 traces.

7.2.3.3 *Attack on a Physical System*

After the algorithm was attacked in simulation the design was implemented on a Virtex-E 1000 FPGA and the power consumption was measured while the device was performing encryptions using the setup described in section 5.3.2. After 45,000 traces none of the correct key values could be retrieved from the system. The initial value for the first byte of the key was 0x2B, or 43 in decimal notation, a graph of the correlation for each key guess is shown in Figure 7-10. The value with the highest correlation is 137 with a correlation of 0.01029; the correct value of 43 is significantly lower at -0.000841.

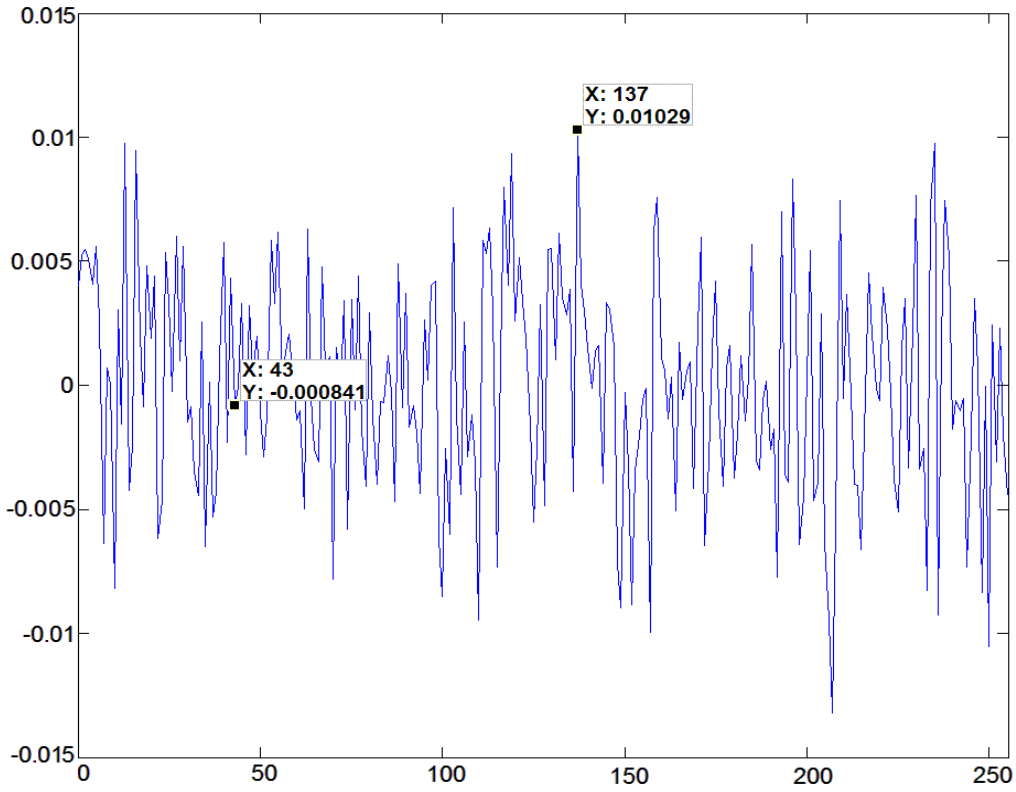


Figure 7-10: Graph of the correlation for each key guess when attacking the first byte of the sub-key of AES with a perpetually expanding key schedule after the s-box using the normal DPA algorithm with 45,000 traces.

7.2.3.4 *Conclusion*

Experimental results suggest that this algorithm is resistant to DPA. This is because the DPA algorithm does not really apply to the modified algorithm as it is not possible to make any meaningful predictions about the contents of registers as the assumption that the sub-key byte that is being targeted is constant is no longer valid. As there are no hardware countermeasures implemented, if a new prediction formula could be derived that got around the changing sub-keys this design would be just as susceptible to DPA as any other.

Due to the way the AES key schedule works after the generation of each new sub-key the value of each byte of the new key is determined by the value of an additional byte of the original key. This means that, for a 128-bit key, after 16 round, less than two encryption blocks, each byte of the sub-key is dependent on every byte in the original key. Making predictions about values inside registers now involves assumptions about the entire key and so this method would offer no advantage over brute force.

7.2.4 Summary of Results

Algorithm	Pre S-Box	Post S-Box
AES	742	345
Strengthened K.S.	761	291
Strengthened K.S. 2nd Byte	999	332
Initial Diffusion	4,096+	4,096+
Initial Diff., (some knowledge of key)	674	-
Perpetually Expanding K.S.	4,096+	4,096+

Table 7-2: Summary of results for DPA attacks on Modelsim simulations of AES.

The results of the attacks on the Modelsim simulations for various versions of AES are summarised in Table 7-2. Strengthening the key schedule using the technique proposed by May *et al* [30] does not add significantly more security in terms of the number of traces that are required for complete key retrieval, although as the other round keys cannot be derived from the first all must be extracted using DPA increasing the overall computation time by a factor of 16.

Adding another level of diffusion after adding the first key does offer some frustration to an attacker. Attempting unmodified DPA on a system running this algorithm did not reveal the key. With knowledge of three bytes of the key in a column the attacker could use minimally modified DPA to retrieve the 4th with approximately the same level of effort as regular DPA. There are some other practical issues when attempting to implement such a system, there must be no registers storing the intermediate data after the initial Add Key, as this would be a potential target for the attacker that completely bypasses the countermeasure. This makes it unsuitable for software implementations. Additionally, with sensitive enough measurements, it may be possible to target logic gates within the additional Mix Columns operation again bypassing the effects of the countermeasure completely

The results of the tests on the perpetually expanding key schedule indicate that it is effective at preventing an attacker from retrieving the key.

7.3 TDES

TDES became the de facto replacement to DES after it was recognised that the security that DES provided was no longer sufficient. It is slowly being replaced by AES as security systems are upgraded, one area where it remains in widespread use it in the electronic payments industry, for example EMV [102], more commonly known as Chip and Pin. Section 7.3.1.1 reports the results from a simulated DPA attack on TDES. Section 7.3.2 discusses the application of the algorithm modification described in section 7.2.3 to TDES, and the effect on the size and throughput of the implementation.

7.3.1.1 *Attack on TDES*

In order to verify the efficacy of the countermeasure compared to normal TDES DPA was performed on an unprotected version of the algorithm. First a VHDL implementation of TDES was downloaded from opencores.org [103], this was then synthesised for a Xilinx Virtex-E 1000 and simulated in Modelsim and the register transitions were recorded using the same method as described in section 5.3.1.1.

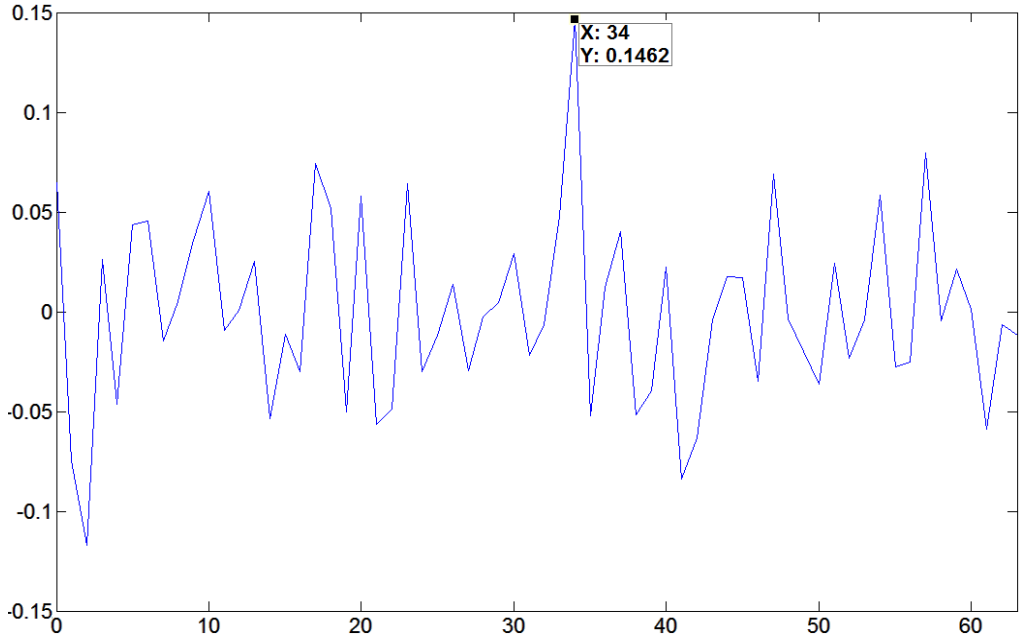


Figure 7-11: The correlation for each key guess in the first key section in the first DES block of TDES with 1,000 traces.

The entire key could be retrieved with 1,640 traces. This is more than is required for the AES implementations. The sections of the key that are individually targeted are only 6 bits long and in each DES block there are 48-bit keys, giving a

total of 144 key bits for TDES all influencing the number of register bits that are changing in the target cock cycle. This means that there is a lower signal-to-noise ratio for the TDES simulation than in AES. A graph of the correlation between the prediction and consumption matrices for the first key section for a DPA with 1,000 traces is shown in Figure 7-11. The actual value of the first key section was 0x22, or 34, and the value retrieved from the DPA was also 34 meaning the correct key had been retrieved. Although 1,000 traces was not enough to extract all eight keys in the experiment it was enough to get six and Figure 7-11 shows a clear peak compared to the noise.

7.3.2 Modified TDES

The DES key schedule forms the various round keys by bit shifting the master key and selecting specific bits from it. This can only give a limited number of possibilities for the key so the technique of continuing the key expansion cannot be directly applied to TDES. For this reason the key schedule in each DES block was replaced by the AES key schedule with a couple of modifications. Firstly, the AES key schedule has four s-boxes that substitute the values inside a 32-bit vector, these are replaced by the expansion function, which converts the 32-bit vector to a 48-bit one, and the eight DES s-boxes, each having a 6-bit input and a 4-bit output returning the data to 32 bits. Obviously, it was allowed to continue expanding for each successive block as described in section 7.2.3.

In order to ensure backwards compatibility with DES, TDES is often used in EDE (encryption-decryption-encryption) mode, as supplying the same key to all blocks gives the same result as supplying the same key to a single DES block. Due to its popularity this was the version that was implemented. This meant that the ability to perform decryption was required by at least one of the DES blocks, this requires additional resources as unlike the original DES key schedule, the AES one can only generate the round keys forwards, so when they are needed in reverse order for decryption then need to be stored. In the standard algorithms the round keys do not change so they are calculated once at the start and stored. This is only a small delay of a few clock cycles for each key. With the constantly expanding approach this time penalty is applied to every block so it becomes more significant. In order to overcome this enough storage is added for two sets of round keys and while the first set is being

used the second set, for the next block, it being calculated. This does however require more space.

The AES key schedule works on blocks of 128 bits, the TDES key is not that long the system must be modified to take this into account. There are several solutions to this, one would be padding the keys, or repeating them or increasing the size of the key space to 384 bits. The solution that was chosen was increasing the size of the 3 DES keys to 64 bits and combining these in different ways to form the 128-bit keys as shown in Table 7-3. These changes make no difference to the internal working of the algorithm, the only effect is that it is supplied with a different set of round keys.

TDES Key Number	Key Formation
1	DES Key 1 + DES Key 3
2	DES Key 2 + DES Key 1
3	DES Key 3 + DES Key 2

Table 7-3 : Arrangemet of keys for the Modified TDES.

7.3.3 Effect on Efficiency

The modified TDES design was synthesised for a Xilinx Virtex-E 1000, as was the original TDES. The size requirements and the clock speed for the two designs are summarised in Table 7-4. There is a significantly greater penalty for the alterations than in AES, the number of slices required increases by a factor of nearly 6. The DES key schedule is simple to implement, it is just a series of bit shifts and permutations, the AES key schedule is significantly more complicated, involving substitutions and XOR operations, and has the additional disadvantage of requiring the storage of round keys for decryption. Also this penalty applies to each DES block so the effect is tripled. The design increases by so much that the modified TDES is larger than the modified AES, which only required 5,616 slices. The clock speed of the design does not change significantly however. This is because the critical path for the orignal design included both the key schedule and the datapath, the datapth being responsible for nearly half of the delay. The modification to the algorithm required the addition of registers between the two sections so although the delay for the key schedule was more than doubled this was offset by the breaking up of the critical path.

	TDES	Modified TDES
Slices	1,201	6,940
DFFs	1,215	6,323
Clock Speed (MHz)	49.478	43.537
Cycles / Plaintext	19	19
Throughput (MB/s)	166.66 MB/s	146.65 MB/s

Table 7-4 : Summary of speed and area requirements for the standard and modified TDES when synthesised to a Virtex-E 1000.

7.3.4 Attack on a Simulated System

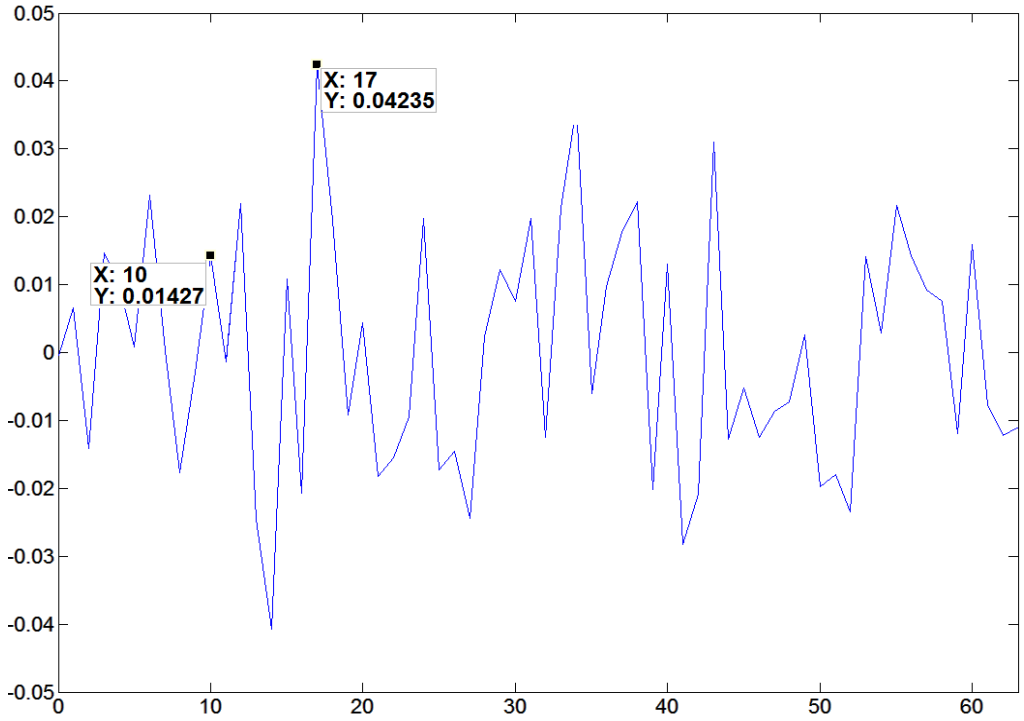


Figure 7-12: Correlation for all 64 key guesses for an attack on a Modelsim simulation of a modified TDES system with 4,096 traces.

The modified TDES was synthesised for a Xilinx Virtex-E 1000 and simulated in Modelsim and the register transitions were recorded using the same method as described in section 5.3.1.1. The correlation between the prediction and consumption matrices for all 64 key guesses using 4,096 traces is shown in. The actual value of the

first key section was 10, and the value retrieved from the DPA was 17, with a correlation nearly 4 times higher, meaning the correct key had not been retrieved.

7.3.5 Conclusion

The technique of continually changing the round keys between block can also be adapted to protect TDES from DPA. There are however some properties of algorithm that mean the implementational penalties for doing this are greater. DES, and by extension TDES, has a very simple key schedule, it consumes little area but can only generate a limited number of round keys. This limitation meant that for the purpose of the modification it had to be replaced with a more complex one. As the original DES key schedule was so compact the modification significantly increased the size of each block and because there are three DES blocks in TDES the area penalties were tripled. The increase in cost is so great that after the technique has been applied the TDES design is larger than the modified AES system.

While this is a powerful technique for making algorithms immune to DPA it is not necessarily appropriate to retrofit all current algorithms using this method. Section 7.4 outlines the requirements it to be implemented in an efficient way and investigates several modern algorithms in order to identify the most appropriate design for perpetually expanding key schedules.

7.4 Application of Perpetual Key Schedule to Other Algorithms

Using a key schedule that continually generates different round keys for successive encryption blocks to defeat DPA generally does not have a high overhead compared to an implementation of the same algorithm with a normal key schedule. There are some limits on when it can be used, the key schedule has to be complex enough to generate a large sequence of bits, the DES, and hence TDES, key schedule cannot do this so when TDES was modified to be secure against DPA the key schedule was replaced by the one from AES. This greatly increased the hardware and performance overhead compared to standard TDES.

When the new algorithm was implemented as an encryptor there was a significantly smaller overhead than when the implementation could also perform decryption. This was due to the fact that during decryption the round keys are needed

in reverse order, but they can only be generated forwards. This meant that either a new set of round keys had to be generated for each block before decryption could take place, or additional memory was required to store the results of the calculation of the next set while the current set was being used. Algorithms that can generate the round keys in any order would not suffer from this problem, further reducing the overhead for the technique. This section evaluates a number of modern encryption algorithms for their suitability for this countermeasure and identifies the properties of key schedules that give the best performance in terms of the relative cost of changing the key schedule. Section 7.4.1 describes the key schedules of the algorithms that are being investigated and section 7.4.2 discusses the advantages and disadvantages of the various properties the different key schedules have.

7.4.1 Key Schedules of Modern Algorithms

The following algorithms were selected as AES finalists or are recommended algorithms for either NESSIE, (New European Schemes for Signatures, Integrity and Encryption) a European project to identify secure cryptographic algorithms, or CRYPTREC, an equivalent project set up by the Japanese government.

7.4.1.1 *MARS*

MARS [104] was developed by IBM in 1998, the design team included Don Coppersmith, who also helped design DES. MARS was a finalist in the AES process, and as such works on a block size of 128 bits, processing the data in 32-bit words, and supports variable key lengths, from 128 bits to 448. It has a Feistel structure and is 20 rounds long and so requires 40 words, or 1,280 bits, of expanded key data.

7.4.1.1.1 Key schedule description

The master key is placed into a table of 15 32-bit words called *T*, the key will always be shorter than this so it is concatenated with a binary value of its length in words and then padded with 0s. The array *T* is then put through four rounds of the following transformations:

for $j = 0$ to 3

for $i = 0$ to 14

$$T_i = T_i \oplus ((T_{i-7 \bmod 15} \oplus T_{i-2 \bmod 15}) <<< 3) \oplus (4i + j)$$

for $k = 0$ to 3

for $i = 0$ to 14

$T_i = T_i \oplus S$ (Low 9 bits of $T_{\text{i-1 mod 15}}$)<<<9

for $i = 0$ to 9

$$K_{10j+i} = T_{4i \bmod 15}$$

Figure 7-13: Pseudo-code for the key schedule of the MARS algorithm.

Where S is MARS's s-box. Finally, to ensure that none of the key words that are involved in multiplication, $K_{5,\ 7,\ \ldots\ 35}$, in the algorithm have certain properties, namely that the lowest two bits are set to 1 and there are no groups of ten consecutive 1s or 0s the following operation is performed.

The lowest two bits of K_i are recorded and set to 1. If patterns of more than ten consecutive 1s or 0s are detected in K_i then the runs are XORed with bits from entries 265 through to 268 of the s-box, the particular entry being selected by the original value of the two lowest bits in K_i, and rotated by a number of bits selected using the value of the five lowest bits in $K_{i\text{-}1}$. In the final section $i = 5, 7, \ldots 35$.

7.4.1.1.2 Analysis

The key schedule of MARS could be modified to use a perpetually expanding key schedule, the use of the s-box and significant bit mixing between sub key bytes ensures a complex enough relationship and the last 480 bits of the expanded key could be used as a new input.

While the key schedule uses the same s-boxes as the datapath the structure is significantly different and so hardware specifically to expand the key will have to be included. It will always be possible to generate round keys in parallel with the main datapath. In decryption keys are needed in reverse order so have to be pre-generated and stored. The key expansion for MARS is quite complex, requiring multiple clock cycles, or significant unrolling of loops, but the amount of expanded key data is a

modest 1,280 bits. Due to this, double buffering the key schedule as described in section 7.2.3 will not have as large an area penalty as for other algorithms.

7.4.1.2 *RC6*

RC6 [105] was published in 1998 by Ron Rivest, Matt Robshaw, Ray Sidney, and Yiqun Lisa Yin. It was submitted as a candidate for AES and it was a finalist. RC6 is a Feistel network that acts on variable blocks sizes, number of rounds and key lengths. Technically the algorithm is specified as RC6-*w*/*r*/*b* where *w* is the data word size, *r* is the number of rounds and *b* is the number of bytes in the key. The version that shall be discussed here is the one that was submitted as a candidate for AES, so acts on block sizes of 128 bits, four 32-bit words, supporting key lengths of 128, 192 and 256 and consists of 20 rounds. RC6 requires 2 (r + 1) words of expanded key data. This variant of RC6 requires a total of 1,344 bits of expanded key data. The designers have largely recycled the key schedule from the 1995 RC5 algorithm.

7.4.1.2.1 Key schedule description

The key schedule of RC5, and hence RC6 initialises the expanded key space with two "magic constants", *P* and *Q*, which are both odd and of length *w*, and are derived from the hexadecimal representation of Euler's constant and the Golden Ratio [106]. The first word in the expanded key, *S*, is set to *P* and each successive word is set to the previous word + *Q*, where the addition is performed modulo-2^w. Next the master key is copied to the array *L* and mixed in with the pseudo random bit streams in the following way:

$A = B = 0$

$i = j = 0$

Repeat (3 * # words in *S)*

$A = S_i = (S_i + A + B) <<< 3) \oplus (4i + j)$

$B = L_j = (L_j + A + B) <<< (A + B)$

$i = (i + 1)$ mod # words in S

$j = (j + 1)$ mod # words in L

Figure 7-14: Pseudcode for the bit mixing of the key schedule of RC6.

7.4.1.2.2 Analysis

RC6 can be protected from DPA by taking the final 128, 192 or 256 bits of the expanded key, copying it into the array *L* and repeating the key expansion process. This key schedule cannot generate the sub-keys in any order. In AES a given round key depends only on the round key before it, in RC6 the sub-keys are generated with a three stage process so cannot be derived in an online fashion. This means that the expanded key would probably be pre-computed and stored, as the only other option would be to replicate hardware to unroll the loop which would be expensive. The offline design makes the maximum area penalty slightly less for the comparison between the original algorithm and a modified version as there must always be enough memory to store the entire expanded key. As the key schedule does not use the cryptographic primitives from the main datapath there is little area penalty incurred from providing the hardware to enable the new encryption key in parallel with the encryption of the previous plaintext. This would require the addition of another 1,344 bits of memory.

7.4.1.3 *Serpent*

Serpent was published in 1998 [107] by Ross Anderson, Eli Biham, and Lars Knudsen, it was submitted as a potential algorithm for AES and was a finalist. Serpent is a 32 round SPN and like the other AES submissions it works on 128-bit blocks and supports key lengths of 128, 192 and 256 bits. In the final round an additional round key is also used bringing the total amount of expanded key data up to 33 round keys or 4,224 bits.

7.4.1.3.1 Key schedule description

The master key is padded to 256 bits by adding a 1 followed by as many 0s as is required and split un into eight 32-bit words labelled w_{-8} to w_{-1}. These are then expanded into 132 intermediate keys w_0 to w_{131} using the following relationship:

$$w_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \phi \oplus i) <<< 11 \tag{7-1}$$

Where ϕ has the value 0x9e3779b9, the fractional part of the golden ratio in hexadecimal. The intermediate keys are then converted into the final round keys by passing them through Serpent's s-boxes. Serpent has eight different 4-bit to 4-bit s-boxes. In order to increase parallelisation the key schedule, as well as the rest of the

algorithm, was designed using a bit slice technique. The intermediate key word is substituted not as a series of 4–bit words, but a single bit from four consecutive intermediate words is put into an s-box and the final round keys are made of the corresponding s-box output bits.

7.4.1.3.2 Analysis

If the key schedule of Serpent was modified so that the expansion of the master key into *w* did not stop, but continued, generating 132 intermediate keys, for conversion into round keys, for each plaintext that needed to be encrypted then the algorithm could be protected from DPA.

Serpent requires a large number of expanded key bits so pre-calculating the expanded key and storing it would require a lot of RAM, making an offline key schedule expensive in terms of area. Conversely, generating one round key requires the same number of s-boxes as processing one round so calculating the round keys in parallel would require doubling the number of s-boxes. S-boxes are a significant factor in the size of most hardware implementations of cryptographic algorithms, so doubling the number would have a serious area penalty. There is a compromise of interleaving round key generation and one round of encryption on alternating clock cycles, this would only require storing the 12 32-bit words that the current round key is based on. This is true for the original algorithm as well as a modified version, and the cost of modifying the algorithm can only be compared to an implementation of the original.

For an encryptor that alternated between generating round keys and performing encryption to reduce the area of the design there would not be a significant cost in terms of area or speed for modifying the algorithm, the round keys would have to be generated for each plaintext anyway and no additional memory would be needed. If the round keys were pre-calculated and stored the amount of time required to process a plaintext would double as the keys would still have to be calculated for each plaintext.

As a decryptor has to provide the round keys in reverse order but can only generate them in the actual order any design that can decrypt would need enough memory to store all the round keys and would pre-calculate them. After modification the algorithm would need to generate a new set of keys for each plaintext so again the

amount of time taken to process a plaintext would double. This could be mitigated with the same double buffering technique that was described in section 7.2.3, but this would require doubling the, already significant, amount of memory for storing round keys and also adding another 16 s-boxes so the next set of keys could be calculated which the current plaintext was being processed.

7.4.1.4 *MISTY 1*

MISTY was developed by Mitsuru Matsui in 1995 [108]. It uses blocks of 64 bits and keys of 128 bits, it is a recommended algorithm for both NESSIE and CRYPTREC. It uses a nested Feistel structure, where each round is made up of a three round Feistel structure with a 32-bit datapath, with each sub-round being composed of a smaller three round 16-bit Feistel structure where the two halves are split into 7 and 9 bits. MISTY can have any number of rounds on the condition that it is divisible by four. MISTY1 only requires 256 bits of expanded key data.

7.4.1.4.1 Key schedule description

As MISTY1 does not require to expand the key very much, only to double the amount of key data, the key schedule is fairly simple. The master key is separated into eight 16-bit words and these are all fed through the 16-bit round function, consisting of three rounds of the lowest level Feistel structure. The round key for each 16-bit word is the master key from the word to its right. During a set of four rounds of encryption all 16 key words are used.

7.4.1.4.2 Analysis

The key schedule for MISTY could have a perpetually expanding key schedule. The second half of the expanded key could be used as the new master key, the round function is complex enough to remove the possibility if short repeating patterns appearing in the expanded keys. As the expansion of the key involves putting one word of the key into the round function using another master key word as the round key there is mixing between key words. After eight encryption blocks the value of each block of the key would be affected by the value of each bit in the original master key.

The keys in MISTY can be generated in any order, normally this would enable an online key schedule to be used for both encryption and decryption, MISTY is a

little different. Due to the small nature of the expanded key, the fact that the entire expanded key is required in the first four top level rounds and the reuse of the second level round function to generate the expanded key the more efficient design would still be offline. Very little memory would be required to store the entire key and this would be traded off against the area cost of replicating the 16-bit round function. The speed penalty for pre-expanding the key would only have to be paid once for the original algorithm, but it would be incurred once encryption for the modified one.

The size of this cost as a proportion of the total run time is dependent on the number of rounds. Each top level round contains three of the second level rounds that are used to expand the key and 8 key expansions are required. Expanding the key with whatever hardware there is in the system will require the same amount of time as 8/3 top level rounds, if there are only four rounds this is a 66% decrease in throughput, 33% for 8 round and so on. There is very little increase in area, no additional storage is needed, but a more complex controller will be required.

7.4.1.5 *Camellia*

Camellia was developed by Mitsubishi in 2000 [109]. It has a block size of 128 bits and supports key lengths of 128, 192 and 256 bits. It was selected as a recommended algorithm for both CRYPTREC and NESSIE. It uses a Feistel structure and consists of 18 rounds when a 128-bit key is being used and 24 for a 192 and 256-bit key. There is also additional key mixing at the start and end of the algorithm and every six rounds two round keys are used this bring the total number of 64-bit round keys required by the algorithm to 26 for the 128-bit key and 34 for 192 and 256-bit keys.

7.4.1.5.1 Key schedule description

The key schedule of Camellia uses the same cryptographic primitives as the main algorithm to expand the master key into the round keys. The master key is separated into two 128-bit blocks K_L and K_R. In the case of the 128-bit key length K_R is set to 0, in the case of the 192 bit key the right hand side of the K_R is set to the compliment of the left hand side. K_L is XORed with K_R and this is then encrypted for four rounds with a constant set of round keys, the new value is K_A. For key lengths greater than 128 bits K_A is then XORed with K_R and encrypted for another two rounds

giving K_B. The final values of the 64-bit round keys are then selected from K_L and K_A for the 128-bit key and from all four values for the 192 and 256-bit keys.

7.4.1.5.2 Analysis

The Camellia key schedule can be modified to prevent DPA by setting K_R and K_L to K_A and K_B, generating a new K_A and K_B and deriving the next set of round key from these values. The cipher reuses the round function to expand the key, so a new set of round keys could not be calculated in parallel without almost doubling the size of a design, this is not a significant problem for the original algorithm as the keys only have to be calculated once and the amount of RAM needed to store them is small as they are derived from four 128-bit values. For the modified version these values need to be re-initialised for every plaintext, adding six clock cycles for every 18, an increase of 33%. The round keys can be generated in any order meaning that there is a greater consistency in the performance of a design that can decrypt compared to one that can only encrypt.

7.4.1.6 *Hierocrypt-3*

Hierocrypt-3 was developed by Toshiba in 2000 [110]. The structure of Hierocrypt-3 is a 16 round nested SPN in which a higher level s-box is itself a smaller SPN. It works on blocks of 128 bits using keys of length 128, 192 and 256 bits which have 6, 7 or 8 rounds respectively. Each round requires two 128-bit round keys and there is one final key addition, therefore a total of 1,664, 1,920 or 2,176 bits of expanded key data are required for the three supported key lengths. It is a recommended cipher from the CRYPTREC program.

7.4.1.6.1 Key schedule description

The master key is padded to 256 bits with a series of 32-bit constants derived from the binary representations of irrational numbers and converted to the first intermediate key with the σ_0 function. The function σ iteratively generates the first four intermediate keys (Z) for 128 / 192-bit keys and 5 for 256-bit keys. One intermediate key is required for each round and the remaining ones are generated by the σ^{-1} function. Figure 7-15 shows the structure of the functions σ_0, σ and σ^{-1}.

In Figure 7-15 K denotes the actual round keys that are derived from the intermediate keys Z, T is the total number of rounds. G denotes the round constants

prevent periodic patterns appearing the intermediate keys and, like the key padding constants, are based on the binary representations of irrational numbers. *P* is a linear permutation that separates the data into four blocks and XORs each block with one other. M_{5E} and M_{B3} are both similar to *P* except they separate the data into two distinct groups of four blocks and XORs data within the groups. The function F_σ separates the data into 8 bytes, passes them through Heirocrypt-3's s-box and then applies the *P* function.

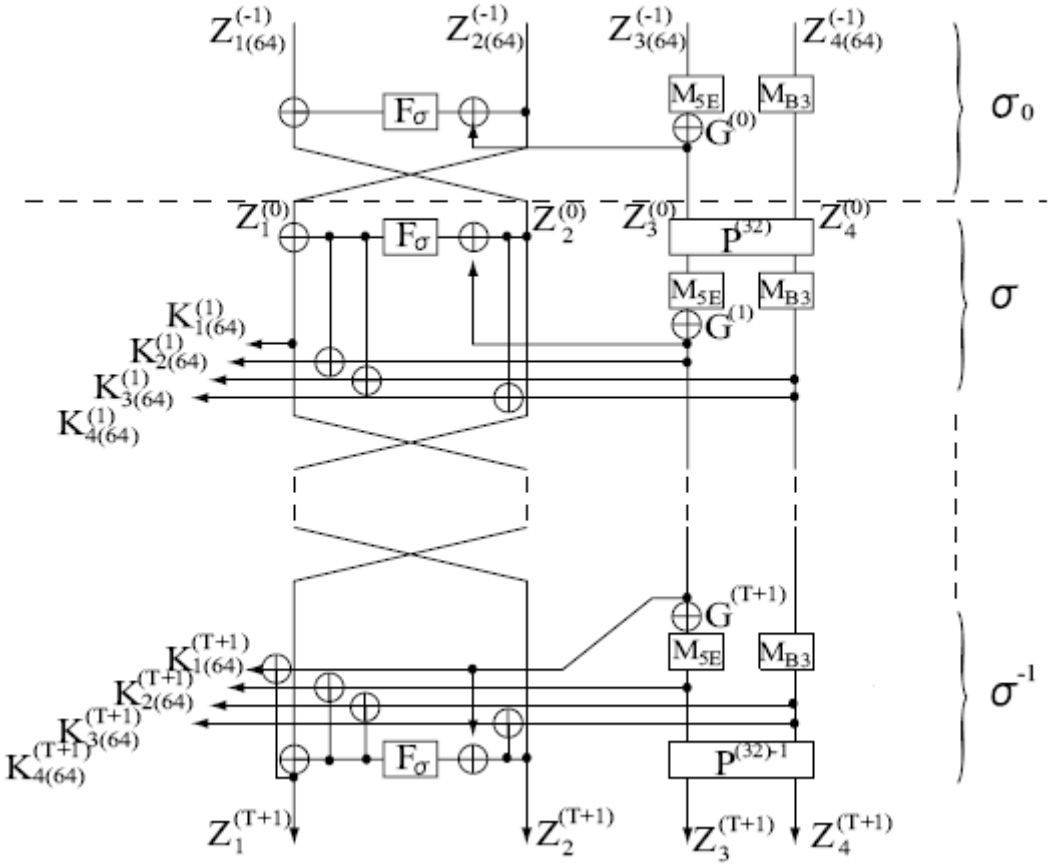


Figure 7-15: The structure of the Heirocrypt-3 key schedule.

7.4.1.6.2 Analysis

The key schedule for Heirocrypt-3 is fairly similar to that of AES. It is an iterative key schedule, a function is applied to the master key to generate a set of intermediate values from which the first round key is derived, and one of two related function are repeatedly applied to the previous intermediate data to provide the intermediate data for the next set of round keys. The round key iteration functions are complex and involve a large degree of bit mixing between sub-key words ensuring the appropriate properties for a perpetual key schedule design.

The key schedule design being similar in structure to that of AES has the same problems. Assuming an online key schedule has been used, altering the encryption algorithm will require very little modification, simply looping the key schedule round. While the key schedule uses the s-box from the datapath it does not use any of the other blocks. This means that the hardware for the key schedule will have to be included irrespective of the style of key schedule that is used so there will always be the possibility of calculating round keys in parallel with the main datapath. For decryption the keys will be needed in reverse order so they will have to be pre-generated and stored. Using the approaches discussed in section 7.2.2.1 it is possible to trade off costs in speed and area by double buffering the memory for the offline key schedule, Heirocrypt-3 required between 1,664 and 2,176 bits of expanded key data.

7.4.1.7 *ARIA*

ARIA is a 128-bit SPN based block cipher that uses 128, 192 and 256-bit keys [111], it was designed in 2003 by cryptographers in South Korea and in 2004 it was selected by the Korean Agency for Technology and Standards to be a standard cryptographic algorithm. ARIA has 12, 14 or 16 rounds depending on the key size that is used and it requires 128 bits of expanded key data per round, with an additional 128 bits for a final key addition. In total 1,664, 1,920 or 2,176 bits are required.

7.4.1.7.1 Key schedule description

The key schedule in the ARIA algorithm has two phases, initialisation and expansion. The initialisation phase uses the master key and three 128-bit constants to generate four 128-bit sub-keys using a 256-bit Feistel cipher. The left half of the Feistel data is the first 128 bits of the master key, the right half is any unused bits of the key padded with zeros to 128 bits. The Feistel cipher is used to generate four 128-bit values using the odd and even round functions F_e and F_o, who differ due to the fact that ARIA uses two different s-boxes and their inverses in a different order on alternating rounds.

$$W_0 = KL \tag{7-2}$$

$$W_1 = F_o\,(W0,\ CK1) \oplus KR \tag{7-3}$$

$$W_2 = F_e\,(W1,\ CK2) \oplus W_0 \tag{7-4}$$

$$W_3 = F_o\,(W2,\ CK3) \oplus W_1 \tag{7-5}$$

From these values the round keys are generated. This is done by XORing one of the values with another after it has been rotated by a certain number of bits. The choice of the two values and the amount of rotation are determined by the round.

7.4.1.7.2 Analysis

ARIA can be modified for DPA resistance by continuing the initialisation Feistel cipher to generate four new sub-keys for every plaintext that must be encrypted. After the initialisation stage of the key schedule the round keys can be generated in any order so no additional area or speed penalties are paid when implementing a system that can perform decryption. As the key initialisation phase uses the round function of the encryption datapath it could not be performed in parallel without essentially doubling the area requirements of a design, encryption would have to be temporarily stopped in order to re-initialise the key, increasing the amount of clock cycles per processed plaintext by between 25% and 33% depending on the key length.

7.4.2 Application of Perpetual Key Schedule to Other Algorithms

Several of the algorithms described in section 7.4.1 have similar designs, Hierocrypt-3, Serpent, MARS and RC6 iteratively apply a function to the key data, the output of which is both a round key and the input for the function to generate the next key. Camellia and ARIA use a two stage process, the first stage takes the master key and generates four intermediate keys, in the key generation stage these are combined in a variety of different ways to form the round keys. All of the key schedules reuse some of the cryptographic primitives that are found in the main datapath, Camellia, ARIA and MISTY-1 use the entire round function for key expansion. This section discusses the advantages and disadvantages of the key schedule structures with respect to modifying algorithms so they have a key schedule that protects them from DPA.

7.4.2.1 *Initialisation vs. Iterative*

There are two main approaches to generating round keys in the algorithms that have been discussed in section 7.4.1. Firstly there is the iterative approach where a function is applied to a block of expanded key data with a fixed length, the output of

which is a new round key which forms all or part of the input to the next iteration. Examples of these are Hierocrypt-3 and Serpent, and MARS and RC6, which repeat the iterative process across a number of rounds. The second approach uses an initialisation phase, where a, typically more complex, transform converts the master key into a relatively small set of intermediate keys, the bits of which are combined in a number of different ways to generate the various round keys. This is the technique used in Camellia and ARIA. The exception to this is the algorithm MISTY 1, which needs such a small amount of expanded key data that it simply applies the round function to the master key in order to double the amount of key data available.

The advantage of the initialisation approach is that once the intermediate keys have been determined the round keys can be generated in any order. This is a significant advantage when performing decryption and it makes the performance of encryptors and decryptors a lot more consistent.

7.4.2.2 *Reuse of Cryptographic Primitives*

Lots of the algorithms have key schedules that make some use of the cryptographic primitives to expand the key. All of the ones described in section 7.4.1 make use of their s-box, except RC6, which doesn't use an s-box. Hierocrypt-3 also reuses some of the datapath functions for permuting bits and ARIA, Camellia and MISTY 1 reuse the entire round function.

There are several advantages to reusing the entire round function. The main one is that hardware can also be reused making the design potentially much smaller and simplifying the implementation process as less has to be designed. Also, as noted by May *et al.* in [30] key schedules designed in an *ad hoc* fashion tend to perform relatively poorly in terms of the confusion and diffusion properties of the expanded key. By reusing the round function assuming the cipher performs well in these areas the key schedule will also. Having good confusion and diffusion performance will also reduce the need for time consuming complex multi-round key expansion algorithms like those in MARS and RC6.

Reusing the entire round function to generate the key schedule is somewhat of a double edged sword, in order to make a design that can calculate expand the key schedule and perform the encryption in parallel the area requirements are almost doubled, whereas if the hardware to expand the key has to be implemented anyway

that automatically means it can be run in parallel. This is less of a problem with a traditional algorithm that only has to generate one set of round keys as they can be pre-calculated and stored, but when new round keys have to be generated for each plaintext the cost to processing time is incurred for each plaintext. This can be less of a disadvantage if the key schedule uses an initialisation approach to generating round keys as less will have to be calculated. In ARIA and Camellia only four intermediate keys were needed to generate all the round keys.

7.4.3 Conclusion

The majority of modern algorithms have complex enough key schedules to produce long enough cycles for perpetually expanding keys to be applied to be a practical countermeasure to DPA. The design and structure of the key schedule clearly has a vast affect on the performance cost of the modification of the algorithm in terms of speed and area relative to the original. As a new set of keys must be calculated for each plaintext having to pre-calculate the expanded key would greatly reduce the speed of encryption. For a normal algorithm it would have a much lower effect on the average throughput as the amount of time spent processing the key will be insignificant compared to the time encrypting all of the plaintexts. The algorithms described in section 7.4.1 that can generate the round keys in any order do so by using a two stage process, with a short initialisation phase that generates values which are then combined into the round keys.

In order to save time it is advantageous to be able to generate keys in parallel with the main datapath. Conversely in order to make designs smaller it is advantageous to reuse the hardware from the datapath to also generate the round keys. Reuse of cryptographic primitives also ensures that round key generation adequately satisfies the confusion diffusion requirements.

7.5 Case Study: ARIA

Of all of the various key schedules in the algorithms described in section 7.4 designs that use an initialisation phase to generate a relatively small number of values that are then combined in various ways to generate the round keys appear to be best suited to being protected from DPA with a modified key schedule. In this section the

algorithm ARIA is modified and the DPA resistance and performance of the new version is evaluated.

In order to test the effect the modification had on the speed and area requirements and to verify the DPA resistance of the new version of the algorithm four implementations were designed. Two versions were the original ARIA, one that could only encrypt and one that could both encrypt and decrypt. For each of those a counterpart implementation was created that used the new key schedule, which is described in detail in section 7.5.1. For simplicity all of the versions only used 128-bit keys. The VHDL designs were then synthesised for a Virtex-E 1000 bg560 and the area and timing requirements were noted, these are discussed in section 7.5.2. Finally using the post-synthesis VHDL and the method described in section 5.3.1.1 simulated DPA was performed on the two encryptor designs, the results and analysis of this are in section 7.5.3.

7.5.1 Design of Key Schedule

The original ARIA key schedule generates four 128-bit intermediate keys. This is achieved by taking the master key and putting it through a Feistel cipher made from the round function of the main datapath. The details of precisely how to generate the four intermediate keys are given in equations (7-6) - (7-9) where R represents the round function and k is the key used for the process, it is derived from the binary representation of $1/\pi$. MK is the master key that can have a length of 128, 192 or 256 bits. The right hand 128 bits is used in the generation of w_1, if the key is not that long then it is padded with 0s.

$$w_0 = MK_{0-127} \tag{7-6}$$

$$w_1 = R(w_0, k_0) \oplus MK_{128-255} \tag{7-7}$$

$$w_2 = R(w_1, k_1) \oplus w_0 \tag{7-8}$$

$$w_3 = R(w_2, k_2) \oplus w_1 \tag{7-9}$$

These round keys are made by XORing two intermediate keys after a rotation has been applied to one, as shown in Table 7-5

Key	Formation	Key	Formation	Key	Formation
1	$w_0 \oplus (w_1 >>> 19)$	7	$w_2 \oplus (w_3 >>> 31)$	13	$w_0 \oplus (w_1 >>> 97)$
2	$w_1 \oplus (w_2 >>> 19)$	8	$w_3 \oplus (w_0 >>> 31)$	14	$w_1 \oplus (w_2 >>> 97)$
3	$w_2 \oplus (w_3 >>> 19)$	9	$w_0 \oplus (w_1 >>> 67)$	15	$w_2 \oplus (w_3 >>> 97)$
4	$w_3 \oplus (w_0 >>> 19)$	10	$w_1 \oplus (w_2 >>> 67)$	16	$w_3 \oplus (w_0 >>> 97)$
5	$w_0 \oplus (w_1 >>> 31)$	11	$w_2 \oplus (w_3 >>> 67)$	17	$w_0 \oplus (w_1 >>> 109)$
6	$w_1 \oplus (w_2 >>> 31)$	12	$w_3 \oplus (w_0 >>> 67)$		

Table 7-5 definitions of the round keys for AIRA

The basic concept of the new algorithm is to continue this process in order to generate four new intermediate keys for each plaintext. Assuming that the round function can be performing on one clock cycle, calculating the intermediate keys would take three clock cycles, also the ARIA specification has three 128-bit constants that are used as round keys for the Feistel cipher. For simplicity, in the new scheme w_3 replaces $MK_{0\text{-}127}$ and w_2 replaces $MK_{128\text{-}255}$, by doing this generating a new set of keys still only takes three clock cycles and requires the three original 128-bit constants. This is shown in equations (7-10) - (7-13). As w_0 becomes the previous version of w_3 some key data is reused, this is not a source of insecurity however as to generate the round keys two intermediate keys are combined with rotations so no round keys will be repeated.

$$w_0 = w_3 \tag{7-10}$$

$$w_1 = R(w_0, k_0) \oplus w_2 \tag{7-11}$$

$$w_2 = R(w_1, k_1) \oplus w_0 \tag{7-12}$$

$$w_3 = R(w_2, k_2) \oplus w_1 \tag{7-13}$$

7.5.2 Efficiency of Implementation

Four different implementations of ARIA were produced, one that could only encrypt, one that could both encrypt and decrypt, and equivalent versions of the algorithm when modified to be resistant to DPA. There is very little difference in the overall performance of the hardware. All designs use approximately the same number of flip flops, this was expected as they all have the same memory requirements, the slightly larger increase in the Modified Encryptor is due to signal duplicated by the synthesis tool. The area in general does not change significantly, there is an increase in the number of slices for the modified algorithm and the designs that can perform decryption of between 11% and 16%.

The clock speed of the design does not change significantly, it actually slightly increases for the modified versions. The largest penalty that is incurred from changing the algorithm is the number of clock cycles used to process a plaintext. It rises from 12 to 15, this is because a new set of sub-keys must be generated each time and as it uses the same hardware as the main datapath it cannot be calculated concurrently. This decreases the throughput by nearly 20%.

	ARIA Enc.	ARIA Enc/Dec	Modified Enc.	Modified Enc/Dec
Slices	2,536	2,962	2,825	2,913
DFFs	1,052	1,057	1,070	1,059
Cycles / plaintext	12	12	15	15
Clock speed	20.191 MHz	19.481 MHz	21.116 MHz	19.939 MHz
Throughput	215.37 MB/s	207.80 MB/s	180.19 MB/s	170.15 MB/s

Table 7-6 : Details of the area, clock-speed and throughput for the different versions of ARIA: with and without decryption and DPA resistance .

7.5.3 Countermeasure Efficacy

In order to verify the efficacy of the algorithm modification, first a DPA attack was attempted on the standard implementation of ARIA. This was performed using the Modelsim simulation method as described in section 5.3.1.1. The attack was successful. Figure 7-16 shows the correlation values for each of the 256 key

hypotheses for the first byte of the first round key, the value of which was 198 or 0xC6. In Figure 7-16 a large peak at the correct value is unambiguously identifiable. 679 traces were required to correctly identify all 16 round key bytes.

DPA was then attempted on the modified algorithm, the first byte of the first round key was again 198. After 4,096 traces the correct value was not identifiable. The correlation for all 256 key hypotheses are shown in Figure 7-17, the largest peak is at 224 with a value of 0.05314 while the correct value of 198 is close to zero and slightly negative. The plot gives no indication that the 198 is the correct value. The modified algorithm is not susceptible to DPA.

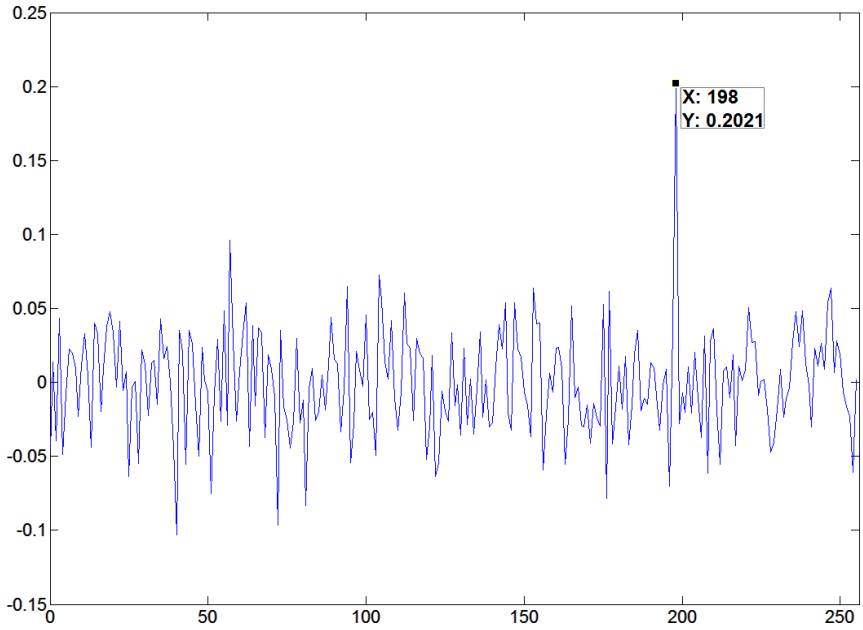


Figure 7-16: Graph showing the correlation of the 256 key guesses for a 1,000 trace DPA attack on a Modelsim simulation of an FPGA running ARIA.

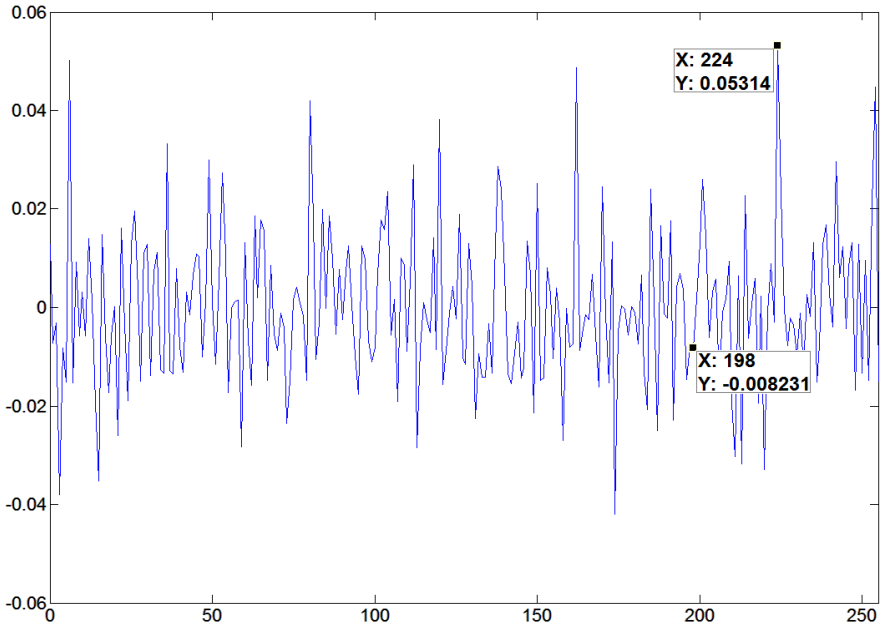


Figure 7-17 : Graph showing the correlation of the 256 key guesses for a 1,000 trace DPA attack on a Modelsim simulation of an FPGA running ARIA.

7.5.4 Conclusion

It can be seen from the results in section 7.5.3 that the modified ARIA key schedule provides protection from DPA. The protection also comes at a modest cost, for the encryptor there is approximately a 10% increase in area. There is even a slight increase in clock speed, although this is more than compensated for by the 25% increase in the number of clock cycles required to process a plaintext. It is worth noting that the percentage increase in the number of clock cycles would fall as longer keys are processed as they require more encryption rounds but the same number of key expansion rounds.

The structure of the key schedule of ARIA is much more suited to being a perpetually expanding one than that of AES as the cost in terms of speed and area is comparable between encryptors and decryptors. The key property of the key schedule that allows this is its ability to generate the round keys in any order, making pre-calculating the decryption keys unnecessary. While key initialisation occurs before encryption and decryption fewer sub-keys are needed than the number of round keys so the penalty is both consistent and less.

7.6 Improved Modified TDES

Drawing from the conclusions of examining the key schedules of other modern cryptographic algorithms and modifying the key schedule of ARIA to protect it from DPA it is possible to vastly improve the modified version of TDES described in section 7.3. That version was significantly larger than the original TDES, using nearly six times the number of slices on a FPGA. This was largely due to the additional registers that were needed to store two sets of round keys. This was needed as the round keys need to be provided in reverse order for decryption and so have to be pre-calculated. If only one set of round keys could be stored it would have doubled the amount of time taken to decrypt and one DES block is used in decryption mode during TDES encryption. Additionally as TDES contains three DES blocks any increase in area in a DES block is tripled for TDES.

Adopting a key schedule design similar to that of ARIA a much smaller modified TDES was implemented that still has inherent resistance to DPA. There is a greater throughput penalty as the initialisation phase uses the main datapath so it cannot be performed in parallel without nearly doubling the amount of hardware required. Section 7.6.1 describes the new key schedule in more detail, section 7.6.2 details the effects of the changes to the algorithm on the speed and area requirements and section 7.6.3 shows the new design is also immune to DPA.

7.6.1 Design of Key Schedule

The improved modification to the TDES key schedule calculates the round keys in two phases, initialisation and generation, each DES block uses 64-bit keys, giving a total TDES key length of 192 bits. The initialisation phase is heavily based on the key schedule of ARIA, it splits the master key into two 32-bit halves and uses the Feistel structure to expand the master key into the four sub-keys. As the initialisation phase uses the DES round function it needs round keys. The actual values of these are not particularly important, it is important that the chosen values do not insert a backdoor into that algorithm that only the designers are aware of, for this reason numbers like this are generally chosen to be binary expansions of irrational numbers. In ARIA the key initialisation round keys come from the value of $1/\pi$, these is no advantage for choosing different ones here.

$$w_0 = R(MK_{0-31}, k_0) \oplus MK_{32-63} \tag{7-14}$$

$$w_1 = R(w_0, k_1) \oplus MK_{32-63} \tag{7-15}$$

$$w_2 = R(w_1, k_2) \oplus w_0 \tag{7-16}$$

$$w_3 = R(w_2, k_3) \oplus w_1 \tag{7-17}$$

Like the ARIA modification, the key generation is extended across multiple plaintext encryptions by using sub-keys 3 and 2 to replace the first and second halves of the master key respectively, i.e. equations (7-14) and (7-15) are replaced with (7-18) and (7-19) respectively.

$$w_0 = R(w_3, k_0) \oplus w_2 \tag{7-18}$$

$$w_1 = R(w_0, k_1) \oplus w_3 \tag{7-19}$$

In DES the size of the Feistel datapath, and hence the length of the sub-keys derived from this scheme, is 32 bits, but, due to the expansion function, the size of the round keys must be 48 bits. Each round key is split into three 16-bit blocks and these are made out of the combination of two halves from two different sub keys. To ensure that all sections of the round keys are unique the sub-keys are bit shifted either left or right by 5 bits. This is summarised in table Table 7-7, where w signifies the sub-key and the a or b determining whether it is the first or second half.

Key		Formation	Key		Formation
1	0 - 15	$w_{0a} \oplus w_{1b}$	9	0 - 15	$w_{0a} \oplus (w_1)>>>5_b$
	16 - 31	$w_{1a} \oplus w_{2b}$		16 - 31	$(w_1)>>>5_a \oplus (w_2)<<<5_b$
	32 - 47	$w_{2a} \oplus w_{0b}$		32 - 47	$(w_2)<<<5_a \oplus w_{0b}$
2	0 - 15	$w_{1a} \oplus w_{3b}$	10	0 - 15	$w_{1a} \oplus (w_3)<<<5_b$
	16 - 31	$w_{2a} \oplus w_{1b}$		16 - 31	$(w_2)>>>5_a \oplus w_{1b}$
	32 - 47	$w_{3a} \oplus w_{2b}$		32 - 47	$(w_3)<<<5_a \oplus (w_2)>>>5_b$
3	0 - 15	$w_{2a} \oplus w_{3b}$	11	0 - 15	$w_{2a} \oplus (w_3)>>>5_b$
	16 - 31	$w_{3a} \oplus w_{0b}$		16 - 31	$(w_3)>>>5_a \oplus (w_0)<<<5_b$

Key		Formation	Key		Formation
	32 - 47	$w_{0a} \oplus w_{2b}$		32 - 47	$(w_0)\text{<<<}5_a \oplus w_{2b}$
4	0 - 15	$w_{3a} \oplus w_{1b}$	12	0 - 15	$w_{3a} \oplus (w_1)\text{<<<}5_b$
	16 - 31	$w_{0a} \oplus w_{3b}$		16 - 31	$(w_0)\text{>>>}5_a \oplus w_{3b}$
	32 - 47	$w_{1a} \oplus w_{0b}$		32 - 47	$(w_1)\text{<<<}5_a \oplus (w_0)\text{>>>}5_b$
5	0 - 15	$(w_0)\text{>>>}5_a \oplus (w_1)\text{<<<}5_b$	13	0 - 15	$(w_0)\text{<<<}5_a \oplus w_{1b}$
	16 - 31	$(w_1)\text{<<<}5_a \oplus w_{2b}$		16 - 31	$w_{1a} \oplus (w_2)\text{>>>}5_b$
	32 - 47	$w_{2a} \oplus (w_0)\text{>>>}5_b$		32 - 47	$(w_2)\text{>>>}5_a \oplus (w_0)\text{<<<}5_b$
6	0 - 15	$(w_1)\text{>>>}5_a \oplus (w_3)\text{<<<}5_b$	14	0 - 15	$(w_1)\text{<<<}5_a \oplus (w_3)\text{>>>}5_b$
	16 - 31	$(w_2)\text{<<<}5_a \oplus w_{1b}$		16 - 31	$w_{2a} \oplus (w_1)\text{<<<}5_b$
	32 - 47	$w_{3a} \oplus (w_2)\text{>>>}5_b$		32 - 47	$(w_3)\text{>>>}5_a \oplus w_{2b}$
7	0 - 15	$(w_2)\text{>>>}5_a \oplus (w_3)\text{<<<}5_b$	15	0 - 15	$(w_2)\text{<<<}5_a \oplus w_{3b}$
	16 - 31	$(w_3)\text{<<<}5_a \oplus w_{0b}$		16 - 31	$w_{3a} \oplus (w_0)\text{>>>}5_b$
	32 - 47	$w_{0a} \oplus (w_2)\text{>>>}5_b$		32 - 47	$(w_0)\text{>>>}5_a \oplus (w_2)\text{<<<}5_b$
8	0 - 15	$(w_3)\text{>>>}5_a \oplus (w_1)\text{<<<}5_b$	16	0 - 15	$(w_3)\text{<<<}5_a \oplus (w_1)\text{>>>}5_b$
	16 - 31	$(w_0)\text{<<<}5_a \oplus w_{3b}$		16 - 31	$w_{0a} \oplus (w_3)\text{<<<}5_b$
	32 - 47	$w_{1a} \oplus (w_0)\text{>>>}5_b$		32 - 47	$(w_1)\text{>>>}5_a \oplus w_{0b}$

Table 7-7: The combination of intermediate keys that makes up the round keys for the second version of the modified TDES.

7.6.2 Efficiency of Implementation

The area requirements, clock-speeds and throughputs of the three versions of TDES are compared in Table 7-8. The new modified version of TDES is much smaller and has a significantly faster clock-speed than the first modified version, although the extra clock cycles required to initialise the key mean the throughput is lower. Compared to the original TDES the modified algorithm uses 75% more area and the throughput falls by 15%. These penalties are still lower than those incurred by

the majority of hardware countermeasures and it is important to note that while the increase in area is significant TDES, using the DES key schedule, has one of the simplest key schedules of all algorithms, it consisting merely of a selection of bits from the master key. All other algorithms would have a key schedule that requires more hardware and so there would be a smaller relative penalty when it was replaced.

	TDES	Modified TDES 1	Modified TDES 2
Slices	1,201	6,940	2,101
DFFs	1,215	6,323	1,680
Clock Speed (MHz)	49.478	43.537	50.769
Cycles / Plaintext	19	19	23
Throughput (MB/s)	166.66 MB/s	146.65 MB/s	141.27 MB/s

Table 7-8: The area requirements, clock-speeds and throughputs of the three different versions of TDES.

7.6.3 Countermeasure Efficacy

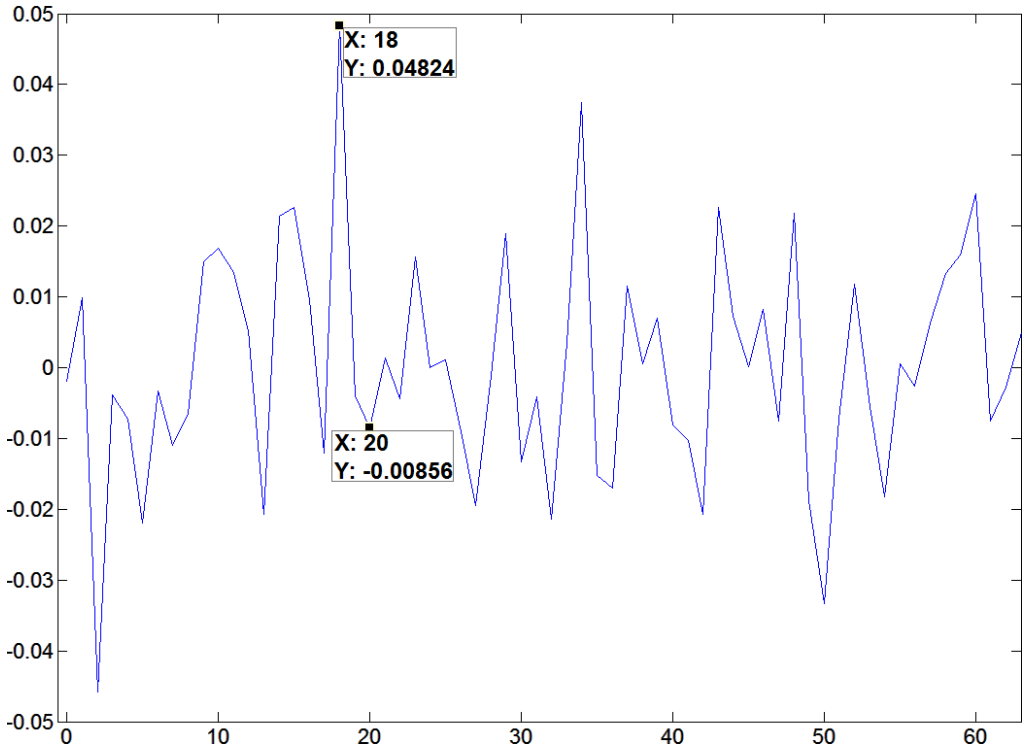


Figure 7-18 : The correlation for all 64 possible key values of the first 6-bit word of the first round key for the second version of the modified TDES.

The new version of the modifications to TDES also protects the algorithm from DPA. In order to show this DPA was attempted using the Modelsim simulation method described in section 5.3.1.1. The first 6-bit word of the first round key was 20, or 0x14, it was not revealed after 4,096 traces. Figure 7-18 shows the correlation for all 64 of the key hypotheses after 4,096 traces, the largest peak is at 18 with a value of 0.04824 and the correct key value is very close to zero and slightly negative at -0.00856. There is no realistic way an attacker could discern the correct value of the key word from this correlation data.

7.6.4 Conclusion

As expected, as it is a combination of the countermeasures proposed in sections 7.2.3 and 7.5, this modification protects TDES from DPA. The design is much smaller than the original TDES modification in section 7.3, only being 75% bigger rather than nearly 500% bigger. Out of the three designs it does have the lowest throughput, even though the clock speed is the highest, this is because it required an additional four clock cycles to initialise the key. If a faster design is required it would be possible to remove the increase in clock cycles by adding an additional DES datapath that would initialise the other three DES blocks' keys while they processed the input. A single DES datapath is approximately 400 slices as the original TDES is 1,201, also an additional 768 bits of memory would be required to store the next set of sub-keys.

7.7 Conclusion

Hardware countermeasures to DPA come with a high cost in terms of area requirements and the throughput of cryptographic designs, sometimes increasing them by a factor of four. Additionally, as discussed in section 4.3.8, they do not offer complete protection, only requiring an attacker to collect more traces before successfully performing DPA. A much better solution is to have algorithms that are already immune to DPA so no costly countermeasures are needed. This chapter proposed three different alterations to AES to protect it from DPA. The only one that was effective was the perpetually expanding key schedule which generated a new set of round keys for each plaintext. This technique is effective as it removes the main strength of DPA which is the ability to exploit power consumption data from several encryptions based on the knowledge that the round keys are always the same.

Other than its effectiveness, a big advantage of this countermeasure is the low overhead, when it was applied to AES the throughput only fell by 28% and the size only increase by 5%. The modification can be easily applied to most modern algorithms, and even when the key schedule is unsuitable, such as TDES, it can simply be replaced before the technique is applied. If this is done it is important to select a suitable key schedule structure to replace the original with. The key schedule of ARIA has properties that work well with this technique, and conform to the definition of a good key schedule given by May *et al.* in [30], as, after initialisation, the round keys can be generated in any order, making the overhead consistent between encryptors and decryptors. Modifying existing algorithms is not the most significant use of this technique, rather, the next generation of algorithms could eliminate DPA entirely as a potential problem.

Chapter 8 Summary

The current state of cryptography is that there are no published mathematical attacks that can break the full versions of modern algorithms, and with $3.4 * 10^{38}$ different values for even 128-bit keys it is not feasible to use brute force to get a key either. With the development of *side channel attacks* in 1996 [19] another avenue for breaking encryption was opened. Side channel attacks exploit the fact that encryption is performed by a physical device which is subject to other physical processes, and by monitoring those "side channels" it is possible to discern information about the data that is being processed by the device. This thesis is mostly concerned with power analysis, where power consumption is the channel that leaks information about the internal state of the device, specifically, *Differential Power Analysis* (DPA). DPA performs a statistical test on a set of power consumption data from several encryptions and predictions about the contents of registers, based on the plaintext and a guess about the value of a byte of the key, in order to determine which of the key value hypotheses is the most likely.

DPA requires calculating the correlation between the predictions of register transitions inside the device and the measured power consumption of the device. There are a number of sources of random variability in the result of this calculation. Firstly there is noise in the circuit, from the power consumption of other parts of the circuit and random thermal noise, and the measurements will also contain errors. All of these contribute to the noise in the SNR and will decrease the correlation between the prediction and the power consumption. The other source of variation in the results is due to the fact that any calculation of the correlation using a series of samples, the *sample correlation*, is only an estimation of the true *population correlation*. The difference between the two values is the *sampling error*. The variance in the sample

correlation is controlled by the number of samples, in this case the number of power traces available to an attacker.

The results of a DPA attack on a single key byte will be made up of 256 correlations, one for the correct key and 255 for the incorrect ones. The population correlation for the predictions based on the correct key guess will be directly related to the SNR. The population correlation of the incorrect values will be the correlation between their predictions and the predictions from the correct guess (controlled by the structure of the s-box) multiplied by the population correlation for the correct predictions; this means the correct guess will always have the highest population correlation. That does not mean that DPA will always be successful. Superimposed on the population correlations is the sampling error. This is a random variable with a variance controlled by the number of samples. If there is a low SNR then the difference between the correct and incorrect correlation will be small (in terms of absolute value rather than ratio), and easily overwhelmed if not enough samples are taken to ensure a small sampling error.

Even though all the factors in the shape of a set of DPA results are controlled by two variables, the SNR and the number of traces, the random element added due to the sampling error means that there is a stochastic element to the results and hence it is never definite that a particular attack will give the correct value. A method to calculate the probability of success from the SNR and number of traces was derived in Chapter 6 of this thesis. An attacker may like to know how many traces would be required to ensure a certain probability of success for a given system with a known SNR. A designer may like to know the value of the SNR that will ensure a particular number of traces are required to give a chosen probability of success. Methods for determining both of these have also been developed in this thesis.

As is the nature of cryptography, whenever a new cryptanalysis technique is developed cryptographers work to develop ways to protect against it. There have been several ideas for modifications to chip designs that will help combat DPA, from balancing the logic so there is always the same number of transitions [82], to masking the intermediate variables in secret shares [8, 23, 88], and using *Dynamic Voltage and Frequency Scaling* (DVFS) to stop the attacker sampling the power consumption at the correct time [11]. None of them are completely effective. There are ways of defeating some of the countermeasures, such as targeting logic gates with DPA [93],

or using *High-Order DPA* (HODPA) [22]. In the cases of DVFS and balanced logic, the countermeasures frustrate attempts at DPA by reducing the SNR and requiring an attacker to record more power data to give themselves a reasonable chance of success [5, 95]. Apart from not offering complete protection from DPA, the other disadvantage of the proposed countermeasures is that they often come with a high cost, significantly increasing the area of the design or decreasing the data throughput sometimes by up to a factor of 4 [5, 23].

All these countermeasures are modifications to the hardware implementations of algorithms. A better solution would be to design algorithms in a way that defeats DPA. In Chapter 7 the DPA mitigation potential of several alterations to the *Advanced Encryption Standard* (AES) algorithm are investigated. The most successful one protects the algorithm by using the key schedule to generate a new set of round keys for each plaintext rather than reuse the same set each time. In DPA the attacker combines the changing, but known, plaintext with a guess about one byte of the round key, which is constant, so the validity of the guess is the same for each plaintext. This is no longer true, and while it would still be possible to correlate predictions about the contents of registers with power consumption, with the correct set of predictions giving the highest value, it becomes infeasible as a way to discover the key. There are only two ways an attacker could be ensured to have a set of predictions that contains the correct answer. Firstly, by trying all possibly combinations of different values of a round key byte for each plaintext, this gives $256^{\text{number of plaintexts}}$, which quickly becomes impossibly large. Secondly, as the influence of the value of a particular byte of the master key is diffused through the entire round key more with each successive key schedule operation, accurate predictions about the value of a given round key byte can only be made when the entire master key is known. An attacker could make a guess at the entire master key and they would still be able to use DPA to determine which of their guesses was correct, but DPA no longer offers any advantage over brute force.

Other than offering full protection from DPA the other main advantage of an algorithmic approach to DPA countermeasures is efficiency of implementation. When AES was modified the size of the implementation of an encryptor increased by 5% and the speed fell by 17%. For a design that could also decrypt the area increased by 28% and the speed fell by 15%. Some key schedules are not suitable for direct

modification, the key schedule for DES (and hence TDES) generates the different round keys by selecting different bits from the master key in different arrangements so it can only produce a limited number from a given key schedule. Even the AES key schedule has some disadvantages with the design, it can only produce round keys forwards, but decryption needs the round keys in reverse order. In order to perform decryption an implementation needs to either interleave round key generation with decryption, which would reduce the throughput of the design, or calculate and store the next set of round keys while one set is being used, which would require more area. As TDES uses three DES key schedules they all had to be replaced with a more complex one and TDES always needs to be able to perform decryption. This meant the modifications came at a significant penalty, especially in terms of area which increased by 230%. Throughput fell by 12%.

After examining a series of modern algorithms a set of design principles for efficient implementation of a modified key schedule was identified. The main disadvantage with the AES key schedule in this context is its inability to generate the round keys in any order. AES decryption always needed the round keys pre-calculated, but in the original algorithm they were only calculated once so it was not a large overhead. Rather than using an iterative approach to generating round keys it would be better to take the master key and apply a series of transforms to it to generate a small number of values and combine these in different ways to get the round keys. This would save both time and area as fewer calculations are required to get the smaller number of values and they require less memory to store. If the key schedules reuse the cryptographic primitives that make up the algorithm then this reduces the amount of hardware that is needed and it ensures good levels of *confusion* and *diffusion,* important measures of the strength of a cipher, in the expanded key. The efficiency of key schedules designed with the rules was confirmed by implementing a modified version of ARIA and an updated modified TDES. The DPA protection cost no extra area for ARIA and 75% extra for TDES and reduced the speed of ARIA by 18% and TDES by 16%.

The ability of DPA to extract information from the power consumption of an electronic device does not have solely cryptanalytic applications. Using DPA as a means for detecting a particular pattern of register transitions can be used to detect a “watermark” in the power consumption, proving the device contains a particular piece

of intellectual property. This can be achieved by adding a block of circuitry whose sole purpose is to produce a known set of register transitions. There are slight differences between this technique and cryptographic DPA. When trying to break encryption there are a set of correlations, one of them definitely corresponds to the correct key and it is assumed to be the one with the highest value. With watermark detecting there is only one value if it above a threshold then the watermark is likely to be there. As it is unlikely that it would be possible to determine the SNR of the watermark without first knowing that it is present, it is not possible to know what value for the correlation to expect, making it difficult to set the threshold. Fortunately, there is another important difference that helps, with cryptographic DPA the correlations for the incorrect key guesses were non-zero, if the watermark is not there then the correlation will be between two completely unrelated sets of numbers so the population correlation will be zero. The probability that the measured value is above a particular value, if the watermark is not there, is based on the sampling error, and hence the number of power traces that are taken. In section 6.5 of this thesis a statistical test is described that is able to determine whether it is reasonable to assume that a watermark is present. In situations where the SNR of the potential watermark is known, a method of calculating the number of traces required to get a given probability of successfully detecting it is also derived.

This thesis presents a novel approach to DPA countermeasures that are both efficient to implement in hardware and prevent rather than impede the attack. Also a statistical model of DPA is derived and used to find a method to calculate the probability that a particular attack will be successful. From this it is also possible to calculate the SNR or number of traces that would be required to ensure a given probability of success, useful for the designers of either crypto-systems or DPA attacks. A benign use for DPA was also explored, and a method for detecting a watermark for protecting intellectual property was derived.

Chapter 9 Conclusion and Future Work

9.1 Conclusion

DPA is a statistical attack, by understanding the statistics behinds how the results are generated knowledge about how they are affected by changes in the SNR and number of traces used in the attack can be gained. This is important as it allows the analysis of potential countermeasures and attacks before they are implemented. In this thesis a statistical model of the DPA attack was created. From the model a method for calculating the probability of successfully retrieving a single byte of a key based on the SNR of the system and the number of traces. Using this it is possible to assess whether an attack is likely to succeed before performing it. The method can also be modified to calculate the number of traces required to ensure a given probability of success for an attack on a particular system, or the amount of noise required to ensure an attacker must take a minimum number of traces in order for them to guarantee a given probability of success. These can be used as tools for either an attacker to plan his attack in advance (assuming he has knowledge of the SNR), or for the designer of a cryptographic device to guarantee a particular level of security against the attack.

There are two problems with previous attempts at adding DPA countermeasures to cryptographic hardware, they are very expensive, reducing the performance and increasing the size of designs, and that, generally due to inevitable imperfections in their implementation, they can never offer complete protection from DPA, only reduce the correlation between the data being processed and the power consumption.

As can be shown from the statistical model of DPA, this reduction can never completely stop DPA but only make it more inconvenient, requiring an attacker to record more power traces. The main innovation in the countermeasure developed during the course of this research was where to put it, instead of adding the countermeasure to the completed implementation of the hardware it was added to the algorithm itself. This has two potential advantages, the algorithm can still be implemented in an efficient way, and by using a suitable technique it will offer complete protection from DPA. Clearly there is no algorithmic way to divorce the power consumption from the data being processed, but by attacking the assumption on which DPA is based a technique to prevent it can be found.

To retrieve a key using DPA the correlation is calculated and used as a test to determine which hypothesis about the value of a single byte of the key is correct. This is only possible because multiple samples are available, all with the same value for a given byte of a round key. If this is rendered untrue then the entire attack falls apart. This can be achieved by creating a key schedule that constantly changes the value of the round key. The technique was used to modify AES, TDES and ARIA. It always offered protection from the attack and during the course of adapting the technique for these algorithms rules for ensuring the key schedules could be implemented efficiently were developed. Thus it has been demonstrated that algorithmic countermeasures to DPA can completely remove the threat of DPA which still allowing efficient implementations of the algorithm.

The uncanny ability of DPA to divine the internal state of a device can be put to other, more benign, tasks. This can be seen in the development of a method for adding a watermark to intellectual property by including hardware that will produce a known set of register transitions, and hence power consumption, that can then be detected using a DPA-like technique.

9.2 Summary of Contributions

This work has resulted in a conference paper and a journal paper (under consideration). The specific contributions are outlined below.

- A novel, algorithmic based method for defeating DPA was devised.

- Identification of the design principles of key schedules for efficient implementation of algorithms that use the new technique.
- A statistical model of the DPA attack was derived.
- A method for calculating the probability that an attack will be successful given the number of traces that were recorded and the SNR of the system.
- A technique for using the above method to calculate the SNR or the number of traces required to give a particular probability of success.
- A method for determining if it is reasonable to believe that a specific watermark is present in the power consumption of a design.
- A method for calculating the number of traces required to get a given probability of being able to detect the watermark present in the power consumption do a design given the SNR of the watermark.

9.3 Future Work

9.3.1 A method for calculating the probability that a set of results from a DPA attack gives the correct value.

DPA does not always give the correct result. When the attack is being performed in the lab this is of little practical concern, generally the researcher has set the value for the key so already knows the value and can tell if the value is correct or not. For an actual attacker this is not true and if the highest peak is small it is not always clear whether that is the correct result. The accuracy of a DPA attack can be judged approximately by eye, but only with any accuracy when the cases are extreme, either with one large, obvious peak or when there are several peaks of approximately the same height. If DPA were ever to be used in a real world situation it would need a formal method to determine the validity of results.

9.3.2 A method for estimating the SNR of a system using only the results from a relatively small set of DPA results

Both the population correlation of the correct key guess (which the highest peak is in theory an estimate of) and the standard deviation of the results are related to the

SNR of the system. The SNR is a very useful piece of information to have as it enables an attacker to determine how many traces would be required to have a good chance of cracking the system, and it would help in evaluating the correctness of any results. In theory it is possible to take either the highest peak in the DPA or the standard deviation of the results and estimate a value for the SNR. The population correlation can be converted into the SNR using equation (6-4). The standard deviation of the DPA results is given by the following formula:

$$std(DPA) = \sqrt{\left(\sqrt{1/Traces - 3}\right)^2 + \left(\frac{SNR}{9.6}\right)^2} \tag{9-1}$$

The value 9.6 was determined empirically, Figure 9-1 shows a graph of the standard deviation of DPA results from a Matlab simulation vs. SNR, and the modelled relationship using the above formula.

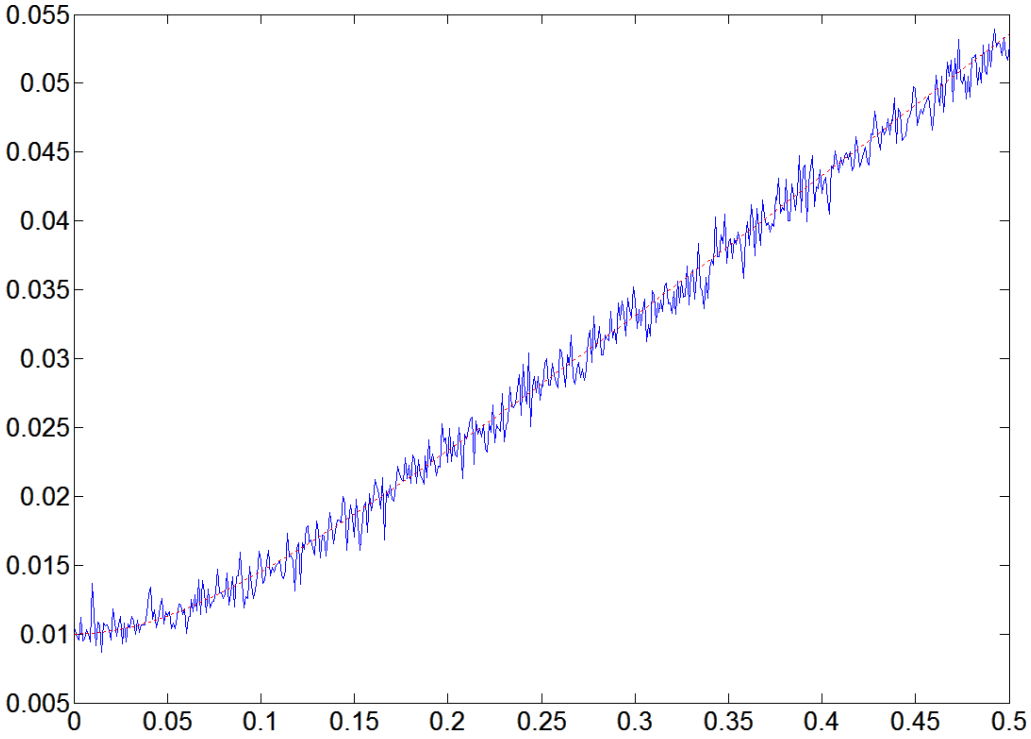


Figure 9-1 : The standard deviation of DPA results vs SNR for the results from Matlab simulations and a model of the relationship.

The problem is that the estimate will only be accurate if the number of traces is fairly large compared to the SNR, and in this case knowing the value of the SNR is less useful as the attacker generally has a clear, unambiguous peak that generally indicates a correct result. Finding a way of accurately determining the SNR from the

results when there are not enough traces to give an accurate key value would enable an attacker to better plan and evaluate his attack.

9.3.3 Improving the accuracy of DPA by tuning the power consumption model to a particular device.

If there is any non-linear behaviour in the power consumption model, i.e. if not all bits make the same contribution to the power consumption, or a 0 to 1 transition consumes a different amount of power to a 1 to 0 transition, then the population correlation of the correct guess could be reduced making it more difficult to perform DPA. If this is the case then it might be possible to improve the accuracy of DPA on a particular device by measuring this and then compensating for it in the power consumption model.

References

[1] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," 1999.

[2] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *International Cryptology Conference on Advances in Cryptology*, 1999, pp. 388-397.

[3] A. Schuster, "Differential Power Analysis of an AES Implementation," Institut for Applied Information Processing and Communications – IAIK25/6/2004 2004.

[4] E. Oswald and K. Schramm, "An Efficient Masking Scheme for AES Software Implementations," in *Workshop on Information Security Applications—WISA 2005*, 2005, pp. 292 - 305.

[5] K. Tiri, D. Hwang, A. Hodjat, B.-C. Lai, S. Yang, P. Schaumont, and I. Verbauwhede, *Prototype IC with WDDL and Differential Routing - DPA Resistance Assessment*, 2005.

[6] J. Blömer, J. Guajardo, and V. Krummel, "Provably Secure Masking of AES," in *Selected Areas in Cryptography Workshop*, 2004, pp. 69-83.

[7] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen, *A Side-Channel Analysis Resistant Description of the AES S-Box*, 2005.

[8] E. Trichina, D. De Seta, and L. Germani, "Simplified Adaptive Multiplicative Masking for AES," in *CHES 2002, Cryptographic Hardware and Embedded Systems*, Redwood Shores, CA, USA, 2002, pp. 187 - 197.

[9] N. Pramstaller, F. K. Gurkaynak, S. Haene, H. Kaeslin, N. Felber, and W. Fichtner, "Towards an AES crypto-chip resistant to differential power analysis," in *European Solid-State Circuits Conference*, 2004, pp. 307-310.

[10] S. Chari, C. Jutla, S., J. Rao, R., and P. Rohatgi, "Towards Sound Approaches to Counteract Power-Analysis Attacks," in *Advances in Cryptology*, 1999, pp. 398-412.

[11] S. Yang, W. Wolf, N. Vijaykrishnan, D. Serpanos, and Y. Xie, "Power Attack Resistant Cryptosystem Design: A Dynamic Voltage and Frequency Switching Approach," *Design, Automation and Test in Europe,* vol. 3, pp. 64 - 69, 2005.

[12] J. Casanova, "The Complete Memoirs of Casanova," Globusz Publishing, New.

[13] S. Singh, *The code book*: Delacorte Press, 2002.

[14] C. E. Shannon, "Communication Theory of Secrecy Systems," *Bell System Technical Journal,* vol. 28, pp. 656-715, 1949.

[15] H. Feistel, "Cryptography and Computer Privacy," in *Scientific American*. vol. 228, 1973, pp. 15 - 23.

[16] M. Matsui and A. Yamagishi, "A new method for known plaintext attack of FEAL cipher," in *EUROCRYPT*, Balatonfűred, Hungary, 1992.

[17] M. Matsui, "The First Experimental Cryptanalysis of the Data Encryption Standard," in *EUROCRYPT* Santa Barbara, California, USA, 1994.

[18] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler, "Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker," in *CHES 2006*, Yokohama, Japan, 2006.

[19] P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *CRYPTO*, 1996, pp. 104-113.

[20] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM Side-Channel(s)," in *Cryptographic Hardware and Embedded Systems*, 2002, pp. 29-45.

[21] A. Shamir and E. Tromer, "Acoustic cryptanalysis - On nosy people and noisy machines," http://www.wisdom.weizmann.ac.il/~tromer/acoustic/ Accessed: 17/6/07

[22] L. Goubin and J. Patarin, "DES and Differential Power Analysis The Duplication Method," in *Cryptographic Hardware and Embedded Systems International Workshop*, 1999, pp. 158-172.

[23] N. Pramstaller, E. Oswald, S. Mangard, F. K. Gürkaynak, and S. Häne, "A Masked AES ASIC Implementation," in *Austrochip 2004*, Villach, Austria, 2004, pp. 77 - 82.

[24] Y. Shengqi, W. Wayne, N. Vijaykrishnan, D. N. Serpanos, and X. Yuan, "Power Attack Resistant Cryptosystem Design: A Dynamic Voltage and Frequency Switching Approach," 2005, p. 64.

[25] K. Tiri and I. Verbauwhede, "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation," in *Design Automation and Test in Europe*, 2004, pp. 246-251.

[26] E. Biham and A. Shamir, "Differential cryptanalysis of DES-like cryptosystems," *Journal of Cryptology,* vol. 4, pp. 3-72, 5/2/91 1991.

[27] E. Biham and A. Biryukov, "An Improvement of Davies' Attack on DES," *Journal of Cryptology,* vol. 10, pp. 195 - 205, 1997 1997.

[28] "COPACOBANA - Special-Purpose Hardware for Code-Breaking," http://www.copacobana.org/ Accessed:

[29] "RSA Laboratories - DES Challenge III," http://www.rsa.com/rsalabs/node.asp?id=2108 Accessed:

[30] L. May, M. Henricksen, W. Millan, and G. Carter, "Strengthening the Key Schedule of the AES," in *Information Security and Privacy*, 2002, pp. 226-240.

[31] N. Feguson and B. Schnier, *Practical Cryptography*, 2003.

[32] N. I. o. S. a. Technology, "DATA ENCRYPTION STANDARD (DES)," FIPS1999.

[33] "FIPS PUB 46 - Data Encryption Standard," National Bureau of Standards1977.

[34] D. Coppersmith, "The Data Encryption Standard (DES) and its strength against attacks," *IBM Journal of Research and Development,* vol. 38, pp. 243 - 250, May 1994 1994.

[35] W. Diffie and M. E. Hellman, "Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard," *Computer,* vol. 10, pp. 74-84, 1977.

[36] R. C. Merkle and M. E. Hellman, "On the security of multiple encryption," *Commun. ACM,* vol. 24, pp. 465-467, 1981.

[37] P. van Oorschot and M. Wiener, "A Known-Plaintext Attack on Two-Key Triple Encryption," in *Advances in Cryptology — EUROCRYPT '90*, 1991, pp. 318-325.

[38] S. Lucks, "Attacking Triple Encryption," *Fast Software Encryption,* vol. LNCS 1372, pp. 239–253, 1998.

[39] "Announcing the ADVANCED ENCRYPTION STANDARD (AES)," FIPS, Ed.: NIST, 2001.

[40] E. T. Bell, *Men of Mathematics*: Simon & Schuster, 1937.
[41] S. Morioka and A. Satoh, "A 10-Gbps Full-AES Design with a Twisted BDD S-Box Architecture," *IEEE Transactions on VLSI Systems* vol. 12, pp. 686-691, 2004.
[42] H. Brunner, A. Curiger, and M. Hofstetter, "On computing multiplicative inverses in GF(2m)," *IEEE Transactions on Computers,* vol. 42, pp. 1010-1015, 1993.
[43] R. W. Ward and T. C. A. Molteno, "Efficient Hardware Calculation of Inverses in GF (2^8)," in *Electronics New Zealand*, 2003.
[44] V. Rijmen, "Efficient Implementation of the Rijndael S-box," 2005.
[45] A. Hodjat and I. Verbauwhede, "Minimum Area Cost for a 30 to 70 Gbits/s AES Processor," in *IEEE Computer Society Annual Symposium on VLSI Emerging Trends in VLSI Systems Design*, 2004, pp. 83-88.
[46] X. Zhang and K. Parhi, "Implementation Approaches for the Advanced Encryption Standard Algorithm," *IEEE Circuits and Systems Magazine,* vol. 2, p. 24, 2002.
[47] A. Hodjat and I. Verbauwhede, "Speed-area trade-off for 10 to 100 Gbits/s throughput AES processor," in *Asilomar Conference on Signals, Systems and Computers,*, 2003, pp. 2147-2150.
[48] N. S. Kim, T. Mudge, and R. Brown, "A 2.3 Gb/s Fully Integrated and Synthesizable AES Rijndael Core," in *IEEE Custom Integrated Circuits Conference*, 2003.
[49] T.-F. Lin, C.-P. Su, C.-T. Huang, and C.-W. Wu, "A High-Throughput Low-Cost AES Cipher Chip," in *Asia-Pacific Conference on AISIC*, 2002, pp. 85-88.
[50] C.-P. Su, T.-F. Lin, C.-T. Huang, and C.-W. Wu, "A Highly Efficient AES Cipher Chip," in *Asia and South Pacific Design Automation Conference*, 2003, pp. 561-562.
[51] S. S. Wang and W. S. Ni, "An Efficient FPGA Implementation of Advanced Encryption Standard Algorithm," in *IEEE International Symposium on Circuits and Systems*. vol. 2, 2004, pp. 597-600.
[52] M. McLoone and J. V. McCanny, "High Performance Single Chip FPGA Rijndael Algorithm Implementations," in *Cryptographic Hardware and Embedded Systems*, 2001, pp. 65-76.
[53] G. Rouvroy, F.-X. Standaert, J.-J. Quisquater, and J.-D. Legat, "Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Applications," in *Information Technology: Coding and Computing*, 2004, pp. 583-587.
[54] X. Zhang and K. Parhi, "High-speed VLSI Architectures for the AES Algorithm," in *IEEE Transactions on VLSI Systems*. vol. 12, 2004, pp. 957-967.
[55] K. Jarvinen, M. Tommiska, and J. Skytta, "A Fully Pipelined Memoryless 17.8 Gbps AES-128 Encryptor," in *International Symposium on Field-Programmable Gate Arrays*, 2003.
[56] L. E. Bassham, "The Advanced Encryption Standard Algorithm Validation Suite (AESAVS) ": National Inst.of Standards and Technology, 2002.
[57] N. Ferguson and B. Schneier, "Practical Cryptography," John Wiley & Sons, 2003, pp. 56-57.
[58] J. Daemen, L. Knudsen, and V. Rijmen, "The Block Cipher SQUARE," in *Fast Software Encryption*, 1997, pp. 149-156.

[59] F.-X. Standaert, L. van Oldeneel tot Oldenzeel, D. Samyde, and J.-J. Quisquater, "Power Analysis of FPGAs: How Practical Is the Attack," in *Field-Programmable Logic and Applications*, Lisbon, Portugal, 2003, pp. 707-711.

[60] F.-X. Standaert, S. B. Örs, and B. Preneel, "Power Analysis of an FPGA Implementation of Rijndael: Is Pipelining a DPA Countermeasure," in *Cryptographic Hardware and Embedded Systems*, Cambridge, MA, USA, 2004, pp. 30-44.

[61] S. B. Örs, F. Gürkaynak, E. Oswald, and B. Preneel, "Power-Analysis Attack on an ASIC AES implementation," in *International Conference on Information Technology: Coding and Computing (ITCC'04)*, 2004, p. 546.

[62] D. Asonov and R. Agrawal, "Keyboard Acoustic Emanations," 2004.

[63] L. R. Knudsen and J. E. Mathiassen, "On the Role of Key Schedules in Attacks on Iterated Ciphers," *Lecture Notes in Computer Science,* vol. 3193, pp. 322-334, Jan 2004 2004.

[64] R. Bevan and E. Knudsen, "Ways to Enhance Differential Power Analysis," in *Information Security and Cryptology*, 2002, pp. 327-342.

[65] A. Rastogi, K. Ganeshpure, and S. Kundu, "A Study on Impact of Leakage Current on Dynamic Power," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, 2007, p. 1069.

[66] L. Lang and W. Burleson, "Leakage-based differential power analysis (LDPA) on sub-90nm CMOS cryptosystems," in *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, 2008, p. 252.

[67] E. Oswald, "Differential Power Analysis Attacks - A New Generation?," IAIK.

[68] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," *LECTURE NOTES IN COMPUTER SCIENCE,* pp. 16-29, 2004.

[69] P. Fahn and P. Pearson, "IPA: A New Class of Power Attacks," in *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, 1999, pp. 173-186.

[70] T. Messerges, "Using Second-Order Power Analysis to Attack DPA Resistant Software," in *Cryptographic Hardware and Embedded Systems*, 2000, pp. 238-251.

[71] J. Waddle and D. Wagner, "Towards Efficient Second-Order Power Analysis," *Lecture Notes in Computer Science,* p. 1, 2004.

[72] R. A. Fisher, "Frequency distribution of the values of the correlation coefficient in samples of an indefinitely large population," *Biometrika,* vol. 10, pp. 507-521, 1915.

[73] S. Mangard, "Hardware Countermeasures against DPA-A Statistical Analysis of Their Effectiveness," *LECTURE NOTES IN COMPUTER SCIENCE,* pp. 222-235, 2004.

[74] E. Prouff, "DPA Attacks and S-Boxes," in *Fast Software Encryption*, 2005, pp. 424-441.

[75] B. Preneel, R. Govaerts, and J. Vanderwalle, "Boolean Functions Satisfying Higher Order Propergation Criteria," in *EUROCRYPT '85*, 1985, pp. 141 - 152.

[76] C. Carlet, "On highly nonlinear S-boxes and their inability to thwart DPA attacks (completed version)," *Cryptology ePrint Archive, Report 2005/387,* 2005.

[77] L. Bohy, M. Neve, D. Samyde, and J.-J. Quisquater, "Principal and Independent Component Analysis for Crypto-systems with Hardware Unmasked Units," 2003.

[78] J.-J. Quisquater and D. Samyde, "Automatic Code Recognition for smart cards using a Kohonen neural network," in *Smart Card Research and Advanced Application Conference*, San Jose, CA, USA, 2002.

[79] R. Novak, "Sign-Based Differential Power Analysis," *Lecture Notes in Computer Science,* pp. 203-216, 2004.

[80] A. Yu and D. S. Bree, "A clock-less implementation of the AES resists to power and timing attacks," in *Information Technology: Coding and Computing*, 2004, pp. 525-532.

[81] K. Tiri and I. Verbauwhede, *A VLSI Design Flow for Secure Side-Channel Attack Resistant ICs*: IEEE Computer Society, 2005.

[82] K. Tiri and I. Verbauwhede, "Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology," in *Cryptographic Hardware and Embedded Systems*
2003, pp. 125-136.

[83] K. Tiri and I. Verbauwhede, "Place and Route for Secure Standard Cell Design," in *CARDIS 2004-Sixth Smart Card Research and Advanced Application IFIP Conference*, Toulouse, France, 2004

[84] K. Tiri and I. Verbauwhede, "Charge recycling sense amplifier based logic: securing low power security ICs against DPA," in *ESSCIRC 2004, European Solid-State Circuits Conference*, 2004, pp. 179 - 182.

[85] D. Suzuki and M. Saeki, "Security Evaluation of DPA Countermeasures Using Dual-Rail Pre-charge Logic Style," in *Cryptographic Hardware and Embedded Systems - CHES 2006*, 2006, p. 255.

[86] L. Goublin and J. Patarin, "DES and Differential Power Analysis The Duplication Method," in *Cryptographic Hardware and Embedded Systems International Workshop*, 1999, pp. 158-172.

[87] M.-L. Akkar and C. Giraud, "An Implementation of DES and AES, Secure against Some Attacks," in *CHES 2001, Cryptographic Hardware and Embedded Systems*, Paris, France, 2001, pp. 309 - 318.

[88] E. Oswald, S. Mangard, and N. Pramstaller, "Secure and Efficient Masking of AES - A Mission Impossible?," 2004.

[89] M.-L. Akkar and L. Goubin, "A Generic Protection against High-Order Differential Power Analysis," in *Fast Software Encryption*, 2003, p. 192.

[90] N. Prarnstaller, F. K. Gurkaynak, S. Haene, H. Kaeslin, N. Felber, and W. Fichtner, "Towards an AES crypto-chip resistant to differential power analysis," in *European Solid-State Circuits Conference*, 2004, pp. 307-310.

[91] J. D. Golic and C. Tymen, "Multiplicative Masking and Power Analysis of AES," in *Cryptographic Hardware and Embedded Systems - CHES 2002*, Redwood Shores, CA, USA, 2003, pp. 198 - 212.

[92] M.-L. Akkar, R. Bévan, and L. Goubin, *Two Power Analysis Attacks against One-Mask Methods*, 2004.

[93] S. Mangard, N. Pramstaller, and E. Oswald, "Successfully Attacking Masked AES Hardware Implementations," in *Cryptographic Hardware and Embedded Systems – CHES 2005*, 2005, pp. 157-171.

[94] F. K. Guürkaynak, A. Burg, N. Felber, W. Fichtner, D. Gasser, F. Hug, and H. Kaeslin, "A 2 Gb/s balanced AES crypto-chip implementation " in *ACM Great Lakes symposium on VLSI* Boston, MA, USA pp. 39 - 44

[95] K. Baddam and M. Zwolinski, "Evaluation of Dynamic Voltage and Frequency Scaling as a Differential Power Analysis Countermeasure," 2007, p. 854.
[96] H. Chang and K. Kim, "Securing AES against Second-Order DPA by simple Fixed-Value Masking," in *Computer Security Symposium*, 2003, pp. 145-150.
[97] M.-L. Akkar and L. Goubin, "A Generic Protection against High-Order Differential Power Analysis," in *Fast Software Encryption*, Lund, Sweden, 2003, p. 192.
[98] D. D. Hwang, K. Tiri, A. Hodjat, B. C. Lai, S. Yang, P. Schaumont, and I. Verbauwhede, "AES-Based Security Coprocessor IC in 0.18-µm CMOS With Resistance to Differential Power Analysis Side-Channel Attacks," *IEEE Journal of Solid-State Circuits,* vol. 41, pp. 781- 792, April 2006 2006.
[99] S. Levy, *Crypto*, 2001.
[100] Y. Alkabani and F. Koushanfar, "Active hardware metering for intellectual property protection and security."
[101] I. Hong and M. Potkonjak, "Behavioral synthesis techniques for intellectual property protection," 1999, pp. 849-854.
[102] M. Ward, "EMV card payments–An update," *Information Security Technical Report,* vol. 11, pp. 89-92, 2006.
[103] OpenCores, "3DES (Triple DES) / DES (VHDL) (3des_vhdl)," OpenCores, http://opencores.org/?do=project&who=3des_vhdl Accessed: 10/10/2007
[104] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S. Matyas Jr, L. O'Connor, M. Peyravian, and D. Safford, "MARS-a candidate cipher for AES," *NIST AES Proposal, Jun,* 1998.
[105] R. Rivest, M. Robshaw, R. Sidney, and Y. Yin, "The RC6 Block Cipher," *NIST AES Proposal, Jun,* 1998.
[106] R. Rivest, "The RC5 encryption algorithm," *LECTURE NOTES IN COMPUTER SCIENCE,* pp. 86-86, 1995.
[107] R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," *NIST AES Proposal, Jun,* 1998.
[108] M. Matsui, "New Block Encryption Algorithm MISTY," *LECTURE NOTES IN COMPUTER SCIENCE,* pp. 54-68, 1997.
[109] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, "Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms–Design andAnalysis."
[110] K. Ohkuma, H. Muratani, F. Sano, and S. Kawamura, "The block cipher Hierocrypt," *Lecture notes in computer science,* pp. 72-88, 2001.
[111] D. Kwon, J. Kim, S. Park, S. Sung, Y. Sohn, J. Song, Y. Yeom, E. Yoon, S. Lee, and J. Lee, "New Block Cipher: ARIA," *LECTURE NOTES IN COMPUTER SCIENCE,* pp. 432-445, 2004.