SIDE CHANNEL ANALYSIS OF STREAM CIPHER HARDWARE

JONATHAN ANDERSON

Side Channel Analysis of

Stream Cipher Hardware

by

©Jonathan Anderson, B.Eng.

A thesis submitted to the

School of Graduate Studies

in partial fulfillment of the

requirements for the degree of

Master of Engineering in Computer Engineering

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

September 2008

St. John's, Newfoundland

Abstract

In today's world of ubiquitous connectivity, communications security is an ever-present concern. In order to protect sensitive information from eavesdropping by foreign governments, identity thieves and other curious individuals and organizations, cryptography is today deployed on a wide scale. No longer strictly the domain of large banks and governments, cryptographic systems are found in such everyday places as building passes and vehicle ignition keys. Cryptanalysis is the study of methods – called *attacks* – that can be used to extract secret information from these cryptographic systems. It is largely a statistical discipline, but out of it has grown a more hands-on approach: side channel analysis.

Side channel analysis is an exciting field of study which attempts to extract secret information from cryptographic systems though the careful measurement of physical characteristics such as power usage and execution time. These characteristics provide "side channels" of information flow that algorithm designers may not anticipate. This research focuses of the power side channel, which extracts information from the instantaneous power either used or radiated by a cryptographic system. Traditional forms of power analysis are ineffective against a large class of ciphers called stream ciphers, but a recently-introduced group of techniques – template attacks – have been shown to be effective against microcontroller-based implementations of stream ciphers.

This thesis describes the theory behind template attacks, and describes how we have applied them to perform power analysis of hardware implementations of stream ciphers. We have built hardware for this purpose, called the Side Channel Analysis Board (SCAB) as well as designed software to perform the necessary analysis. We used our experimental setup to measure the power usage of FPGA-based hardware specifically the Actel ProASIC3 - running a stream cipher building block called LFSR-16. We have also simulated and analysed the power usage of LFSR-16 and a functional stream cipher, Trivium. Trivium is a hardware-focused stream cipher that was vetted by the recent eSTREAM initiative, and is thus of great importance. In both simulation and hardware, we were able to extract secret key information with a probability greater than we would expect to achieve through random guessing. In the case of the cipher building block LFSR-16, we were able to correctly classify four key bits with accuracy greater than 90%. In the case of the stream cipher Trivium, average classification success exceeded 20% where random guessing would have achieved a success rate of just 6.25%.

Thus, we may state that the template attack technique is applicable to hardware-based stream ciphers, and that implementers of such ciphers must be aware of such techniques and attempt to apply appropriate countermeasures where possible.

Acknowledgements

Chrissy, my wife, has been an unfailing source of encouragement and joy.

Dr. Howard Heys, my supervisor, has provided me with the freedom to explore and the guidance to succeed.

Mr. Chris Batten, of MUN Technical Services, lent his invaluable aid in assembling the SCAB platform. After a design flaw was discovered in SCAB Mk I, his steady hand re-routed a single signal from a 208-pin surface-mount chip, and saved a month's worth of refabrication.

This research was supported by the Natural Sciences and Engineering Research Council (NSERC), through the Canada Graduate Scholarship and Discovery Grant programs.

Soli Deo gloria.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The world today is more connected than it has ever been. Business employees log into corporate computers from home via Virtual Private Networks (VPNs), banking customers access their accounts via mobile phones and billions of dollars are spent in online shopping and auctions. With all of this sensitive information flowing across public networks, the incentive for criminals and others to eavesdrop is very high, so security is a top priority.

The study of securing communications is *cryptography*, and it is concerned with two central problems: how to safeguard secret messages, and how to bypass the safeguards of others. The solution to each problem benefits the other, as we cannot build or select security tools without understanding the attacks that may be applied against them. With this principle in mind, in the work presented in this thesis we proceed to attack ciphers that have been implemented in digital hardware, in an effort to circumvent their protections and extract secret information.

The primary tools of cryptography are ciphers, which perform *encryption* (to protect information that is to be kept secret) and *decryption* (to render encrypted data readable again). These ciphers can be classified as belonging to one of two sets:

block ciphers or stream ciphers. There are different applications for these ciphers, but both are important. In 2001, the US National Institute of Standards and Technology (NIST), after a competitive process, published the Advanced Encryption Standard (AES) [1], which has become the *de facto* global standard for block ciphers. In 2008, the European Union's eSTREAM process identified a portfolio of strong stream ciphers - F-FCSR-H v2 [2], Grain v2 [3], MICKEY v2 [4] and Trivium [5] - and it is to this more recently recognized group that we turn our attention.

Our goal, then, is to extract secret information from stream ciphers surreptitiously; i.e. to *attack* them. Rather than the traditional (and well-studied) methods of *cryptanalysis*, whereby mathematical relationships are found among secret information and encrypted data, we turn to the newer approach of *side channel analysis* [6], which extracts secret information from careful measurement of physical quantities such as power consumption.

Traditional forms of side channel analysis are often ineffective against stream ciphers, but a recent class of techniques known as template attacks [7] have proved effective against microcontroller-based implementations of stream ciphers (see Chapter 3). Microcontrollers, however, are large, complex systems. The question before us was, could such attacks be effective against hardware implementations of cryptographic systems? Could we demonstrate their efficacy, not just against a theoretical model of power usage, but against physical hardware? Such a demonstration would impact the design and implementation of stream cipher hardware in embedded hardware such as smart cards and RFIDs, which could impact on the payment and authentication technology sectors.

We built both hardware and software in an attempt to answer these questions. This experimental setup, which is comprised of a custom FPGA-bearing PCB, a purpose-bought mixed-signal oscilloscope and thousands of lines of analysis software,

is described in detail in Chapter 4.

Finally, we discovered the answers to our questions: yes, template attacks are effective against the power usage of hardware cryptosystems, and yes, this effectiveness can be demonstrated using physical hardware.

Chapter 2

Background

2.1 Cryptography

The word *cryptography* comes from the Greek κρυπτός (secret) and γραφεα (writing) [8]. Cryptography is the "science and art of designing ciphers," [9] which are used in many applications to make secret the messages communicated among two or more parties. A basic understanding of cryptography, and its goals, is requisite to understanding the purpose and methodology of the attack that we will present in Chapter 3, and whose results we will give in Chapter 5.

2.1.1 Goals and Actors

Cryptography has many goals, including confidentiality (the ability to keep secrets from those who we wish not to know them), integrity (the ability to verify that messages have not been altered), authentication and non-repudiation (the ability to prove that a party sent a message, even if they choose to deny it later). To illustrate these goals, we will introduce three characters who figure prominently in the literature: Alice, Bob and Eve.

Alice and Bob In the literature, Alice and Bob are often used to represent any two parties who wish to communicate in a secure manner [10, 11]. Since their communications are of a sensitive nature, they use cryptographic tools to protect the content of their messages from being discerned by eavesdroppers (e.g. learning the name of a reporter's source), to prevent adversaries from making undetectable changes to the substance of their messages (e.g. changing a beneficiary's name in a will), and if desired, to prevent them from later denying that they sent a particular message (e.g. an agreement to pay for a good or service). Stated more formally, they use cryptography to provide their communications with confidentiality, integrity, authentication and non-repudiation.

Eve Eavesdroppers are commonly represented by an actor named Eve. Eve is assumed to have complete access to the communications channels that Alice and Bob are using, even the ability to send messages to one or both parties, but good cryptography will prevent her from understanding what Alice and Bob communicate (violating confidentiality), changing the meaning of messages (violating integrity), masquerading as either Alice or Bob (falsifying authentication) or helping either party deny their communication (repudiating transactions).

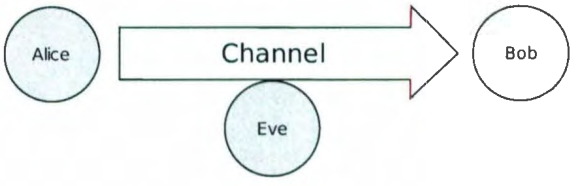

Figure 2.1: Alice, Bob and Eve

2.1.2 Ciphers and Attacks

The primary cryptographic tool used to provide confidentiality is the cipher. A cipher transforms information that we wish to remain confidential – the *plaintext* – into a stream of data – the *ciphertext* – that can be safely transmitted via untrusted channels such as public networks. This transformation is called *encryption*, and it – as well as the reverse transformation, *decryption* – is parametrized by secret information called the *key*. Without this key, an adversary in possession of ciphertext material should not be able to decrypt any of the ciphertext to read the original plaintext.

2.1.2.1 Cryptanalysis

The field of *cryptanalysis* is dedicated to finding weaknesses in cryptographic algorithms such as ciphers, whether for the purposes of better understanding cipher design (as in academic settings) or eavesdropping on secret communications (as in some industrial or governmental settings). There are several methods that can be used to attack a cipher, all of which assume that the attacker knows the cipher being used [12]:

Ciphertext-only attack In this type of attack, it is assumed that the attacker has access to ciphertext, as well as knowledge of the cipher algorithm. It should be computationally infeasible for the attacker to ascertain any plaintext or key information.

The most obvious such attack is an exhaustive search (colloquially, a "brute force" attack). In this approach, the attacker checks every possible key to see if it can be used to decrypt the given ciphertext into an intelligible plaintext. This approach is very inefficient: for an n-bit key, the expected number of keys the attacker must

search, N, is

$$N = 2^{n-1}. \tag{2.1}$$

For the block cipher DES (the Data Encryption Standard) [13], an attacker can expect to search through $2^{55} = 36 \times 10^{15}$ keys. This can be achieved today using dedicated hardware such as the "Deep Crack" machine [14], so newer encryption standards use longer keys [1]. For instance, the smallest key permissible for use with AES is 128 bits [1], so we would expect an exhaustive search to take $N = 2^{127} = 1.7 \times 10^{77}$ decryption operations. An attacker would have be be able to search over 10^{60} keys *per second* in order to expect to finish this search before the death of our sun [15].

Known-plaintext attack In a known-plaintext attack, the attacker has knowledge of the cipher algorithm, ciphertext *and* corresponding plaintext. Even with full knowledge of cipher input and output, it should still be computationally infeasible for the attacker to determine the key (or to decrypt later ciphertext).

Chosen-plaintext attack In this most powerful type of theoretical attack, not only does the attacker have full knowledge of cipher input and output, but she can actually choose plaintexts that are convenient for her purposes. A secure cipher will resist chosen-plaintext attacks – it will still be computationally infeasible for the attacker to determine any key information, or to be able to decrypt later ciphertexts whose plaintexts are not known to the attacker.

Implementation attack Beyond the realm of strict cryptanalysis – attacks on cipher algorithms – there is also a class of attacks that exploit physical properties of cipher implementations. Such implementation attacks include timing analysis, fault analysis, power analysis and electromagnetic analysis, and will be discussed in Section

2.2.

2.1.2.2 Public Key Cryptography

One of the fundamental problems of classical cryptography was the *key distribution problem* [16]. People separated by long distances could protect their communications via ciphers, but this protection was meaningless unless a secret key could be securely communicated. Banks and governments could use trusted couriers and diplomatic pouches, but such means were beyond the means of private individuals.

Key distribution remained an open problem until the 1970s, when *public-key cryptography* was invented. Public-key cryptography uses one-way mathematical functions functions whose inverses, e.g. discrete logarithms, are very hard to calculate in such a way that encryption can be performed by anyone, using a *public key*, but decryption is only feasible for the owner of a *secret key*. The quintessential public-key cryptosystem is RSA, named for its authors: Rivest, Shamir and Adleman [16]. With such a system, encryption keys could be published openly, largely solving the key distribution problem.

The focus of this thesis, however, is stream ciphers, which use symmetric keys. Symmetric-key (or secret-key) cryptography uses the same, secret key for both encryption and decryption. Symmetric-key ciphers are still important, as public-key cryptography is very computationally complex, and is thus often used for the purposes of setting up a *session key* - a secret key that traditional, lower-complexity cryptosystems can use to provide confidentiality for a session. This is the premise behind systems such as PGP - Pretty Good Privacy [17].

2.1.2.3 One-Time Pad

During World War I, Vernam proposed the idea of a simple cipher that could not be broken: the one-time pad [18]. Shannon subsequently demonstrated in [12] that this cipher did indeed provide *perfect secrecy* - if the key is truly random, then intercepting ciphertext provides the attacker with no information about the plaintext.

The critical requirement for perfect security is that the set of possible keys be at least as large as the set of possible plaintexts. In a one-time pad, a long stream of random bits is generated and distributed to both communicating parties (e.g. an embassy's key could be encoded on optical tape and shipped in a diplomatic bag [9]). When a message is encrypted, each plaintext symbol is added to a symbol of key using Galois Field arithmetic, and that portion of key is discarded, never to be used again. Decryption occurs via the inverse process: each ciphertext symbol is added to the Galois Field inverse of an identical keystream symbol — which is afterwards discarded — to produce the original plaintext. If the symbol alphabet is in GF(2), then both encryption and decryption are simply the XOR operation.

Since there is as much key material as plaintext, and if that key material is truly random, then it is impossible to "break" the cipher. If the plaintext and key both have an alphabet of L symbols, then there are N^L possible plaintexts and N^L possible keys, where N is the number of symbols transmitted. From the ciphertext "GDIFBALDKRPDFZLSB" it is impossible to know which of the 17-letter plaintexts "MEETMEATNINETODAY", "MAXSMARTISAGENT86" or even "LOVEYOUSWEETHEART" is correct, as each of their corresponding keys is equally likely to be correct — as shown in Figure 2.2.

Because of the logistical costs of generating and distributing vast amounts of key material, the one-time pad is not used extensively outside of diplomatic and

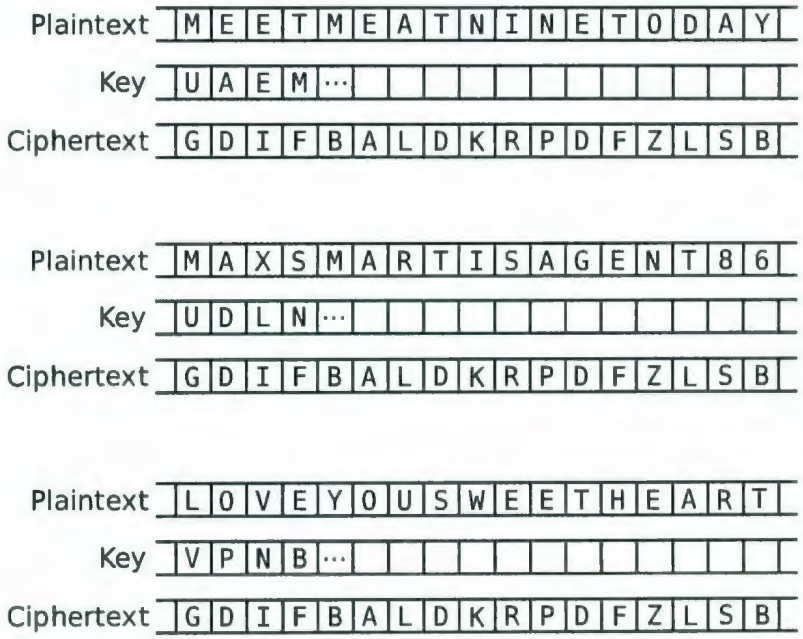

Figure 2.2: The one-time pad in operation

intelligence circles [9], but we will see in Section 2.1.2.5 how its principles are applied in many practical ciphers.

2.1.2.4 Block Ciphers

The last fifty years of symmetric-key cryptography have been dominated by the block cipher. A block cipher is a cipher that operates on fixed-size blocks of data (typically of 64 or 128 bits), transforming plaintext blocks into ciphertext blocks (encryption) or *vice versa* (decryption). An example is the Advanced Encryption Standard [1], a block cipher selected in 2001 to be an official standard of the US National Institute of Standards and Technology.

As shown in Figure 2.3, the unit of communication is a block of ciphertext. An eavesdropper cannot decrypt a block without the secret key, providing confidentiality, as a single bit difference between the correct key and the key guess will render the entire block undecryptable. There are several modes of operation for block ciphers,

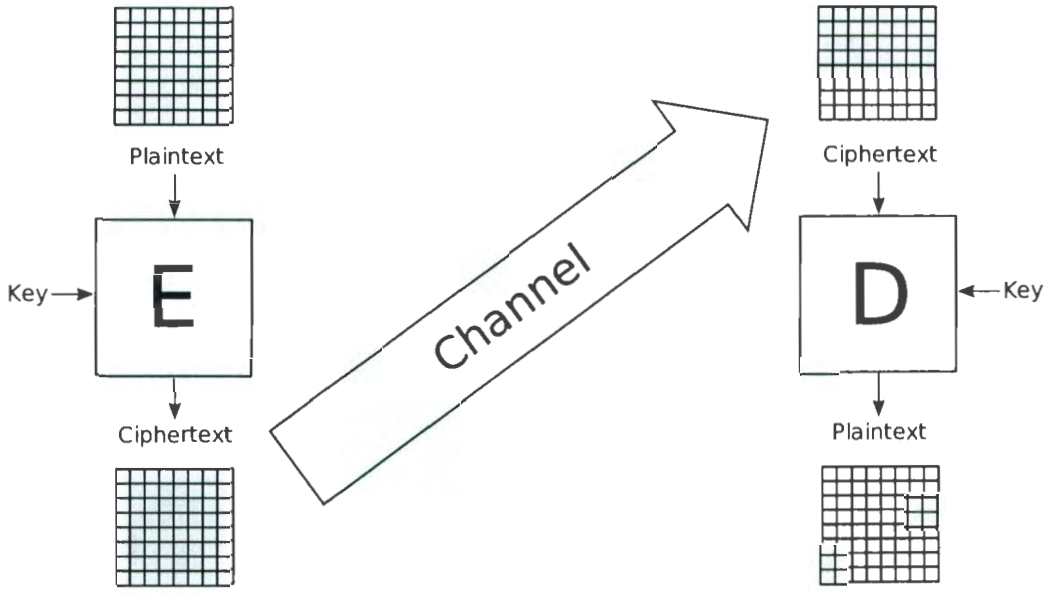

Figure 2.3: Block cipher operation

but all operate with two common characteristics:

- Complexity
  - Block ciphers are often large, complex hardware systems whose power usage can vary greatly, depending on input elements such as plaintext and secret key
- Key Usage
  - Because of the overhead associated with changing encryption keys (both in key management and cipher setup), block ciphers perform many encryptions/decryptions with a single key

While neither of these characteristics interfere with block ciphers' ability to operate securely in a theoretical sense, they will become important when we discuss implementation attacks in Section 2.2.

2.1.2.5 Stream Ciphers

Another class of symmetric-key ciphers, typically associated with resource-constrained environments, is the stream cipher. This class of cipher, shown in Figure 2.4, uses identical encryption and decryption modules, each of which produces a very long (e.g. 2^{80} bits) pseudo-random stream of symbols that is parametrized or seeded by a public bit vector – the *initialization vector* – and a secret key. This pseudo-random stream is called the *keystream,* and it is an approximation of the one-time pad described in Section 2.1.2.3. An example of such a cipher is Trivium [5], part of the eSTREAM portfolio of stream ciphers. Trivium and the eSTREAM portfolio will be addressed in detail in Chapter 6.

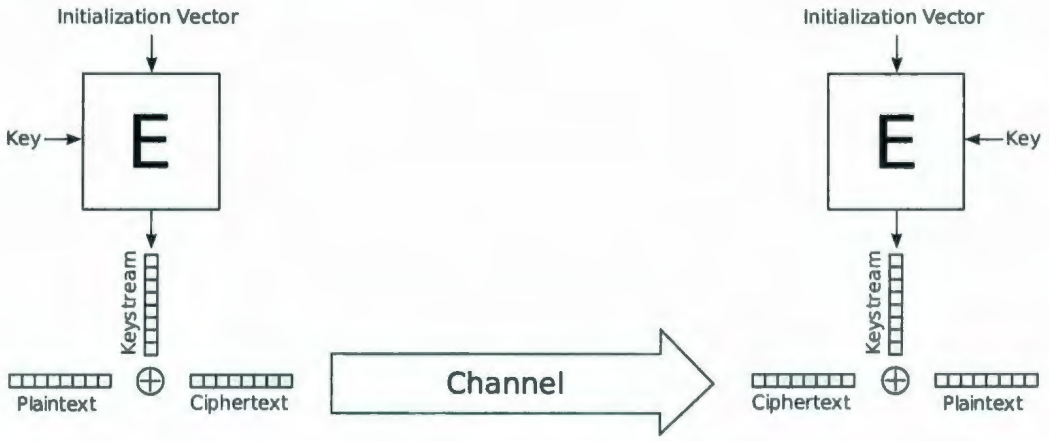

Figure 2.4: Stream cipher operation

As in the case of the one-time pad, the keystream is added to the plaintext using Galois Field arithmetic – typically in $GF(2)$, which is the binary XOR – to produce a ciphertext stream. The ciphertext is transmitted through the communication channel, where it is added to another keystream to produce plaintext again. If the secret key at the transmitter and receiver are identical, then their keystreams will be as well, so the original plaintext will be recovered. If the keys differ, however, even by a single

bit, then the two keystreams will be very different, and the original plaintext will not be recovered from the ciphertext.

Two characteristics of stream ciphers that will become important in Section 2.2 are:

- Complexity
  - Stream ciphers are typically very simple systems, and the power used by their hardware implementations does not vary as greatly as that of block ciphers
- Key Usage
  - The internal state of many stream ciphers (e.g. Grain [3] and Trivium [5]) is initialized with the secret key, but continually changes in such a way that key information is "mixed in" to the state, so no two bits of keystream are generated from the same internal state.

2.2 Side Channel Analysis

When cryptographers design a new cipher, the approach that they take is often very abstract; many cryptographers would agree with [19] when it says that "essentially a block cipher is a keyed permutive mapping (encryption) together with its inverse (decryption)". Such an abstract, mathematical model of a cipher appears in Figure 2.5.

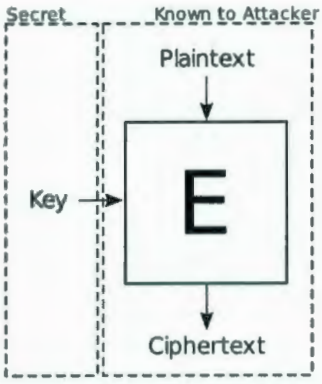

Figure 2.5: An abstract model of a cipher

In this model, one assumes that the cipher algorithm is known to potential attackers, and possibly even pairs of plaintext (data to be encrypted) and ciphertext (encrypted data). The key, which parametrizes the cipher, is not known to the attacker; it is this key that the attacker attempts to find using mathematical relationships between the plaintext and ciphertext.

No cryptographic function, however, exists as merely an abstract algorithm; it is not useful until it has been implemented in hardware or software. The operation of an actual cryptographic cipher implementation can yield information about its internals that the designer did not expect or plan for. This information is said to flow through "side channels", which include:

- power usage - how much power a device uses
- electromagnetic radiation - how much power a device radiates
- execution time - how long an operation takes
- response to faults - how the device reacts to intentionally-induced errors

A more realistic cipher model, which incorporates these side channels, is shown in Figure 2.6. Careful measurement and analysis of the signals in these channels can

capture information "leaking" out of the cipher. The techniques that cryptographers use to exploit these correlations are collectively known as side channel analysis [6].

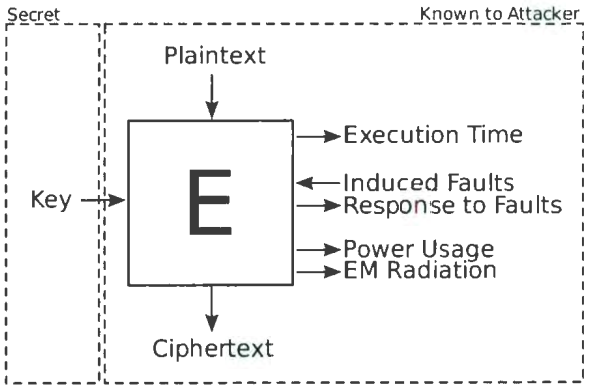

Figure 2.6: A more realistic model of a cipher

Most techniques of side channel analysis require some level of physical access to the cryptographic device under attack. This was once an implausible assumption, but as cryptography moves from secure server rooms to notebook PCs to smart cards in our wallets, it becomes an increasingly realistic and important component of security threat models.

2.2.1 Timing Analysis

Timing Analysis, first demonstrated in [20], uses very precise measurement of algorithm execution time in order to infer bits of data upon which the algorithm is operating. At first glance, there may not seem to be a correlation between these two things, but in fact, execution time can be related to:

- key- or data-dependent branch instructions
- cache hits

- long processor instructions (e.g. multiplication)

For instance, in public-key cryptography, mathematical operations are often performed on very large (512- or 1024-bit) integers. In order to improve performance, many public-key implementations will use conditional (if/else) software instructions that depend on key or data bits, as in Figure 2.7.

```
for(i = ...)
    if(input & (1 << i) != 0)
        output << 1
        output *= input
```

Figure 2.7: Data dependent branching

This pseudocode, which could be part of a large-integer exponentiator, has two lines that only execute if an particular `input` bit is 1. If the `input` and `output` variables in this pseudocode are 512-bit integers, there will be a very significant difference in execution time depending on how many bits of `input` are 1.

Timing attacks have been applied to block ciphers such as RC5 [21] and are even applicable to careless implementations of the Advanced Encryption Standard [22].

2.2.2 Fault Analysis

Fault analysis attempts to induce small (usually single-bit) errors into a cryptographic computation [23]. The resultant ciphertext can be compared to the ciphertext that would emerge if there were no error, and the differences between the two can yield insight into internal bits that should be secret.

Fault Induction In [23], the theoretical effectiveness of fault analysis was demonstrated, but no concrete results against actual, physical cryptosystems were given. Rather, it was assumed the attacker had the power to cause bits in the system to "flip" from 0 to 1 or 1 to 0, as required by the attack. This model was first given in [24], but more recent work has demonstrated practical fault induction.

More recently, [25] showed that, if such faults can be generated, then complete AES keys can be recovered using as few as 128 faulty encryptions. Such faults have been demonstrated in [26], where such commonplace equipment as lenses and camera flashes were used to set individual bits of microcontroller memory with precise timing. This does require opening the packaging of the chip, however. More insidious is the attack in [27], which claims that arbitrary memory bits may be set or reset by an attacker. If practical, such attacks would be very difficult to defend against.

Clock Glitches Side channel analysis is often used against the "smart cards" that control mobile phones, pay TV and satellite receivers, and in some places, even power meters. These devices are very small, and they usually have no on-board power or clock sources; they rely on connected equipment, and this can leave them vulnerable to clock glitches.

In [28], it was demonstrated that by sending a 20MHz pulse to a smart card which operates at 5MHz, faults could be introduced in the system. In fact, it was shown that individual instructions executing on a microcontroller could be bypassed by way of such faults. Hence, the number of encryption rounds could be reduced, making cryptanalysis trivial.

Chip Rewriting In [28], it was shown that single ROM bits could be overwritten with a laser cutter microscope. Again, this could be used to attack data, but it is

even more effective to attack the program, reducing encryption rounds to one or two and allowing for trivial cryptanalysis of the system.

In [29], a focused ion beam is used to cut traces inside of a microchip, or even to lay down new traces. This equipment is expensive, on the order of millions of dollars, but it can be rented for much less. Against such a powerful adversary, it is difficult to imagine countermeasures that would have more efficacy than slowing the attacker down. Indeed, [29] contains confirmations from "a senior agency official" and "a senior scientist at a leading chip maker" that the contents of a microchip *cannot* be kept secret indefinitely from a skilled, equipped and motivated attacker.

2.2.3 Power Analysis & Electromagnetic Analysis

The power usage and electromagnetic radiation side channels are closely related. Power analysis attempts to find internal secrets by correlating them with how much power an electronic device is consuming [30]. Electromagnetic analysis attempts to find correlations with the power that a device is radiating, either from the entire system or from a specific location on a chip [31]. The experimental setups involved with both can be very simple, as shown in Figure 2.8.

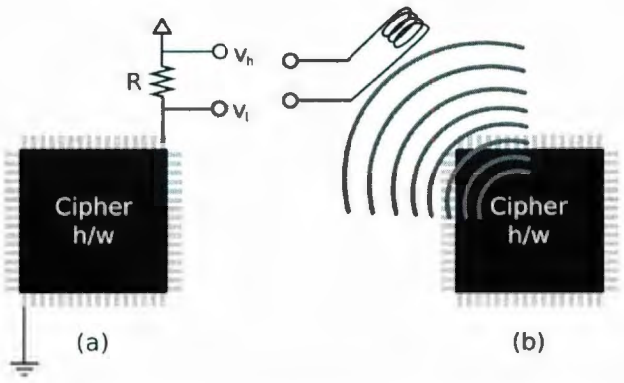

Figure 2.8: Power analysis and electromagnetic analysis

Both forms of SCA have their place: power analysis does not have to contend with high levels of ambient noise, but electromagnetic analysis allows the attacker to focus on a specific part of the device under attack - concentrating on cryptography and ignoring unrelated hardware.

In Figure 2.8(a), we see a hardware device with a small resistor inserted between its V_{CC} terminal and the actual V_{CC} supply. The power consumed by such a device can easily be calculated as

$$p(t) = v(t) \cdot i(t) = v_l(t) \cdot \frac{v_h(t) - v_l(t)}{R}. \tag{2.2}$$

Figure 2.8(b) shows a small electromagnetic probe receiving radiation from the cryptographic device. In both cases, deep memory oscilloscopes are used to record the power being consumed or emitted.

A capture of the power usage over a complete cryptographic operation is referred to as a *power trace*. While it is possible to correlate these traces with internal secrets, it is often made difficult by a very low signal-to-noise ratio (SNR). An attacker may be interested in whether a particular flip-flop in a cryptographic device changes from 0 to 1 or from 1 to 0, but there may be thousands of flip-flops in the device, each of which has just as much effect on overall power usage as the bit being attacked. In order to overcome this SNR problem, increasingly sophisticated methods of analysis are being developed.

Simple Power Analysis (SPA) The first, and simplest, method of power analysis can be used to analyse the power consumption of microcontroller-based software implementations. Naïve implementations of ciphers may incorporate software tech-

niques like branching instructions that depend on key bits. Such techniques build a very high correlation between power usage and individual key bits, as the difference in power traces where a branch was or was not taken can be obvious even to the naked eye. In these situations, an attacker may be able to simply examine the power trace and pick out key bits by observing whether or not power-intensive instructions following branch instructions were executed.

For instance, Figure 2.9 shows a current trace from a DES operation. Arrows point to dips in current characteristic of rotation functions, clearly showing the attacker that one rotation occurred in one round and two in the next. Since the number of rotations are key-dependent, being able to count rotations gives the attacking information about the secret key.

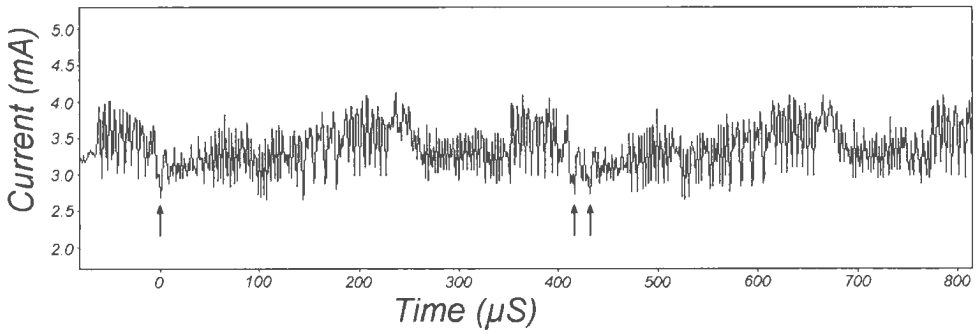

Figure 2.9: Simple Power Analysis [30]

This technique is called simple power analysis (SPA) [30], and it relies on a relatively high SNR. It has been used practically, as shown in the SPA attack against DES in [30], but it is a very simple matter for a cipher implementation to counteract this threat: all that is required is for the designer and/or implementer to ensure that branching instructions do not depend on key bits. This may increase execution time, but avoiding key-dependent shortcuts means that the high SNR necessary for Simple Power Analysis is not attained, and so SPA is rendered ineffective against the implementation.

Differential Power Analysis (DPA) The differential power analysis (DPA) technique [30], which is suitable for attacking hardware and software systems, attempts to overcome low SNR by analysing many power traces which use the same key information and statistically testing hypotheses concerning internal key bits. The greater the number of traces used, the higher the resultant SNR, but there is a caveat: the attacker may have to gather thousands of power traces from the device being attacked, which may be difficult to acquire without arousing suspicion.

It has been shown that DPA can be used in practical attacks on real cryptosystems involving block ciphers such as DES [30], but it is often not effective against stream ciphers [32]. The reason is that, as stated in Section 2.1.2.5, the secret key often only exists in the cipher's internal state for a few clock cycles, so the analysis described in [30] does not work unless the attack can obtain many traces from cipher re-keyings. Obtaining power traces of several thousand re-keyings, with the same key, from an in-production device can be prohibitively difficult; this type of key re-use is purposefully avoided in most protocols to minimize susceptibility to traditional cryptanalysis.

Template Attacks A newer approach to the SNR problem, which removes the requirement for obtaining thousands of power traces from the device under attack, is referred to as a template attack, as presented in [7]. We will present this attack in more detail in Chapter 3.

The template attack takes a two-step approach to power analysis:

1. Template Preparation

   A cryptographic device identical to the one under attack is acquired[1] and *tem-*

[1] When we say that a device has been *acquired*, it may be either constructed or otherwise obtained (e.g. by purchasing the same model of device). It is very realistic to assume that this is practical, as many cryptographic systems are built with standard commercial components, such as ISO-standard smart cards [33]; only the secret keys are not available to the attacker.

plates are built, which are multivariate Gaussian models of the noise associated with particular guesses at key bits.

2. Actual Attack

   In this step, a *single* power trace is collected from an actual device in use and, for each template, the probability that it belongs to that template is calculated.

Communications engineers will see that this approach is analogous to using matched filters to resolve received signals. The technique shows much promise, having been used to successfully attack a microcontroller-based implementation of the stream cipher RC4 [7]. To date, however, the template approach has not been applied to hardware-based implementations of stream ciphers.

2.3 Summary

Cryptography is an important part of daily life in our networked world. One of the most fundamental tools of cryptography is the cipher, which provides confidentiality for parties wishing to communicate in the presence of an eavesdropping threat. These ciphers may be attacked through methods of cryptanalysis, which may be classified as ciphertext-only, known-plaintext, chosen-plaintext or implementation attacks.

Ciphers can be categorized as symmetric-key or asymmetric-key. Among symmetric-key ciphers, which this thesis is concerned with, there are two broad categories: block ciphers and stream ciphers. Stream ciphers attempt to approximate the one-time pad which has perfect secrecy by generating long pseudo-random keystreams from secret keys. Encryption consists of adding this keystream to the plaintext stream, and decryption consists of adding it to the ciphertext stream. One important stream cipher today is Trivium, which will be considered in detail in Chapter 6.

Side channel analysis is a broad term for a class of implementation attacks that attempt to extract secret information via careful measurement of various physical characteristics, such as execution time, response to induced faults and the power consumed or radiated by a system. These measurements can be analysed by inspection, partitioning-based statistics and multivariate Gaussian analysis. The latter approach is called a template attack, and its details are the subject of Chapter 3.

Chapter 3

Template Attacks

The template attack is a powerful method for extracting secret information from cryptographic hardware. Chari et al. claimed in [7] that it is "the strongest form of side channel attack possible in an information theoretic sense" (under certain assumptions concerning the nature of the side channel – see Section 3.3.2). The attack is effective when physical access is limited - an attacker needs just one power trace from the device under attack - and is even effective against stream ciphers, which resist traditional power analysis techniques such as simple power analysis and differential power analysis (see Section 4.1.2 for more information on SPA and DPA).

In this thesis, we focus on the power usage side channel, but template attacks are not inherently limited to power analysis; they can also be applied to other side channels such as electromagnetic radiation and execution time.

3.1 Attack Overview

Template attacks operate according to the principles of signal detection, and they are optimal in the same sense that the matched filter approach is the optimal technique

availabile in its domain. Unlike traditional power analysis techniques, template attacks are split into two steps, only one of which actually requires access to the device under attack [7]:

Template Preparation Analogous to the preparation of matched filters, this step of the attack involves the construction of *templates* - collections of statistical information that will later be used to recognize secret parameters to cryptographic operations.

Cryptographic hardware, similar or identical to the hardware to be attacked[1], is run through the initial stages of operation many times – hundreds or even thousands of times – with certain parameters (e.g. several bits of the secret key) set to known values. Other parameters are permitted to vary randomly, so that, as the numer of sample traces increases, the template comes to reflect only that side channel information which depends on the parameters set above.

The side channel (power usage, timing, etc.) is measured carefully for each operation that the attacker chooses to target (e.g. one for each of the 16 possible combinations of four particular key bits). Statistics are compiled, and a set of this statistical data – the template – is generated. The set of templates – one per operation – is then used in the second step of the attack.

Template Application In this second step, the attacker captures traces from the device under attack – just one sample can be sufficient, though additional traces can increase the probability of success. These traces are then compared to each template to determine which template each trace is "closest" to (see Section 3.9 on page 30 for

[1]The assumption is that the attacker has access to similar or identical hardware, but this assumption is very realistic. From smart cards to tamper-resistant PC cards and associated libraries, standard hardware and software is available on the open market for the would-be attacker to legally acquire.

a precise definition of closeness). The operation associated with this template (e.g. a guess at a portion of the secret key) is assumed to be correct, and the attack can repeat on other parameters, such as other key bits.

3.2 Attack Details

The theory of template attacks is rooted in the statistics of multivariate normal distributions, as presented in [34] and [35]. It is assumed that side channel measurements can be characterized by such a distribution; the validity of this assumption is considered in Section 3.3.2.

3.2.1 The Multivariate Normal Distribution

Suppose we have a random variable which is an $n \times 1$ vector. Since it is both random and a vector, it will be represented here by $\vec{\mathbf{x}}$. For the purposes of the template attack, this random variable could be a set of power or timing measurements. The probability distribution of this vector can be represented by a *mean vector* and an independant *covariance matrix*. The mean vector is defined as:

$$\vec{\mu}_{\vec{\mathbf{x}}} = E\{\vec{\mathbf{x}}\} = \begin{bmatrix} E\{\mathbf{x}_1\} \\ E\{\mathbf{x}_2\} \\ \vdots \\ E\{\mathbf{x}_n\} \end{bmatrix}, \tag{3.1}$$

where $\vec{\mathbf{x}}$ is the $n \times 1$ random variable, $\mathbf{x}_i$ is an element in the random variable (e.g. a single power or timing value), $E\{\}$ is the expectation operator and $\vec{\mu}$ is the mean vector. The covariance matrix of the random variable is:

$$\Sigma_{\vec{\mathbf{x}}} = \operatorname{cov}(\vec{\mathbf{x}}) = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1n} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{n1} & \sigma_{n2} & \cdots & \sigma_{nn} \end{bmatrix}, \tag{3.2}$$

where $\sigma_{ij} = E\{(\mathbf{x}_i - \mu_i)(\mathbf{x}_j - \mu_j)\}$ is the covariance of i-th and j-th elements of the $n \times 1$ random variable $\vec{\mathbf{x}}$, and $\mu_i = E\{\mathbf{x}_i\}$.

We may now express the distribution's probability density function (PDF) as

$$f_{\vec{\mathbf{x}}}(\vec{x}) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_{\vec{\mathbf{x}}}|}} e^{(\vec{x}-\vec{\mu}_{\vec{\mathbf{x}}})^T \Sigma_{\vec{\mathbf{x}}}^{-1} (\vec{x}-\vec{\mu}_{\vec{\mathbf{x}}})}, \tag{3.3}$$

where $\vec{x}$ is an $n \times 1$ vector, $\vec{\mu}_{\mathbf{x}}$ is the distribution's mean vector, $\Sigma_{\mathbf{x}}$ the covariance matrix and $|\Sigma_{\mathbf{x}}|$ the determinant of the covariance matrix. This PDF is the general (multi-dimensional) form of the well-known univariate Gaussian PDF:

$$f_{\mathbf{x}}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{3.4}$$

where $\mathbf{x}$ is a real value, μ is the distribution's mean value and σ is its standard deviation.

Our side channel values (e.g. the power usage of the cipher when, say, the last four key bits are 0110) can be represented by such a distribution, with each element of the vector $\vec{x}$ being a different power measurement (e.g. power at time $t = 20ns$, $t = 40ns$, etc.).

3.2.2 Maximum Likelihood Estimators

The *template* in a template attack is a maximum-likelihood estimation of $\vec{\mu}$ and Σ for a set of possible side channel values, e.g. the power usage of a cipher for a particular

Operation	Description
$O^{(1)}$	Cipher with last four key bits 0000
$O^{(2)}$	Cipher with last four key bits 0010
$O^{(3)}$	Cipher with last four key bits 0011
⋮	⋮
$O^{(16)}$	Cipher with last four key bits 1111

Table 3.1: Stream ciphering operations

subset of key bits. After collecting a large number of side-channel values – hundreds or thousands – we can calculate the *maximum likelihood estimations* of the actual mean and covariance matrix.

Using the nomenclature of [7], we first identify a number of *operations* that we wish to study. If the identified operations are microprocessor instructions, then the template attack will enable an attacker to identify when particular instructions execute. In our case – attacking stream cipher hardware – an operation will be the execution of a cipher with a particular subset of known key bits. For instance, the initial round of attack may involve 16 operations, given in Table 3.1.

Again using the nomenclature of [7], we will now define several values important to the attack:

K The number of operations we wish to study

L The number of sample traces we will measure per operation

N The number of data points in each sample trace

Note that, for reasons given below, L should be greater than or equal to N (and in practice, $L > 2N$).

We may organize the sample values into K matrices, one per operation, each

denoted $\mathbf{S}^{(k)}$, where k represents the operation, and containing L vectors of N points of side channel data:

$$\mathbf{S}^{(k)} = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1N} \\ s_{21} & s_{22} & \cdots & s_{2N} \\ s_{31} & s_{31} & \cdots & s_{3N} \\ s_{41} & s_{41} & \cdots & s_{4N} \\ \vdots & \vdots & \ddots & \vdots \\ s_{L1} & s_{L2} & \cdots & s_{LN} \end{bmatrix}. \tag{3.5}$$

Having generated the matrix $\mathbf{S}^{(k)}$, we may estimate the operation's mean vector. The maximum-likelihood estimation of the true mean vector is simply the *sample mean vector*, an arithmetic average of each sample trace. Let k be the number of the operation being studied (in the range $[1, K]$) and $\vec{s}_j^{(k)}$ be the j-th column of $\mathbf{S}^{(k)}$. The arithmetic average of all L samples of side channel measurements for operation $O^{(k)}$ is a vector of N values, represented by $\hat{\mu}^{(k)}$ and given by

$$\hat{\mu}^{(k)} = \begin{bmatrix} \frac{1}{L}\sum_{i=1}^{L} s_{i1} \\ \frac{1}{L}\sum_{i=1}^{L} s_{i2} \\ \vdots \\ \frac{1}{L}\sum_{i=1}^{L} s_{iN} \end{bmatrix}. \tag{3.6}$$

Once we have calculated an operation's sample mean vector, we may calculate the *noise vector*, $\hat{n}_i^{(k)}$ for each sample trace for operation $O^{(k)}$, $\vec{s}_i^{(k)}$:

$$\vec{n}_i^{(k)} = \vec{s}_i^{(k)} - \hat{\mu}^{(k)} = \begin{bmatrix} s_{i1} - \hat{\mu}_1 \\ s_{i2} - \hat{\mu}_2 \\ \vdots \\ s_{iN} - \hat{\mu}_N \end{bmatrix}. \tag{3.7}$$

The noise vectors of all L sample traces are used to calculate the maximum-likelihood estimate of the operation's covariance matrix:

$$\hat{\Sigma}^{(k)} = \frac{1}{L} \sum_{i=1}^{L} \left(\vec{s}_i^{(k)} - \hat{\mu}^{(k)} \right) \left(\vec{s}_i^{(k)} - \hat{\mu}^{(k)} \right)^T = \frac{1}{L} \sum_{i=1}^{L} \left[\vec{n}_i^{(k)} \left(\vec{n}_i^{(k)} \right)^T \right]. \tag{3.8}$$

This $N \times N$ matrix is our maximum-likelihood estimate of the operation's covariance matrix $\Sigma^{(k)}$, and the template for operation $O^{(k)}$ is $\left(\hat{\mu}, \hat{\Sigma} \right)$.

3.2.3 Signal Classification

Having built K templates, one per operation, we can classify any signal $\vec{s}$ by calculating that signal's noise vector, $\vec{n}$, and the Mahalanobis distance between that noise vector and each operation's mean, $\vec{\mu}^{(k)}$ [36]:

$$D_{\mathrm{M}}^{(k)}(\vec{n}) = \sqrt{\vec{n}^T \left(\hat{\Sigma}^{(k)} \right)^{-1} \vec{n}}, \tag{3.9}$$

where $\hat{\Sigma}^{(k)}$ is the sample covariance matrix for operation $O^{(k)}$. Having calculated $D_{\mathrm{M}}^{(k)}$ for each of the K templates, we may classify the signal $\vec{s}$ as belonging to the operation $O^{(k)}$ which has the smallest Mahalanobis distance $D_{\mathrm{M}}^{(k)}(\vec{n})$.

The method introduced in [7] attempts to effect classification by using the multivariate Gaussian PDF directly as a probability:

> "... the noise probability distribution is given by the N dimensional

multivariate Gaussian distribution $p_{\mathbf{N}_i}(\cdot)$ where the probability of observing a noise vector $\mathbf{n}$ is:

$$p_{\mathbf{N}_i}(\mathbf{n}) = \frac{1}{\sqrt{(2\pi)^N |\Sigma_{N_i}|}} \exp\left(-\frac{1}{2}\mathbf{n}^T \Sigma_{N_i}^{-1} \mathbf{n}\right),$$

where $|\Sigma_{N_i}|$ denotes the determinant of Σ_{N_i} and $\Sigma_{N_i}^{-1}$ is its inverse." [7]

This is not strictly valid, as a point on a PDF is not a probability. The probability of a point on a continuous distribution is vanishingly small, as probabilities are obtained by integrating under a PDF and the area underneath a point is infinitesimal.

While the nomenclature is not precise, the method does work — it is concerned with ratios of "probabilities" rather than the probabilities themselves. Indeed, though a value of a point on the PDF may be much greater than 1, a ratio-based comparison of PDF values can be an effective classification mechanism.

Given Equation 3.9, we see that the PDF from [7] can be represented as:

$$f^{(k)}(\vec{n}) = \frac{1}{\sqrt{(2\pi)^N \|\mathbf{S}^{(k)}\|}} e^{-\frac{1}{2}\left(D_M^{(k)}(\vec{n})\right)^2} = \frac{1}{\sqrt{(2\pi)^N \|\mathbf{S}^{(k)}\| \, e^{\left(D_M^{(k)}(\vec{n})\right)^2}}}. \tag{3.10}$$

The ratio between PDF values for a given noise vector and two operations, $O^{(k_0)}$ and $O^{(k_1)}$, is:

$$\begin{aligned} \frac{f^{(k_0)}(\vec{n})}{f^{(k_1)}(\vec{n})} &= \sqrt{\frac{(2\pi)^N \|\mathbf{S}^{(k_1)}\|}{(2\pi)^N \|\mathbf{S}^{(k_0)}\|} \cdot \frac{e^{\left(D_M^{(k_1)}(\vec{n})\right)^2}}{e^{\left(D_M^{(k_0)}(\vec{n})\right)^2}}} \\ &= \sqrt{\frac{\|\mathbf{S}^{(k_1)}\|}{\|\mathbf{S}^{(k_0)}\|} e^{\left(D_M^{(k_1)}(\vec{n})\right)^2 - \left(D_M^{(k_0)}(\vec{n})\right)^2}}. \end{aligned}$$

Since e^x, x^2 and $\sqrt{x}$ are monotonically increasing functions with respect to x (where $x \geq 0$), we see that if $\left\|\mathbf{S}^{(k_0)}\right\| = \left\|\mathbf{S}^{(k_1)}\right\|$, then choosing the operation whose PDF value is largest is equivalent to choosing the operation whose Mahalanobis distance is smallest. Our experiments have shown that, while $\left\|\mathbf{S}^{(k_0)}\right\|$ may not be equal to $\left\|\mathbf{S}^{(k_1)}\right\|$, they are typically on the same order of magnitude, whereas $e^{D_M^{(k_0)}(\vec{n})}$ and $e^{D_M^{(k_1)}(\vec{n})}$ often differ by orders of magnatude. Thus, the method described in [7] is effective, even if the nomenclature is imprecise.

3.2.4 Template Masking

Computing large templates can be computationally intensive: for L sample traces and template size N, the computational complexity is in the class:

$$\begin{aligned} & \Theta\left(LN + LN + LN^2\right) \\ = \ & \Theta\left(LN^2\right). \end{aligned}$$

Fortunately, we are able to reduce the template size N though a process of *masking*, as not all points in a side-channel trace are equally significant. Often, the power used or emitted by a cryptographic device at the passing of a clock edge is more significant than the power used or emitted between clock pulses. Some clock cycles may be more significant than others, as the change in Hamming weight may vary more because of certain key bits than others at certain times.

We reduce the size of templates by selecting for the template only those points in the side-channel trace which are significant. For instance, we may select 32 data points out of 1600 measured, leading to a thousand-fold reduction in computational complexity. This selection is accomplished as follows:

1. The sample mean vector $\hat{\mu}^{(k)}$ is calculated for each operation $O^{(k)}$.

2. An overall mean vector $\bar{\mu}$ is calculated:

$$\bar{\mu} = \frac{1}{K} \sum_{k=1}^{K} \hat{\mu}^{(k)}.$$

3. The inter-operation standard deviation of the mean vectors is calculated:

$$\vec{\sigma} = \sqrt{\frac{1}{K} \sum_{k=1}^{K} (\hat{\mu}^{(k)} - \bar{\mu})^2}.$$

4. For a chosen value N (e.g. 32 points of interest), the N points with the greatest inter-operation standard deviation are selected for template generation.

Actual inter-operation mean and standard deviation vectors are shown in Figure 3.1. This data was derived from the experimental setup to be described in Chapter 4, and it illustrates just how significant differences can be among data points in the inter-operation standard deviation.

The upper graph shows the inter-operation mean vector. In this vector, we can see clear spikes of power usage whenever a clock edge occurs. This behaviour is common to all operations, and thus, it can be observed in the inter-operation mean. The lower graph is the inter-operation standard deviation vector. This vector shows us two important facts:

1. The greatest differences occur at clock edges.

2. The greatest differences occur early in the operations — before the secret key can "mix into" the cipher state.

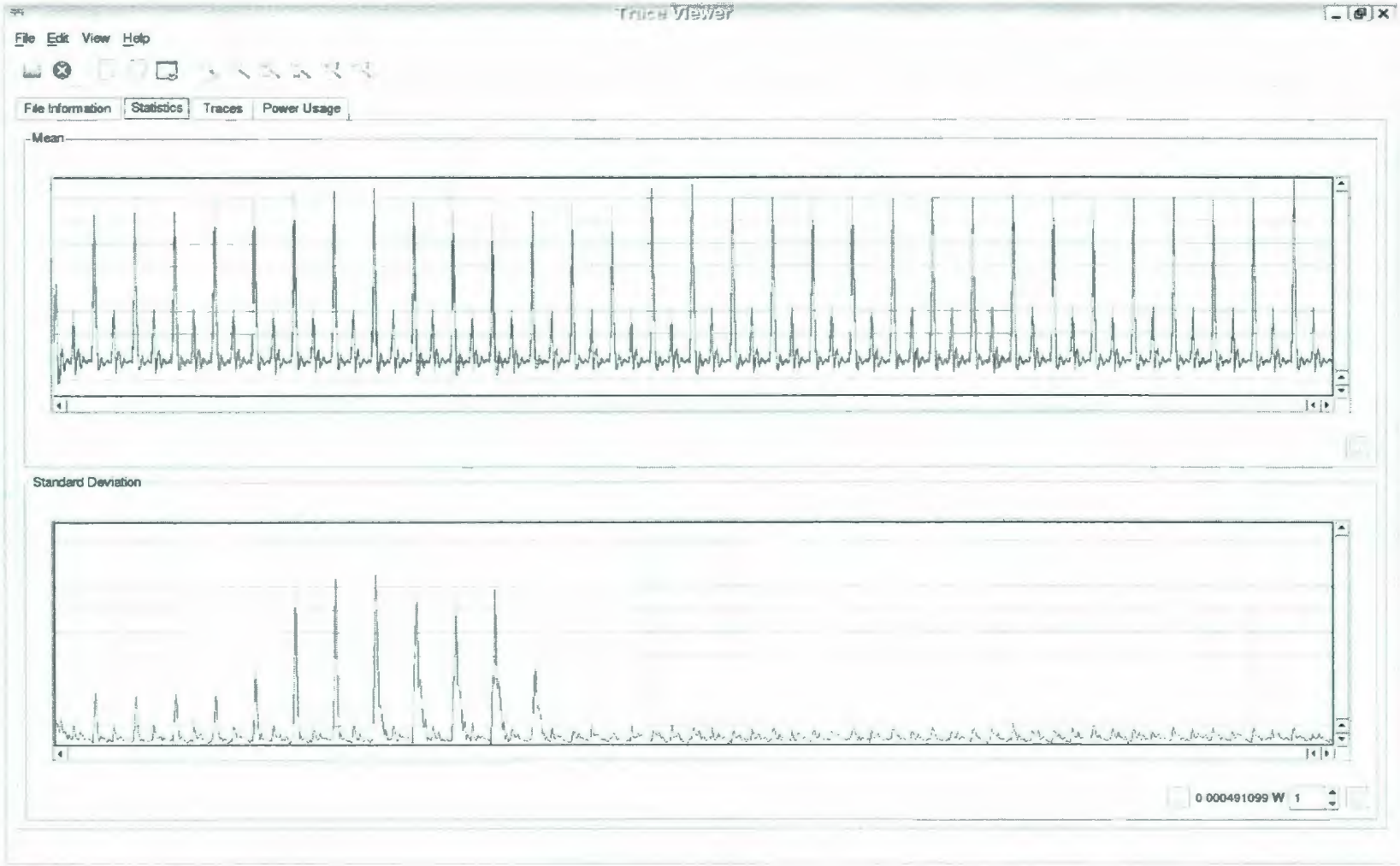

Figure 3.1: Inter-operation mean and standard deviation vectors for actual hardware

3.3 Attack Application

Template attacks have been applied in [7] against microcontrollers running the stream cipher ARC4 (the "Alleged RC4™", so called because the name "RC4" is still protected by trademark, though source code to produce data equivalent to RC4 has been available on the Internet since 1987). As a stream cipher, ARC4 is resistant to differential power analysis (see Section 2.2.3 on page 21), but is highly susceptible to template attacks.

3.3.1 Inapplicability of DPA

Differential power analysis, which can be applied quite successfully to block ciphers, is simply not applicable to most stream ciphers, including ARC4. The reason has to do with the persistence of secret key information. We now turn our attention to explaining this important distinction in detail.

DPA and Block Ciphers When DPA is applied against a block cipher, the attacker makes several *guesses* at a subset of the secret key, as shown in Figure 3.2 on the following page.

This figure shows a model of a block cipher with four encryption *rounds*, each having *S-boxes* (providing non-linear substitution), a *permutation* layer (providing linear diffusion) and a *key mixing* layer. This model is similar to the substitution-permutation network presented in [37], but with the addition of a key mixing layer between each round. The block on the left-hand side of the figure represents the *key scheduler*, which converts the secret key into several *round keys*, which are added via XOR in each round's key mixing layer. The secret key shown is 0xXXBXX7XX in hexadecimal, where an 'X' digit represents bits of key that are not part of the current

guess.

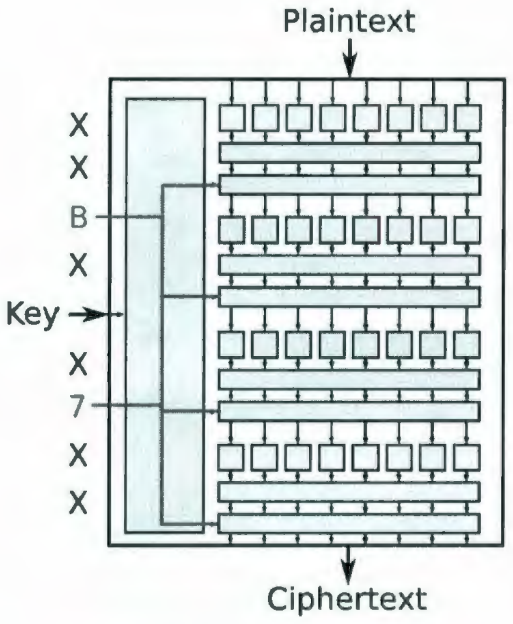

Figure 3.2: DPA key guesses

After making this guess, the attacker observes a large number of blocks of ciphertext and records power traces associated with their production. The secret key guess allows the attacker to work backwards through the cipher to determine a single bit whose value can be inferred if the key guess is correct.

In Figure 3.3, the guess of subkey bits, combined with knowledge of the key scheduling algorithm, permutation layer and S-box construction allows the attacker to evaluate a particular key bit entering an S-box in the last round of encryption. This bit is used to *partition* the side channel data into two sets: those for which the internal bit is 0, and those for which that bit is 1. If the attacker's key guess was incorrect, then we expect traces whose internal bit was 0 and 1 to be evenly distributed among the two sets. If, however, the guess is correct, then the partitioning will be correct, and there will be a significant difference between the averages of the two sets of side channel data. Resultant trace differences are shown in Figure 3.4, which shows four

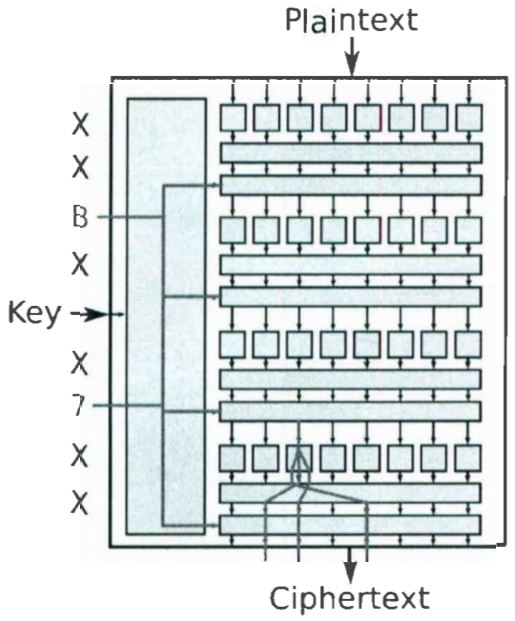

Figure 3.3: DPA bit guess

graphs:

1. A reference current trace (from which power may be derived, since $p = vi$)

2. A graph showing the difference between the averages of two samples trace sets, where the sets have been partitioned by a correct key guess

3. Two graphs showing differences between the averages of two samples trace sets each, where the sets have been partitioned by an incorrect key guess

The current spikes in the middle of the trace show that there is a material difference between the traces in the partitioned sets. That is, the initial key guess was correct, which made the partitioning effective. The attacker may now move on to another subset of the key, then another, until eventually the entire key is revealed. In this way, the secret key of a block cipher can be recovered in a linear way using a divide and conquer approach instead of the 2^N approach of exhaustive search.

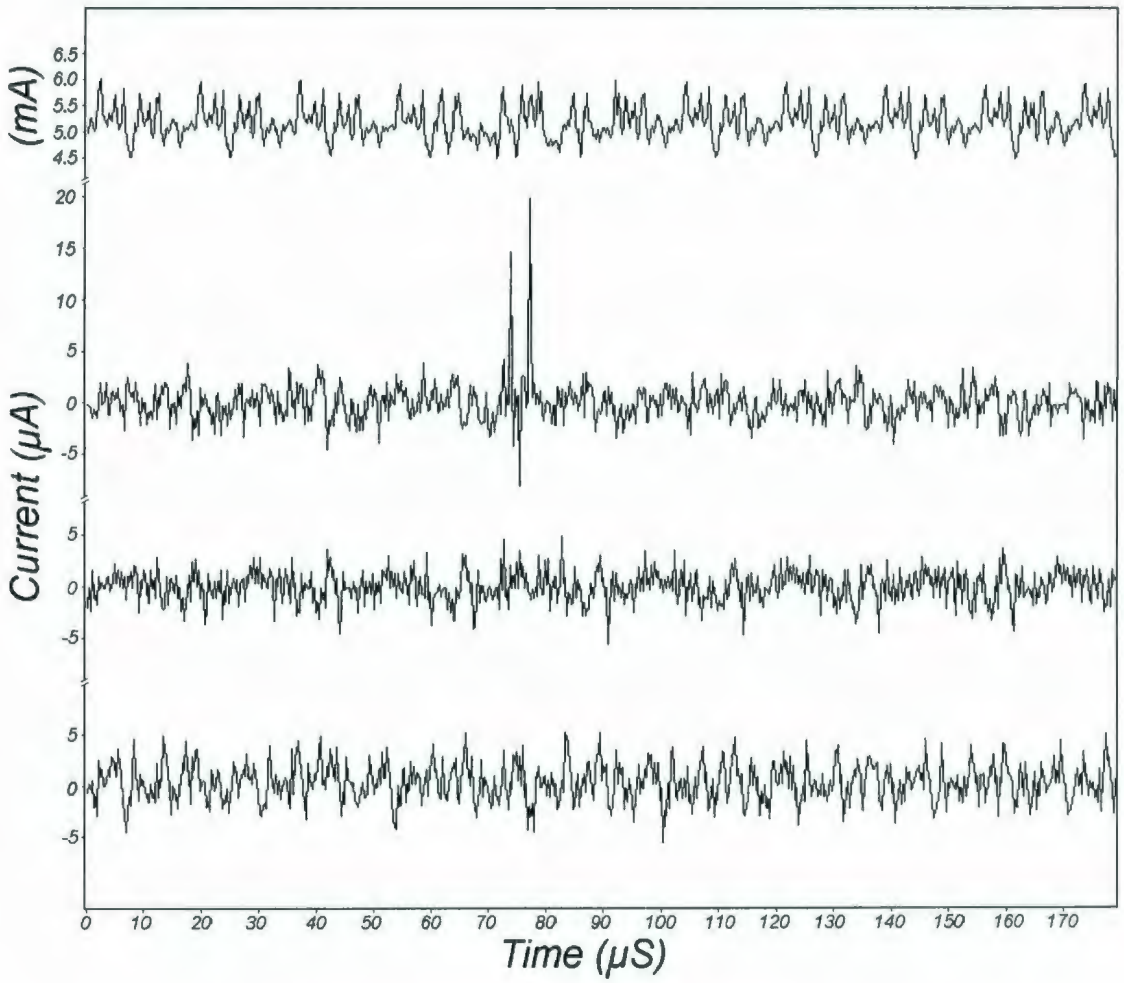

Figure 3.4: DPA trace differences [30]

DPA and Stream Ciphers Against stream ciphers, however, the differential power analysis technique is ineffective. In order to perform DPA, the attack must gather a large number of ciphertext/side channel pairs while the cipher's secret key remains constant. A stream cipher's internal state, however, is constantly changing; the secret key is only found at initialization, and by the time the keystream (and, thus, ciphertext) can be generated, the key has been mixed with an initialization vector (IV) so that the starting state is never the same twice.

A DPA attack might be effective if the attacker could obtain many ciphertext/-

side channel pairs from a device using a constant key and initialization vector, but cryptographic protocols are designed not to re-use IVs. Hence, an attack that does not rely on a persistent secret key being used by the device under attack is needed; one such attack is the template attack.

3.3.2 Applicability of Template Attacks

Figure 3.5 shows two graphs, each a difference between two power traces obtained from the key initialization phase of ARC4. These graphs show:

1. The difference between two power traces produced using the same key.

2. The difference between two power traces produced using different keys.

These graphs do not reveal the striking differences that an attacker might expect to see - differences like those in Figure 3.6. In fact, the first graph actually exhibits larger differences than the second, even though its keys were identical. This is due to the stochastic nature of power measurements: there is always noise associated with unrelated hardware or operations, so identical operations may be more dissimilar than different operations.

Figure 3.6 shows graphs for the same conditions - ARC4, the first graph for traces using the same key and the second graph for traces using different keys - but unlike Figure 3.5, the graphs show differences between averages of sets of traces and there are spikes of dissimilarity in the second graph which do not appear in the first (or in Figure 3.5). This dissimilarity reveals that the two graphs were produced by dissimilar operations, which in turn gives the attacker information about the secret key. It is this information that can be exploited by a template attack.

Template attacks assume that side channel information can be characterized by a multivariate Gaussian distribution. This characterization may introduce error

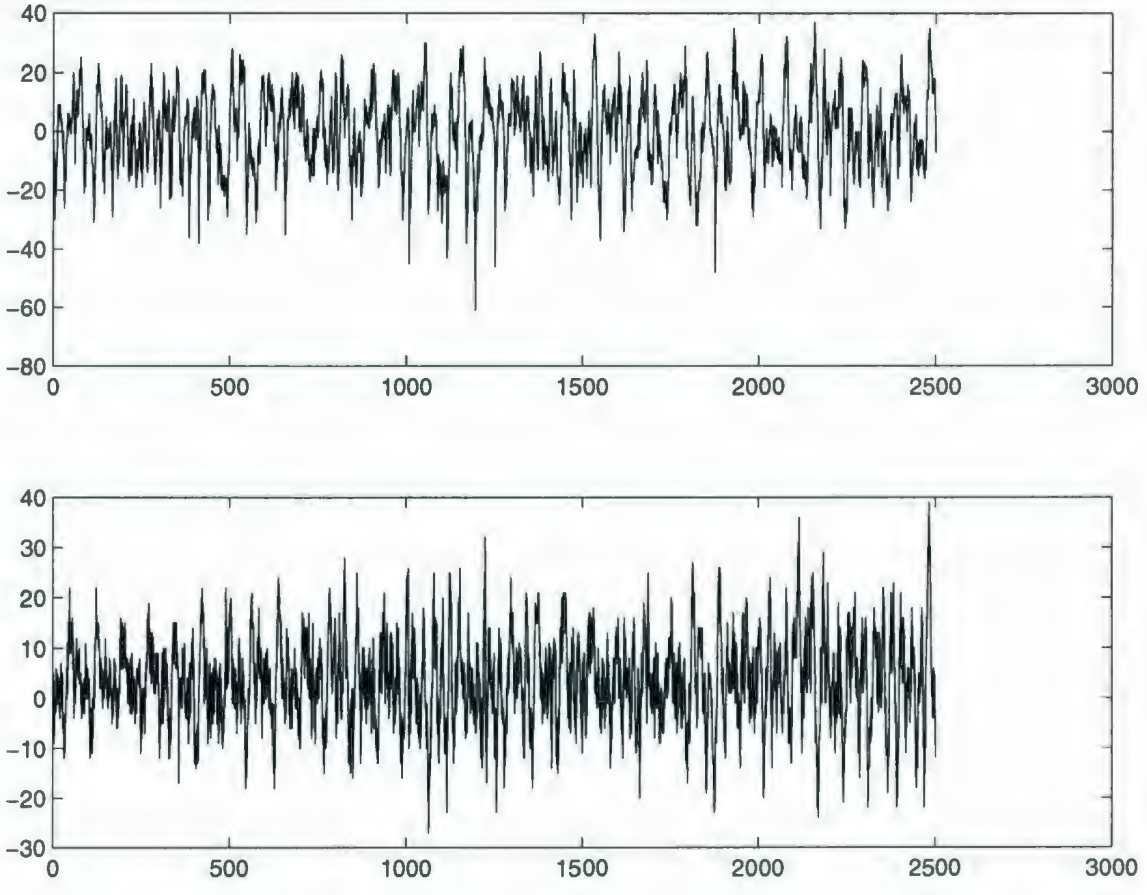

Figure 3.5: Difference of averages – 1 sample [7]

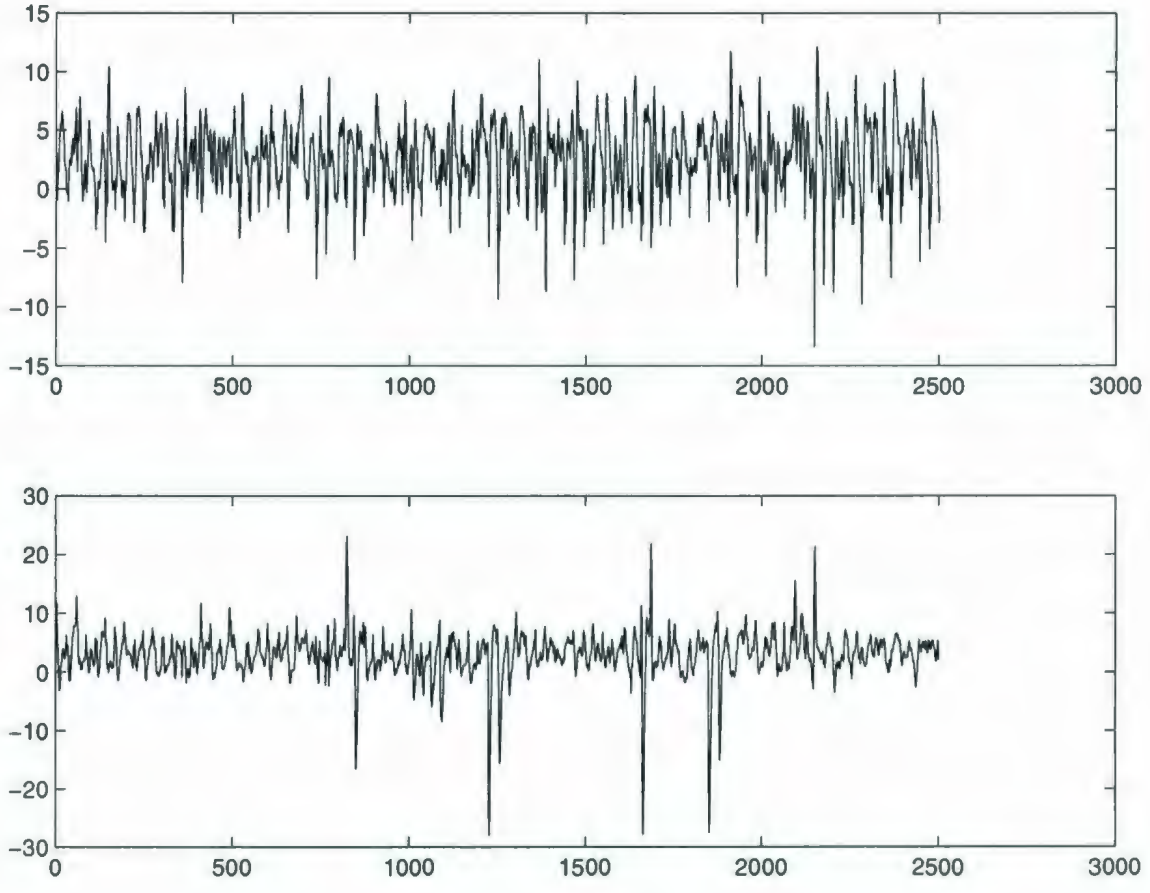

Figure 3.6: Difference of averages – 50 samples [7]

indeed, the interdependance of hardware elements suggests that the Gaussian distribution is not ideal – but the results in [7] show that it is a useful characterization. Further study could better model the characteristics of side channels, but such study is beyond the scope of this research.

3.3.3 Applicability to Hardware Implementations

Through template attacks, the classification success rate for ARC4 running on an embedded microcontroller was shown in [7] to be 98.1% – 99.3%, a clear success. Microcontrollers, while less resource-intensive than traditional CPUs, are still much more complex systems than the "pure" hardware that many ciphers are implemented in. The lower complexity of simply hardware implementations has been thought to be a defense against side channel analysis:

> "the only exposure for [fast cipher hardware] is the loading of the key bytes from EEPROM which usually leaks the hamming weight" [7]

The question that we set about answering was:

Can template attacks be used to differentiate secret keys on such a small scale of power usage as seen in digital hardware?

3.4 Summary

Template attacks, based on the theory of signal detection and classification, are very powerful side channel attacks. If the noise associated with the side channel is Gaussian, then this technique is in fact optimal [7].

Template attacks consist of two important stages:

1. Template Preparation

(a) The attacker collects a large number of sample traces from a cryptographic system identical to the one being attacked.

(b) These traces are used to build templates pairs, each consisting of a mean vector and a covariance matrix for a number of operations.

2. Template Application

(a) A small number of side channel traces (possibly just one) are taken from the device being attacked.

(b) The traces are classified using their Mahalanobis distances to each operation: for each trace, the operation whose Mahalanobis difference is smallest or whose probability density function is largest is selected as the producer of the trace.

Computational effort can be reduced by "masking" the trace data points used to construct the template; only those points with significant inter-operation standard deviation are used. This pruning process may reduce computation time a thousandfold, but often does not reduce the success rate of the attack by more than several percentage points.

Template attacks are applicable to stream ciphering systems, though older techniques such as differential power analysis (DPA) are not. This is because, unlike DPA, template attacks do not rely on a system using a persistent secret key. Template attacks have been shown to be very successful against microcontroller-based cryptographic implementations, but we will show in Chapter 5 that they are also effective against implementations in digital hardware.

First, however, we will describe the setup used in our experimentation. This experimental setup is the subject of Chapter 4.

Chapter 4

Experimental Setup

For this thesis, we wished to both apply the template attack by simulating hardware based on a model derived from actual physical characteristics – and also to apply the attack to measurements taken of the characteristics of physical hardware. These characteristics, especially power usage, were to be measured while real hardware performs cryptographic operations. Simulating, measuring and analysing these characteristics required:

1. A hardware platform on which we could run cryptographic operations
2. Sensitive measurement equipment with suitable amounts of memory
3. Software to simulate hardware and analyse physical measurement data

4.1 SCAB - Side Channel Analysis Board

The Side Channel Analysis Board (SCAB), shown in Figure 4.1, is a development board intended to facilitate the study of side channel attacks (SCA) against cryptographic hardware. It was developed for the purpose of the research contained in this

thesis, but its long-term objective is to provide security researchers with a platform on which any algorithm can run - thanks to reconfigurable hardware - and many physical properties can be studied.

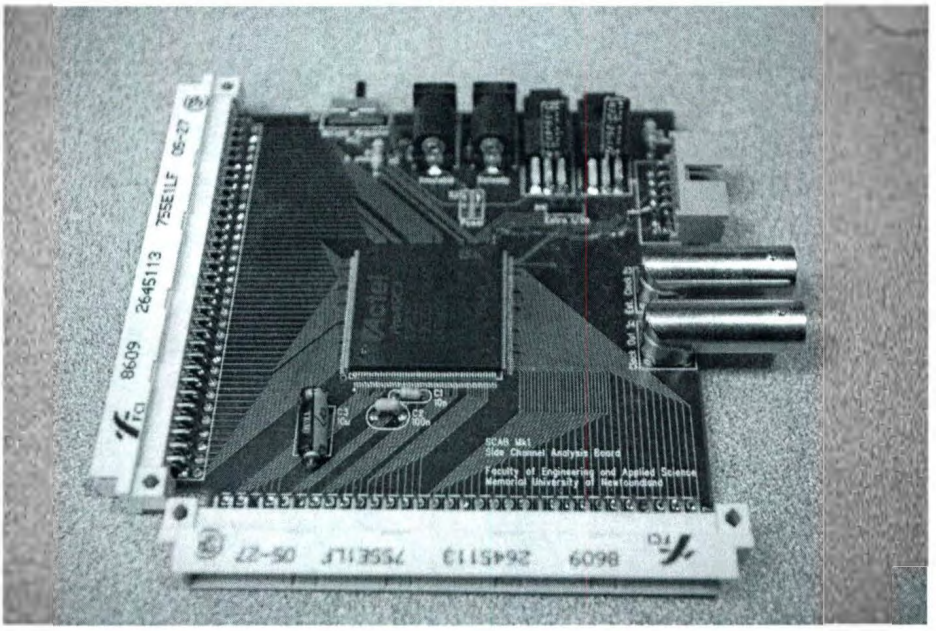

Figure 4.1: SCAB - Side Channel Analysis Board

In order to study arbitrary cryptographic operations, we chose to make use of reconfigurable hardware, namely Field Programmable Gate Arrays (FPGAs). After studying many commercially available FPGA development kits, it was found that none were suitable for our use, for several reasons:

- development boards include hardware extraneous to our purposes (e.g. LEDs, keypads, media and storage I/O) that could obscure the FPGA's power usage
- many FPGA I/O pins are tied to this extraneous hardware, making it difficult to load and unload blocks of data and keys
- measuring power usage, which requires inserting a resistor between V_{CC} and the FPGA, would require physically altering the board - cutting traces and inserting

the resistor

For these reasons, as well as the opportunity for learning, we decided to build our own development board, SCAB.

4.1.1 Design Constraints

Although in this thesis, SCAB was used only for the application of template attacks to the power usage side channel of stream cipher hardware, it was designed to be a facility for subsequent researchers to also use. These researchers could focus on any number of side channels, and any number of cryptographic systems that can be implemented in hardware.

The design of SCAB had to satisfy several constraints, some external and some owing to the intrinsic nature of the research:

- It must be possible to configure SCAB with large, fast implementations of modern ciphers such as AES.
  - To accommodate high-throughput designs like that found in [38], the minimum acceptable gate count of the FPGA is 60k gates.
- It must be possible to transfer blocks of data through parallel I/Os.
  - To accommodate large, modern block ciphers, we wish to be able to transfer 128-bit blocks of data in a single clock period.
- It must be possible to assemble SCAB in Memorial's PCB facilities.
  - Non-local PCB construction was acceptable (and indeed, required), but the assembly process required interaction with technicians, which would not have been possible unless SCAB was assembled locally.

  - High-pin count packages such as Ball Grid Array (BGA) may not be used; only through-hole and surface-mount packages are acceptable.

- The design should be as simple as possible.

  - Increased hardware complexity would increase the time required to design SCAB.

  - FPGA support chips would influence power usage and obscure side channel information.

- Each type of SCA also presents its own requirements and constraints; they are discussed below.

In order to fulfill all of these requirements, we selected an Actel ProASIC3 FPGA, which has 125,000 gates, 131 digital I/Os, independent core and I/O power inputs, surface-mount packaging - Quad Flat Package (QFP) - and on-board flash memory, which eliminates the need for external memory chips on the board.

We would have liked to use an FPGA with at least 256 digital I/O pins, but such chips require packaging technology which cannot be handled in a local assembly of the PCB.

4.1.2 Power Analysis

In order to facilitate power analysis, SCAB has two independent power supply nets: V_{CC}, which powers the FPGA's core logic, and V_{CCI}, which powers FPGA I/O and anything else whose power usage is not relevant to the research.

The former of these nets, V_{CC}, has a resistor - R1 in Figure 4.2 - inserted in series with the power supply so that the FPGA's current draw can be measured.

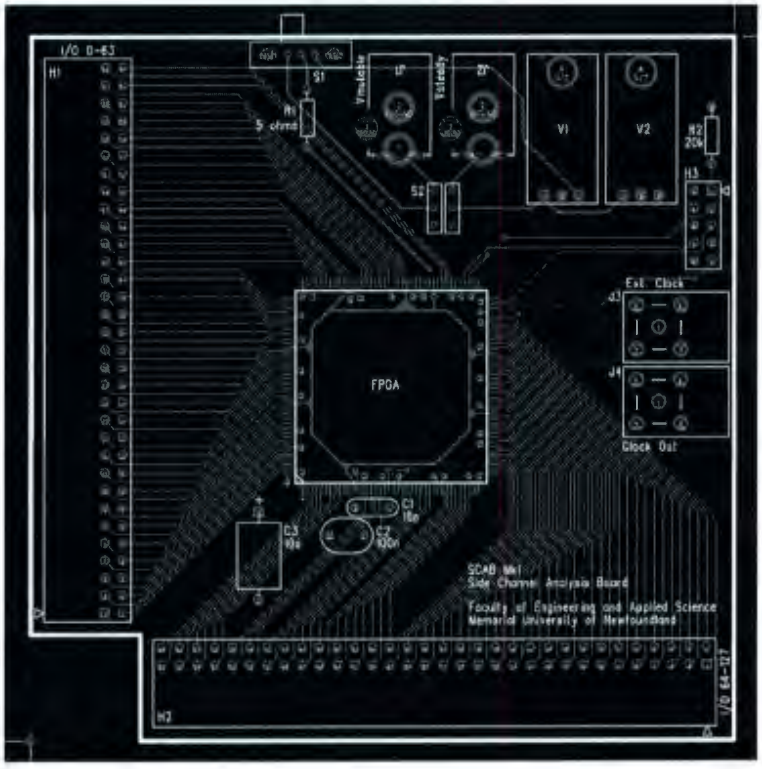

Figure 4.2: PCB Layout for SCAB

This resistor must have a very small value (we have selected 5Ω) so as to keep V_{CC} from falling outside the FPGA's operating envelope. V_{CC} may be powered by one of two sources; when performing power analysis, a researcher will typically choose to power V_{CC} by the voltage regulator V1. The other option - best for fault analysis - is discussed in Section 4.1.3.

The latter net, V_{CCI}, is independent of V_{CC}, so that power used for FPGA I/O does not affect the measurement of core power usage. V_{CCI} is powered by the voltage regulator V2, which ensures that I/O voltage is always held steady, even during fault attacks (Section 4.1.3). This regulator is, in turn, connected to the DC power jack J2 whenever the main power switch (S2) is closed.

4.1.3 Fault Analysis

SCAB also facilitates fault analysis, in which the researcher attempts to induce an incorrect computation through externally-induced faults. The source of these faults may include glitches in the clock signal or unusual power supply characteristics (e.g. too high, too low, spikes). SCAB provides the access needed to study the effect of such faults through its power supply design, external clock and large number of I/O pins.

Power Supply As mentioned in Section 4.1.2, SCAB's V_{CC} supply net may be driven by a voltage regulator or, more interestingly for the purpose of fault analysis, an external power source. This direct connection to the V_{CC} net allows a researcher to set up abnormal power supply conditions, including undervoltage or overvoltage conditions as well as voltage spikes.

External Clock SCAB also supports timing fault analysis. Since SCAB's clock is driven externally (connected via BNC), a researcher can modify clock signals, inducing glitches and changing duty cycles and periods, in an attempt to induce erroneous computation.

I/O Pins Finally, SCAB's large I/O bank allows a researcher to export up to 128 internal signals from a hardware design, which permits the direct observation of how internal values change while the system is under external stress (power, clock or temperature glitches, ionizing radiation, etc.). This level of access permits the study of fault propagation, and it also allows researchers to verify existing fault models.

4.1.4 Timing Analysis

SCAB's external clock and large I/O bank also supports timing analysis: as a researcher can manipulate clock signals at will and gain insight into the internals of a hardware implementation, looking to see not just when an algorithm is complete, but where sub-sections of it are complete.

4.2 Other Hardware

The complete experimental setup is shown in Figure 4.3.

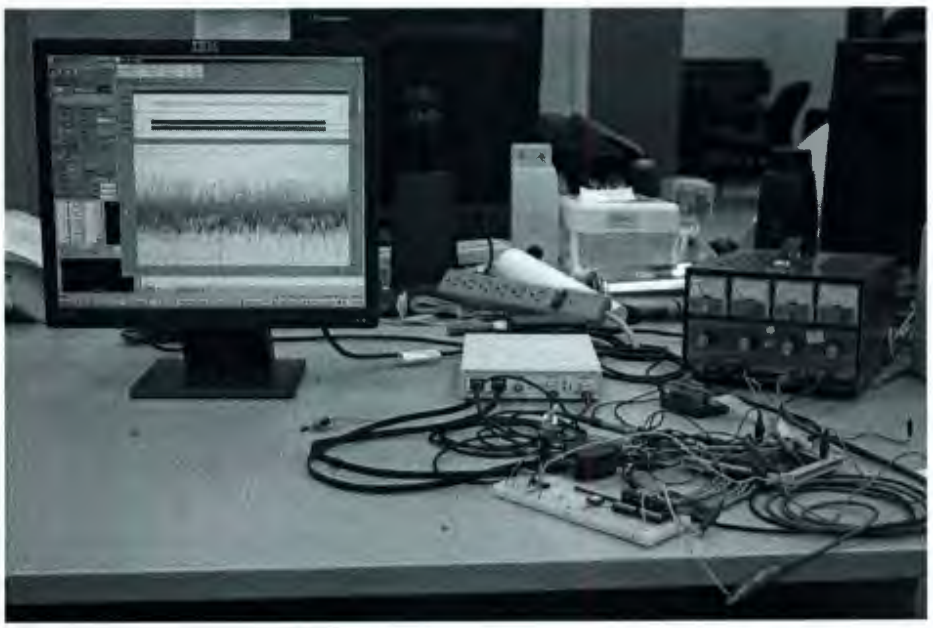

Figure 4.3: Experimental setup

Besides SCAB, other hardware that can be seen in Figure 4.3 includes:

- oscilloscope used for measurement
- DC power supply
- DIP switches (used for parallel key and/or IV bit inputs)

- momentary reset switch
- "go" switch

The latter of these switches provided the hardware with the signal to start (and continue) cryptographic operation. Interfacing this switch directly with the hardware required debouncing to prevent momentary glitches in the "go" signal caused by the mechanical bouncing of switch elements - from reaching the cryptographic hardware. The debouncing circuit is shown in Figure 4.4.

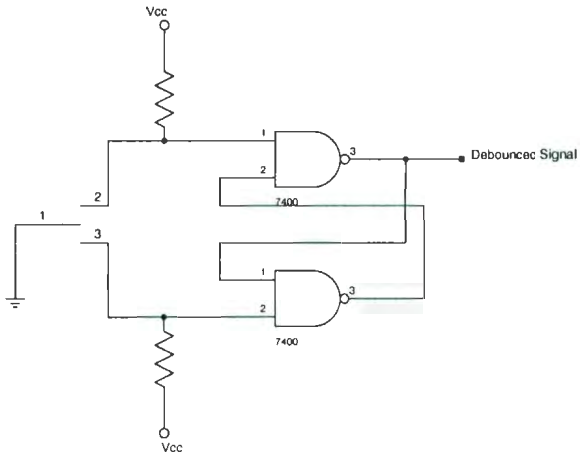

Figure 4.4: Switch debouncing circuit

4.3 Measurement Equipment

The focus of our research is the application of template attacks to the power usage side channel of stream ciphers using SCAB. Our power measurements were all made with the Cleverscope CS328A [39], a PC-based mixed-mode oscilloscope with high time resolution (10 ns minimum period) and deep memory (up to one million data points per channel). This scope allowed us to:

1. Generate clock signals

2. Monitor eight digital channels

3. Measure two analog voltage channels, using analog, digital and/or external triggering

The Cleverscope interface is shown in Figure 4.5.

Figure 4.5: Cleverscope PC interface

The eight digital channels were used to monitor the hardware being tested, including clock input and keystream output. The analog channels were used to measure the voltage before and after the resistor R1 in Figure 4.2. Calculating instantaneous power usage from these voltages is very simple:

$$p(t) = v_{cc}(t) \cdot \frac{v_{supply}(t) - v_{cc}(t)}{R_1}. \tag{4.1}$$

With one million data points per channel at our disposal, we were able to capture thousands of points per key/IV pair, or approximately 50 data points per clock cycle. This data allowed us to construct accurate templates (see Chapter 5).

The data captured by the Cleverscope was saved to text files and interpreted by our software, described in the following section.

4.4 Software

Turning physical simulations or measurements into classification statistics requires software. The software workflow is shown in Figure 4.6. We wrote approximately 10,000 lines of C++ to accomplish these tasks; the programs which accomplish them are described here.

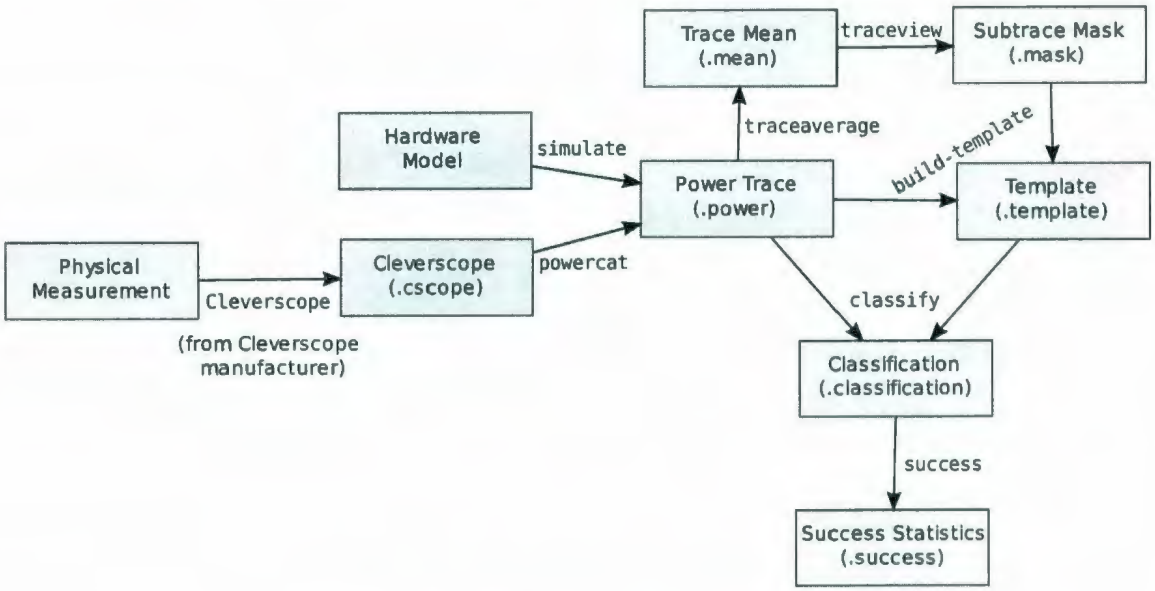

Figure 4.6: Workflow – data files

4.4.1 Power Trace Formatting

After taking physical measurements of side channel data, the output of the Cleverscope program is a file that, for every time increment, specifies the voltage for each analog trace and logic value for each digital trace. We must take this information and turn it into a format more amenable to interpretation[1].

The `powercat` program reads voltage traces from input files (Cleverscope, Tektronix GRAB2212 or our own analog trace format) and outputs them to binary trace files. These files may be a concatenation of several trace files — hence the name of the program — and contain just a power trace and a digital "partitioning" trace (which is described in Section 4.4.2, below).

A text file containing a full capture of Cleverscope memory occupies 50 MB of disk space. If many such captures are required (e.g. when capturing output from multiple keys), storage requirements quickly become enormous. Converting this data to a binary format saves both storage space and computational complexity, as text parsing is not required every time we load a power trace.

4.4.2 Calculating Trace Mean Vectors

The `traceaverage` program takes a power trace file, *partitions* it and averages all of the *subtraces*.

Aside: Partitions and Subtraces A single power trace file may contain traces for many samples. Each of these *subtraces* is denoted by a single digital trace, called the *partitioning* trace. This partitioning trace is shown in Figure 4.7, and it is used by the `traceaverage` program (and others) to partition a long trace file into multiple subtraces. The partitioning trace is equal to 1 during encryption operations and 0

[1] Details concerning file formats are given in Appendix B on page 115

between them. Thus, whenever the partitioning trace switches from 0 to 1, a new subtrace (sample trace) has begun.

4.4.3 Simulating Power Usage

The `simulate` program simulates the power usage of a hardware implementation of a cryptographic cipher. The user can specify a number of parameters:

- the cipher to simulate
  - currently LFSR-16 or Trivium
- the secret key to use
  - specified as 0xXXXX or 0bXXXX, where X can be:
    - a value (0-1 for binary, 0-f for hex)
    - the literal X, meaning "assign randomly for each sample"
- how much noise to add
- sampling period
- the number of samples to simulate
- the number of clock cycles to simulate per sample
- the number of samples to simulate per clock cycle

The power model used can be customized by writing a C++ class that implements the PowerUsageModel interface (see Section B.5.1). This model tells the simulator how much power is consumed by a flip-flop that either:

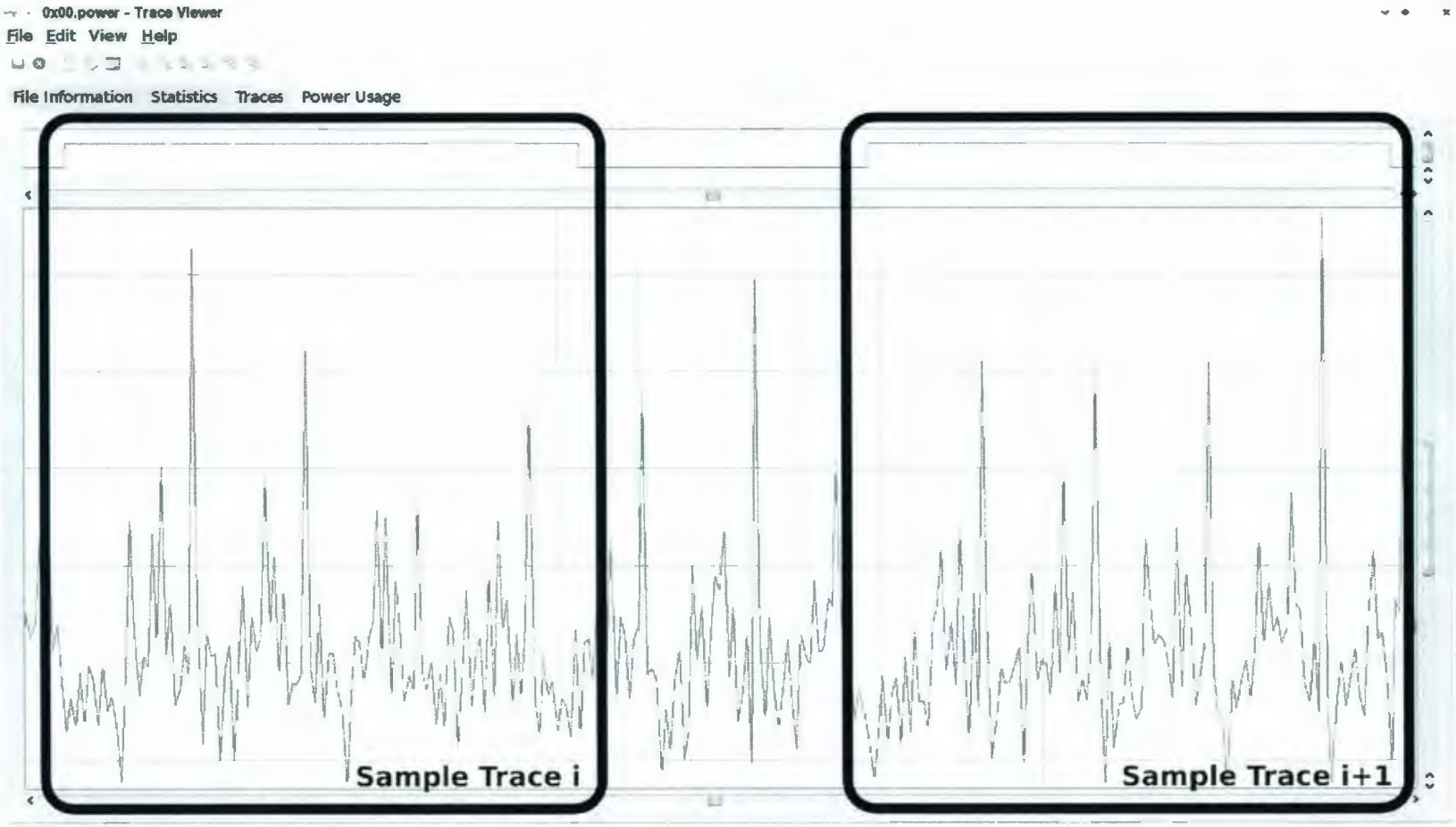

Figure 4.7: Partitioning trace

- changes from high to low,
- changes from low to high,
- remains steady at low or
- remains steady at high.

Determining the number of flip-flops that maintain or change state is the job of another C++ class, one which inherits from the abstract class Cipher (see Section B.5.2). This class tells the simulator, on each clock cycle, how its internal state has changed since the last cycle.

Using the power usage model and the cipher model, the simulator generates power traces for particular ciphers running on particular (simulated) hardware.

The output of this simulation is a power trace file (`.power` extension) and a subtrace mean file (`.mean` extension).

4.4.4 Viewing Power Traces

The `traceview` program is used for two purposes:

1. To view voltage and power trace files, as well as simple statistics about them
2. To view inter-operation statistics and choose subtrace masks (see Section 3.2.4 on page 32).

In the first mode, the program simply displays the contents of a trace file, as in Figure 4.8.

In the second mode, several subtrace mean files are loaded (one per operation), and simple inter-operation statistics can be viewed. From the inter-operation standard

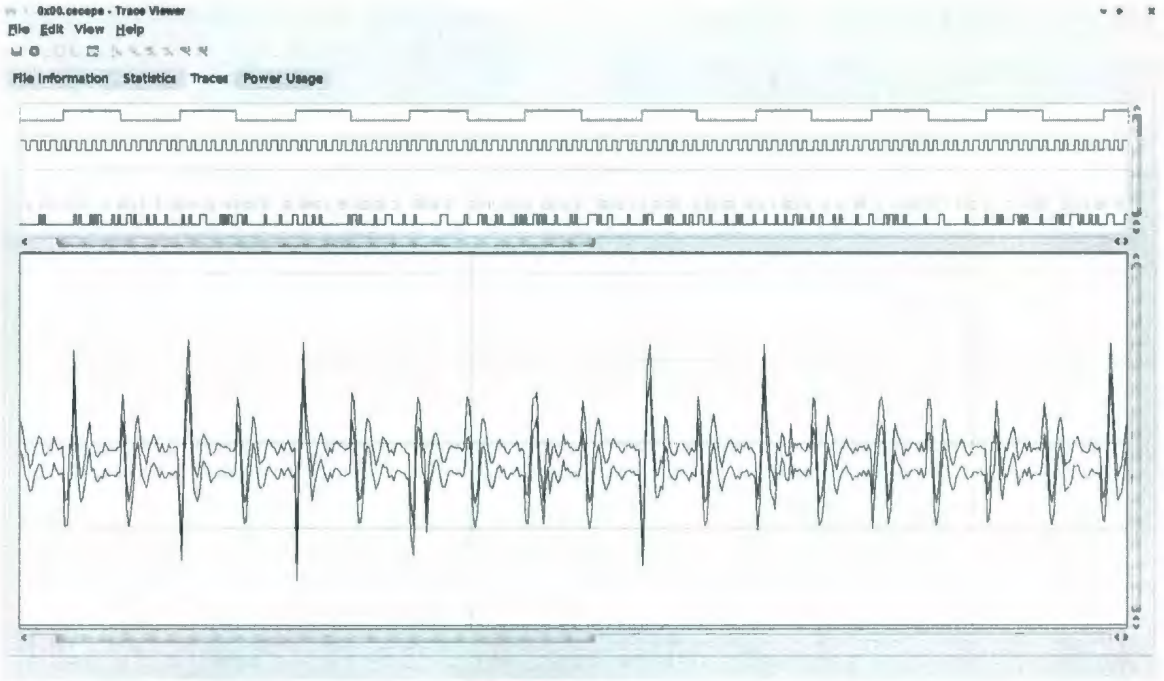

Figure 4.8: `traceview` showing the contents of a Cleverscope file

deviation, we select a subtrace mask to apply when building templates. In Figure 4.9, we can see a line drawn across the inter-operation standard deviation. This line is the cutoff above which the subtrace mask will accept points and below which it will reject them.

In this case, 32 points in each sample trace will become part of the template; the template's size will be $N = 32$. This number N can be varied until the desired value is reached, whether by inspection – placing the cutoff line above the level of background noise, or to achieve a particular probability of classification success. See Section 5 on page 64 for graphs of classification success rate versus template size.

4.4.5 Building Templates

The `build-template` program takes as input a power trace (containing a number of subtraces) and an optional subtrace mask. Its output is a template file containing

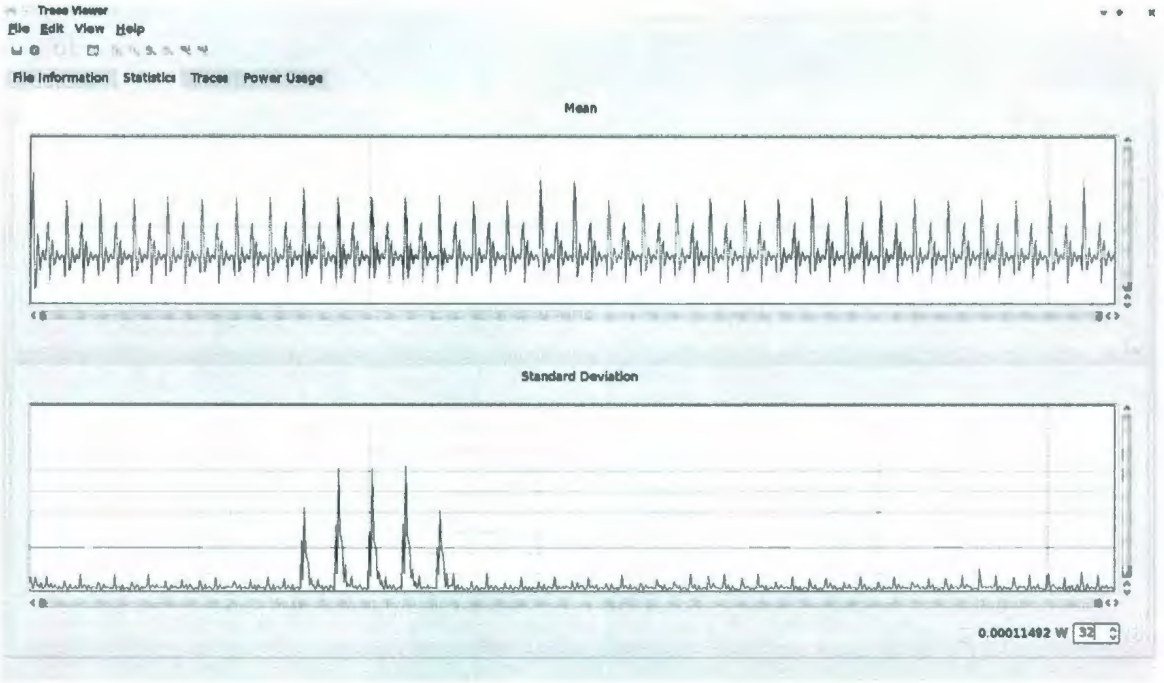

Figure 4.9: `traceview` used to select subtrace mask

the pair $\left(\hat{\mu}^{(k)}, \hat{\Sigma}^{(k)}\right)$ for a particular operation $O^{(k)}$ as explained in Section 3.8 on page 30. This file has a `.template` extension (see Figure 4.6 on page 53) and can be read by the next program in the software workflow, `classify`.

4.4.6 Classifying Power Traces

The `classify` program takes as input a power trace file (`.power`) – whose subtraces are known to have been generated by a particular operation $O^{(k)}$ – a number of template files (`.template`) and an optional subtrace mask file (`.mask`). It partitions the trace file into subtraces, and classifies each as being likeliest to correspond to one of the given templates. The output is a classification file (`.classification`), whose format is shown in Figure 4.10.

The probabilities are derived using the procedure given in Section 3.2.3 on page 30, with one modification: since the attacker knows that *one* of the generated templates

```
classify

Loading mask file: ../mask32.mask
Opening templates: 0x00.template 0x01.template 0x02.
   template 0x03.template 0x04.template 0x05.template 0x06
   .template 0x07.template 0x08.template 0x09.template 0
   x0a.template 0x0b.template 0x0c.template 0x0d.template
   0x0e.template 0x0f.template
Done reading templates.
Reading data to classify...
Opening '../dut/0x00.power'...
[==================================================] 100%
256 traces, sized [32-32] samples/trace
Trace 0:
Probability of template 0x00.template: 0.999997
Probability of template 0x01.template: 1.20044e-07
Probability of template 0x02.template: 1.66347e-14
Probability of template 0x03.template: 5.76915e-08
Probability of template 0x04.template: 9.61753e-07
...
```

Figure 4.10: `classify` output

is for the operation that generated the trace, the probability density function values are normalized such that they add to 1 and represent the probability that a particular operation produced the trace, *given that* one of the templates is correct.

4.4.7 Evaluating Classification Success Rate

The `success` program reads `.classify` files (one per operation) and produces summary statistics, both on per-key and overall bases. The output of this program is shown in Figure 4.11.

```
success
Opening output files...
0x00.classification... key: 0x00
255 correct guesses (99.6094%, 98.3821% certainty)
1 incorrect guesses (0.390625%, 85.6105% certainty)
0x01.classification... key: 0x01
...
Lowest success rate:     97.6562%
Highest success rate:    100%
Average success rate:    99.3896%
```

Figure 4.11: success output

4.5 Summary

We have described the experimental setup used to simulate, realize and measure the characteristics of cryptographic hardware for this thesis.

The Side Channel Analysis Board (SCAB) was designed to be a platform for security researchers to investigate many kinds of side channel analysis. It was designed to meet the following constraints:

- It must be possible to reconfigure SCAB with large, fast implementations of modern ciphers such as AES.
- It must be possible to transfer blocks of data through parallel I/Os.
- It must be possible to assemble SCAB in Memorial's PCB facilities.
- The design should be as simple as possible.
- It should meet side channel-specific constraints:
  - Power Analysis
    * SCAB should incorporate two independent supply nets, one for core logic and one for I/O.

    * The core logic supply should have a small-valued resistor inserted in series with the power supply for measurement purposes.

  - Fault Analysis

    * SCAB should incorporate both on-board – i.e. regulated and external power supply options.
    * SCAB should be driven by an external clock.
    * SCAB should have a large number of I/O pins to expose internal state and allow verification of fault models.

  - Timing Analysis

    * SCAB should be driven by an external clock.
    * SCAB should have a large number of I/O pins to expose internal state.

Other hardware in the setup included power supplies, switches and measurement equipment.

This measurement equipment consisted of the Cleverscope CS328A, a PC-based oscilloscope. It was purchased for this research, and performed its tasks well.

We also wrote 10,000 lines of C++ software to do many things:

- reformat power data
- calculate average power usage
- simulate power usage
- view power traces and select template masks
- build templates
- classify power traces

- calculate classification success rates

This experimental setup was used to apply template attacks to stream cipher hardware. The results of this application are given in the next two chapters.

Chapter 5

Experimental Results and Analysis

In this chapter, we present the initial results of our experimentation. These results consist of basic measurements of hardware characteristics and the application of the template attack technique to both simulated and measured power usage characteristics of a stream cipher building block.

5.1 Initial Experiments

One of the first uses of the experimental setup was to evaluate the difference between the power consumed during the flip of a flip-flop and the power consumed at other times. In order to test this, we built a very simple circuit called "Flip-Flopper," shown in Figure 5.1. This circuit used a large number of identical elements, acting in parallel, to increase the ratio of data-dependent power usage to background noise.

This circuit consists of 512 D flip-flops switching in concert between the values 0 and 1. There is a counter and two comparators, one to check for a counter value of 5 and the other to check for a counter value of 9. These comparators control the changes of the flip-flops: after initializing all values to 0, the circuit counts five clock

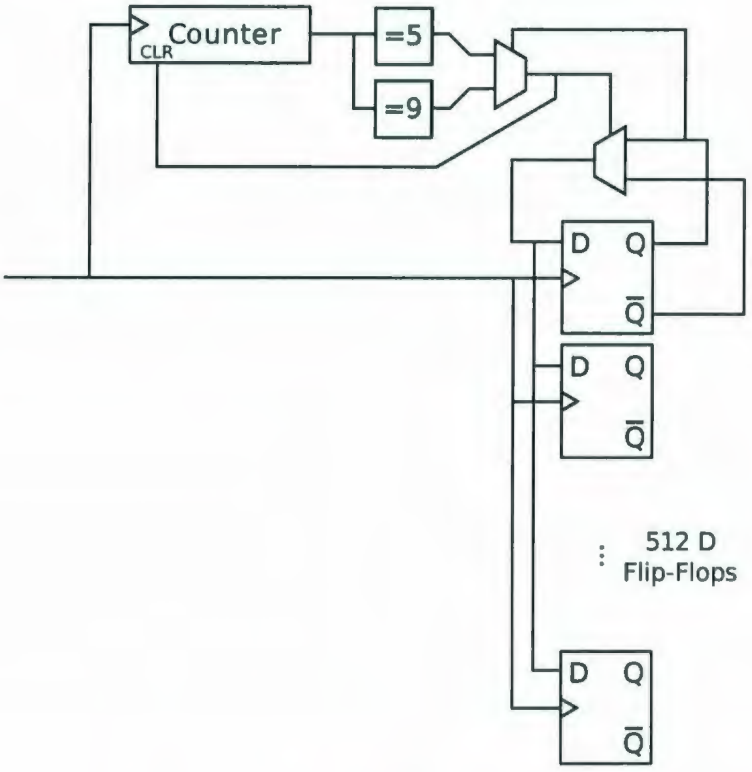

Figure 5.1: The "FlipFlopper" Circuit

cycles, then all 512 flip-flops change their values from 0 to 1. After another nine clock cycles, all flip-flops change from 1 back to 0. The output of this simple circuit, as well as its instantaneous power usage, is shown in Figure 5.2.

This power trace was determined according to Equation 4.1, and it shows that not only is the power consumed by a bit flip greater than the static power, but the power consumed when the bits flip from 0 to 1 (approximately 11 mW in total, for all 512 flip-flops) is greater than the power consumed when the bits flip from 1 to 0 (approximately 6 mW in total). This difference between types of bit flips provides us with more information than we expected.

These initial results provided us with the basic power characteristics of D flip-flops

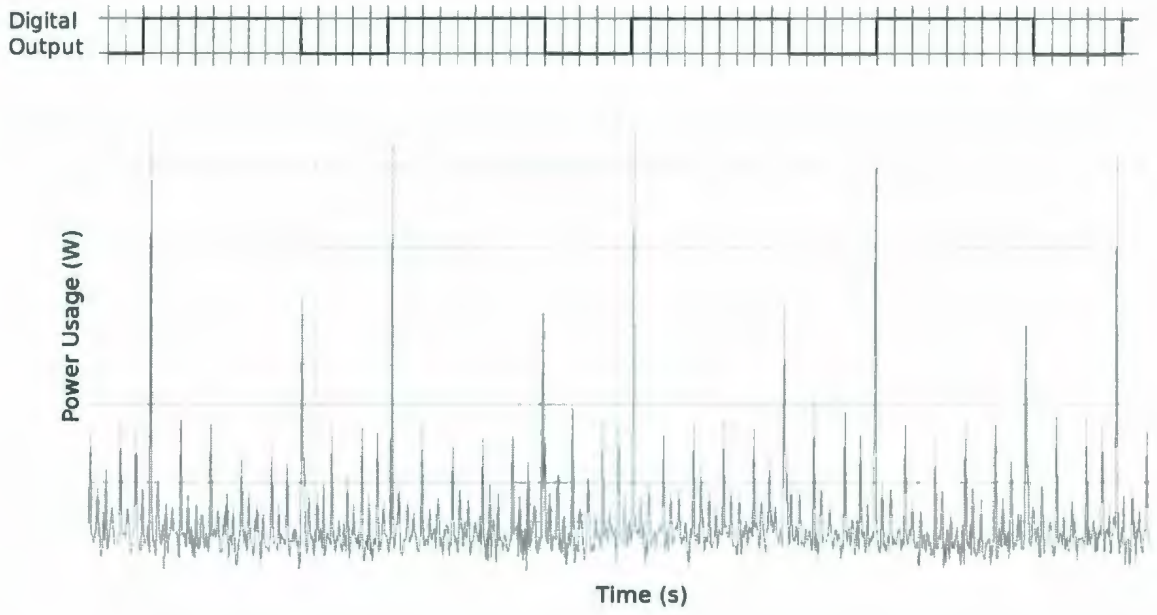

Figure 5.2: FlipFlopper output and instantaneous power usage

in our FPGA hardware, as given in Table 5.1. The values in this table were derived by measuring the total power usage of the system and dividing by 512, the number of flip-flops in the system.

Event	Minimum Power	Maximum Power	Mean Power	Power Range
Static	10.0 μW	11.9 μW	11.9 μW	$\pm$.97 μW
Bit flip (1 to 0)	17.2 μW	18.0 μW	17.6 μW	$\pm$.39 μW
Bit flip (0 to 1)	21.3 μW	22.5 μW	21.9 μW	$\pm$.59 μW

Table 5.1: Power usage characteristics

From the mean values in this table, we created a very simple simulation model: for every bit in a cryptographic system that remains the same, power usage will be 11.9 μW. For every bit that changes from 0 to 1, the power usage will be 21.9 μW, and for every bit that changes from 1 to 0, 17.6 μW. To this ideal value, we add additive white Gaussian noise (AWGN); how much noise we add is a parameter that we vary while studying the attack's effectiveness.

5.2 LFSR-16

Before attacking a full-fledged stream cipher, we started by attacking a basic building block of many stream ciphers, the linear feedback shift register (LFSR).

LFSRs are shift registers that feed back on themselves, inserting a new bit at their tail every clock cycle which is a linear combination of other bits in the register. On its own, an LFSR is not a stream cipher: it can be cryptanalysed trivially because of its linear nature. It is, however, a useful building block in the construction of real stream ciphers.

We chose a simple 16-bit LFSR with the characteristic polynomial given in Equation 5.1.

$$x^{16} + x^{14} + x^{13} + x^{11} + 1 \tag{5.1}$$

This LFSR has a *maximal period*: assuming it is not loaded with $\vec{0}$, it will shift $2^{16} - 1$ times before it repeats a previous state. A simple diagram of LFSR-16 is shown in Figure 5.3.

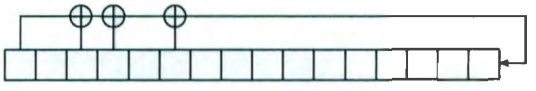

Figure 5.3: Design of LFSR-16

5.2.1 Simulation Results

Using the power usage characteristics in Table 5.1, we simulated LFSR-16 running on an Actel ProASIC3 FPGA. The power usage of this cipher was simulated using a 16-

bit initial state (key) that was determined randomly, except for the most significant four bits. These bits were fixed for any particular operation $O^{(k)}$, so we were able to generate 16 templates, one for each of the keys { 0x0XXX, 0x1XXX ... 0xfXXX }, where the most significant bit of the key is loaded into the left-most bit of the shift register in Figure 5.3.

For each simulation of LFSR-16, we simulated a different number of sample traces (16, 32, 64, 128 or 256). We also varied the amount of noise added to the power trace, as well as the number of data points included in the template mask (see Section 3.2.4 on page 32). The detailed results of this analysis can be found in Appendix A, but we present an overview here.

Figure 5.4 shows the basic inter-operation statistics for the simulated LFSR-16 (64 samples, peak noise 10^{-6}). The top graph is the inter-operation mean, and the bottom graph is the inter-operation standard deviation. As expected, the greatest deviation is early in the sample traces, before the key bits "mix in" to the cipher's state.

The line across the standard deviation graph shows the cutoff for trace masking with $N = 8$. Even with such a small number of data points, we are able to obtain useful information from the traces so as to have very good classification success.

Figure 5.5 shows the classification success rate for simulated LFSR-16 when we use $L = 64$ training samples per operation and the noise present has peak values of 10^{-5}. This noise value was chosen because it fits with the characteristics in Table 5.1. It shows the success rate increasing with template size, and even with template size $N < 5$, the average classification success rate is greater than 6.25%, which is the success rate we would expect if we were guessing randomly.

The four lines on this graph are:

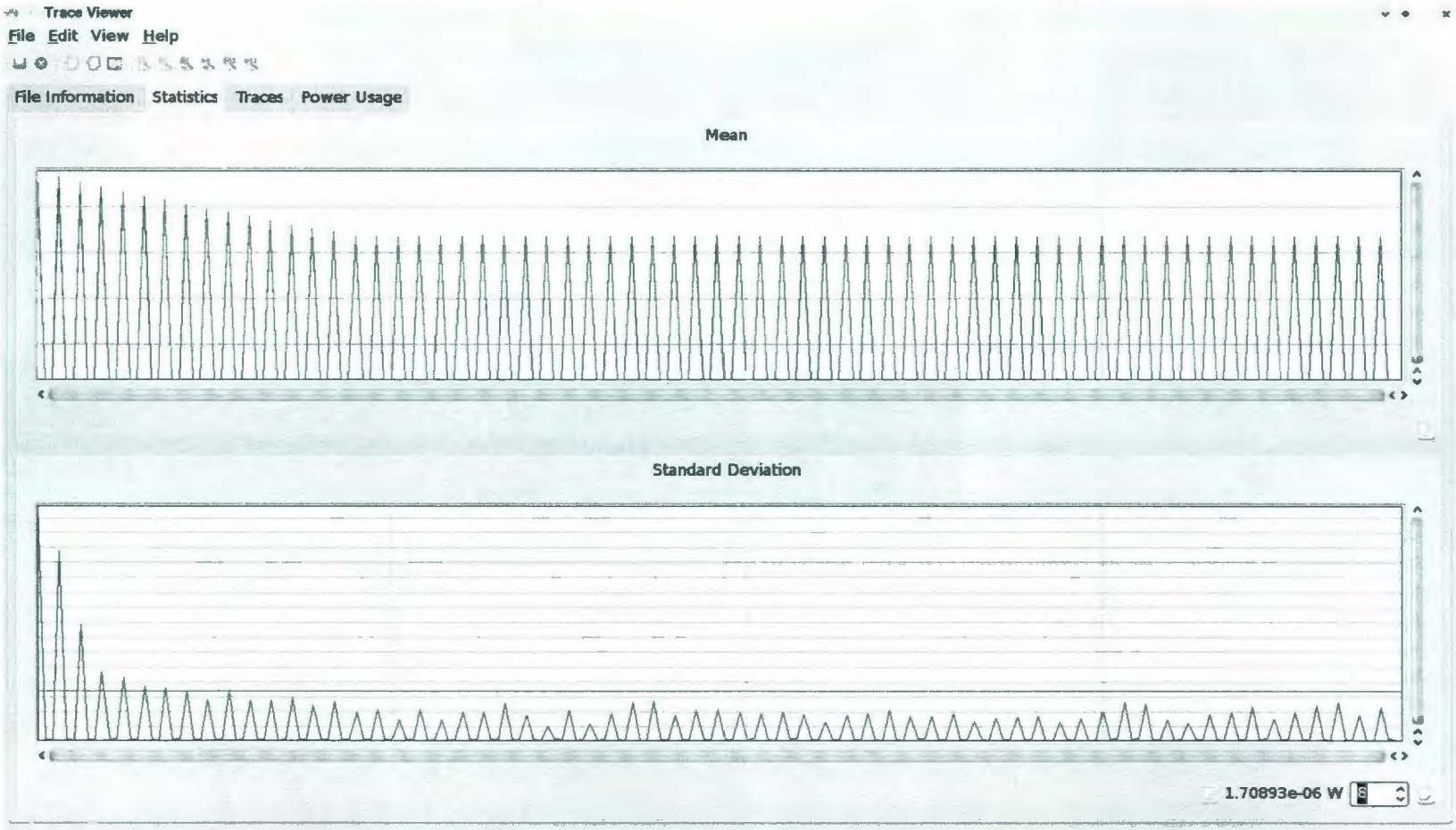

Figure 5.4: Basic statistics of simulated LFSR-16

1. Maximum success rate: the highest rate of correct classification for any operation

2. Average success rate: the average rate of correct classification over all operations

3. Minimum success rate: the lowest rate of correct classification for any operation

4. Guess rate: how successful we would expect to be if we guessed randomly

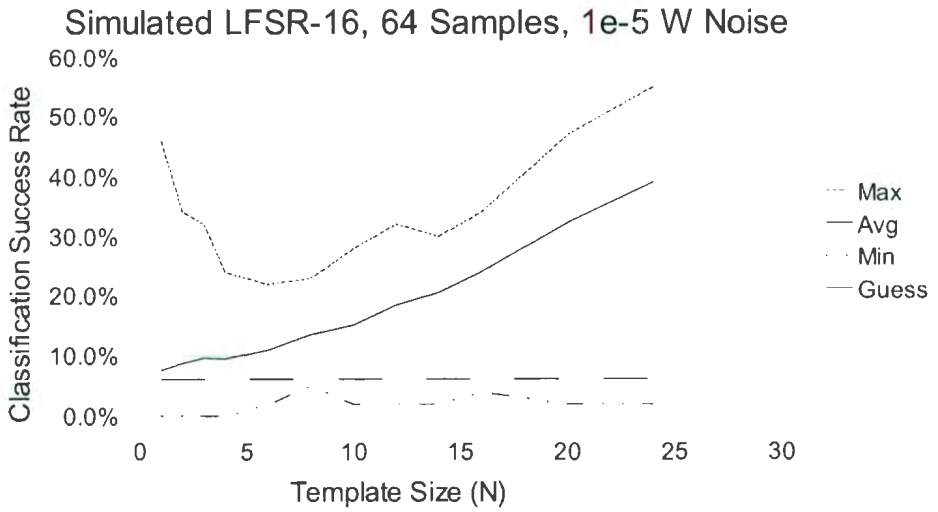

Figure 5.5: Classification success vs. template size

Figure 5.6 shows the classification success rate versus template size when the noise is much lower, with a power peak of 10^{-7} W. This noise level is lower than observed, but as we will see in Section 5.2.2, the results are a closer approximation to those obtained through physical experiment than those obtained using the noise level of 10^{-5} W (peak).

With this noise level, we were able to achieve ~90% average classification success using as few as four data points and approximately 80% minimum success with as few as 10 points, making this an attack of remarkably low computational complexity.

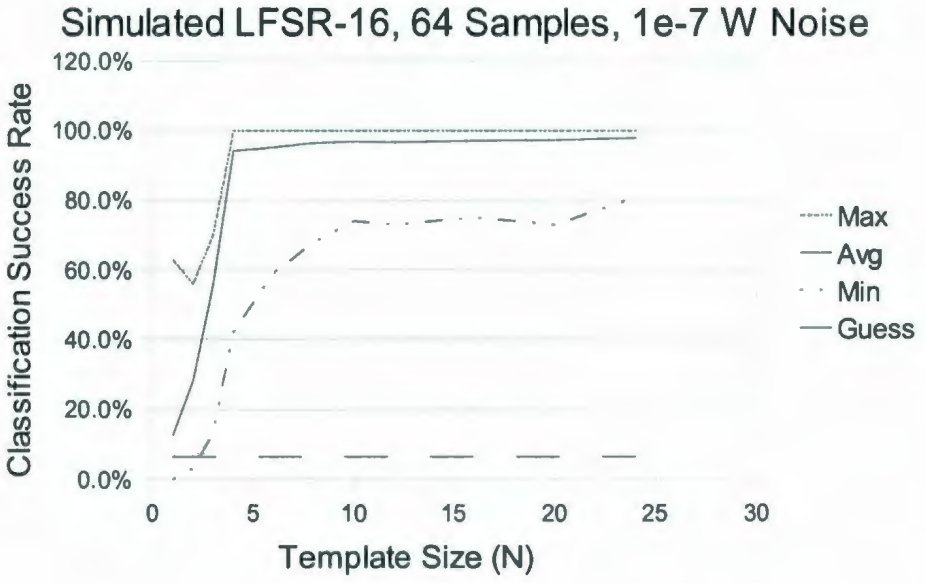

Figure 5.6: Classification success vs. template size

Effect of Noise Increasing the amount of noise in the power traces has a negative effect on classification success, as shown in Figure 5.7. This graph shows a general downwards trend in classification success as the peak noise increases from 10^{-7} W.

Effect of Varying Bits Under Attack Varying which key bits templates were constructed from also affected the success rates of the template attack. Inter-operation statistics are shown in Figures 5.8, 5.9, 5.10 and 5.11.

These figures show that attacking less significant bits leads to more similar operation means, as shown by fewer peaks in inter-operation standard deviation.

Correspondingly, we see in Table 5.2 that classification success rates diminish as we attack progressively less significant bits in the LFSR-16 key.

This behaviour can be explained by observing the feedback "taps" in LFSR-16. The less significant the bits which vary according to operation, the more clock cycles it will take them to reach the feedback taps and affect other bits. Once the least

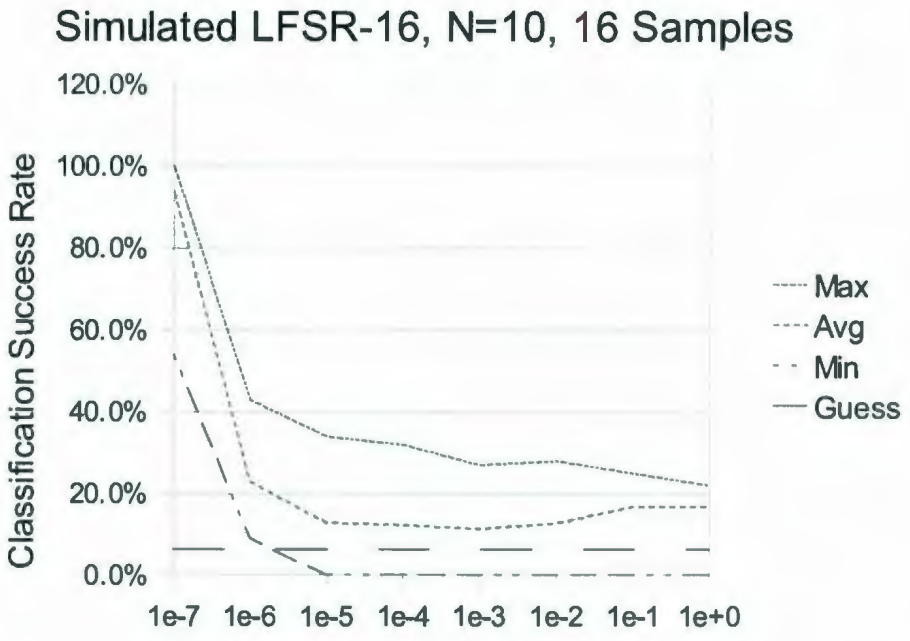

Figure 5.7: Classification success vs. peak noise

Bits Under Attack	Minimum	Average	Maximum
0 – 3	11.4%	19.3%	38.3%
4 – 7	9.7%	14.9%	26.9%
8 – 11	12.2%	17.2%	29.1%
12 – 15	10.3%	16.2%	25.1%

Table 5.2: Classification success rate vs. bits under attack

significant bits have reached the feedback taps, however, the operation of the LFSR will have caused internal states to vary just as greatly within operations as between operations. Thus, inter-operation differences are reduced, as are classification success rates.

Effect of Number of Training Samples For a fixed number of sample points in the template mask, a higher number of training samples was more likely to yield a correct result, as shown in Figure 5.12. With such low noise, classification is very

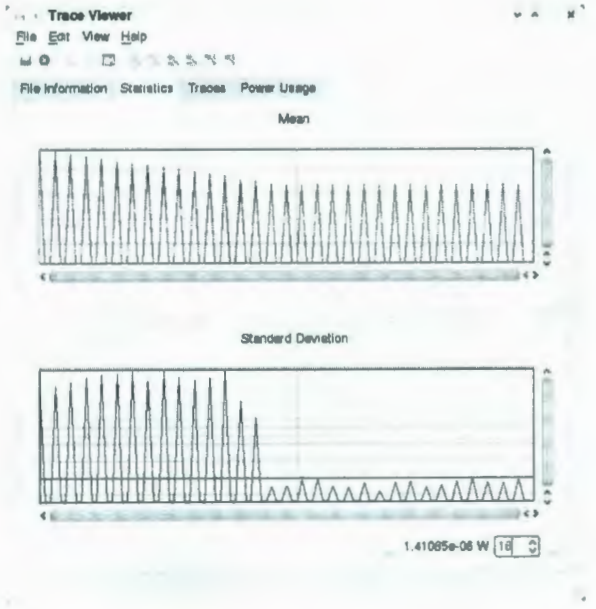

Figure 5.8: Inter-operation statistics: varying bits 0–3

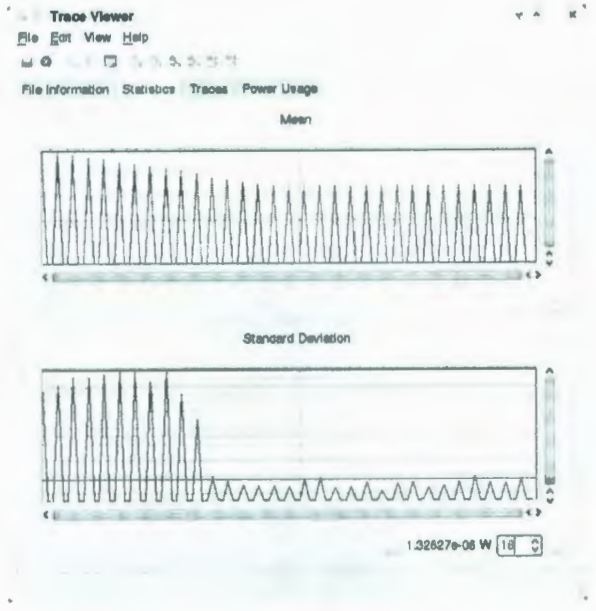

Figure 5.9: Inter-operation statistics: varying bits 4–7

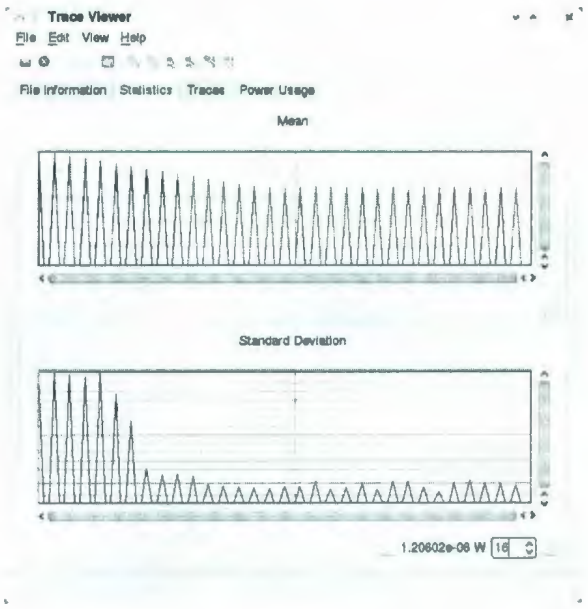

Figure 5.10: Inter-operation statistics: varying bits 8–11

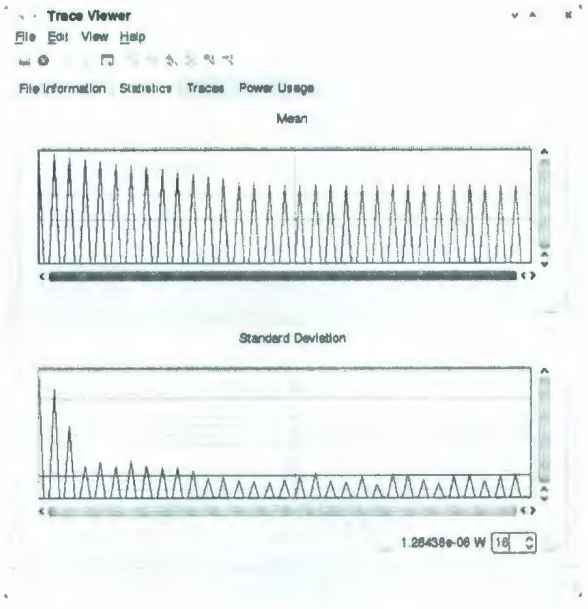

Figure 5.11: Inter-operation statistics: varying bits 12–15

successful even with a low number of training samples, but as we will see in Section 5.2.2, this level of classification success is realistic.

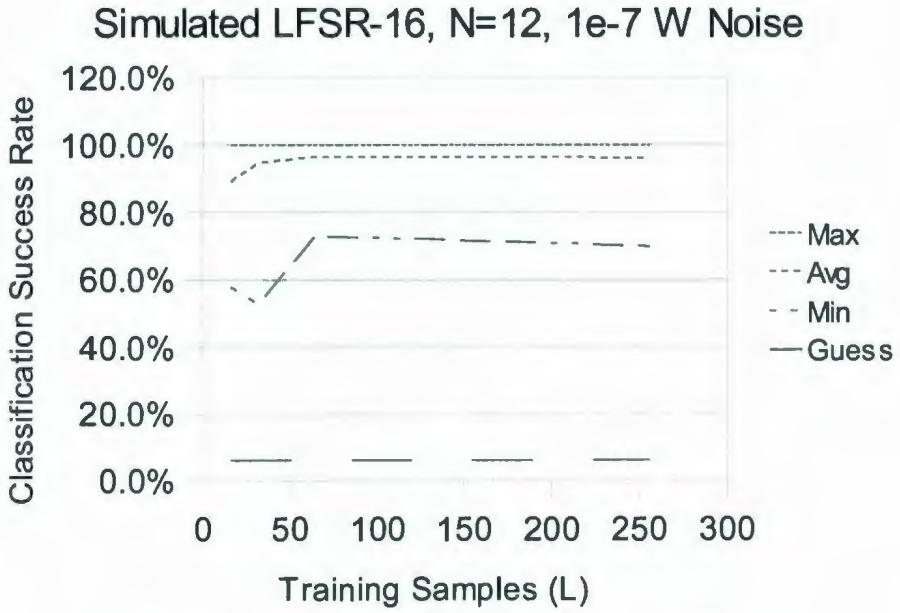

Figure 5.12: Classification success vs. training samples

With this success in hand, we proceed to apply template attacks to the LFSR-16 implemented in hardware.

5.2.2 Experimental Results

Having successfully attacked a simulated LFSR-16, we proceeded to a practical application of the template attack technique in real hardware. Using the SCAB and Cleverscope described in Chapter 4, we carefully measured the power used by LFSR-16 during its initialization. The secret key was set to { 0xXX00, ... 0xXX0f }, and 256 samples were taken, allowing the hardware to initialize once with each possible combination of unspecified key bits.

Figure 5.13 shows the classification success rate versus template size. As in the simulated results, approximately 90% average success was achieved with low template sizes ($N = 11$), and with $N \geq 12$, the minimum classification success was approximately 80% or higher.

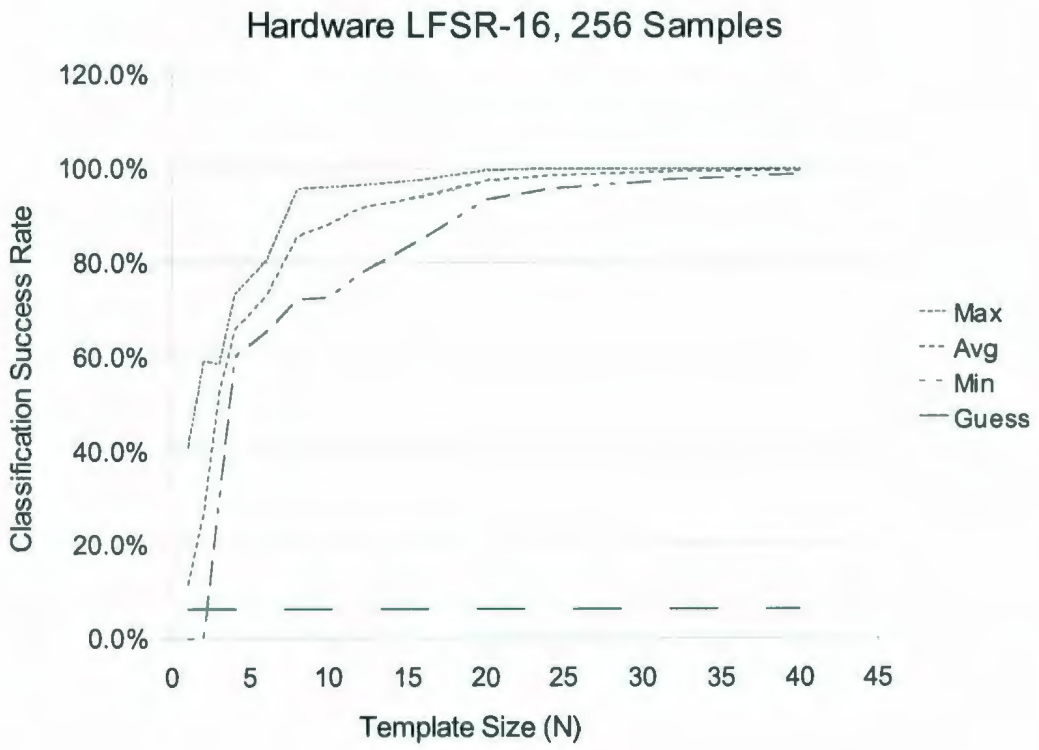

Figure 5.13: Classification success vs. template size

The fact that the template attack performed better against real hardware than against simulated hardware has to do with information content. The simulated LFSR-16's power usage carries information at the edge of a clock pulse, but the physical LFSR-16's trace carries some information during the rest of the pulse, too – though less than at the edge. The inter-operation standard deviation for the physical LFSR-16 is shown in Figure 5.14; compared to Figure 5.4, we can see that there are many

data points per clock cycle whose standard deviation rises well above the background. The line in Figure 5.14's standard deviation graph (the bottom graph) shows that 48 points — all centred around five clock transitions — can be selected from the trace whose values are clearly more significant than the others.

Information content also affects classification success in that, for implementation reasons, the keys used for the hardware LFSR-16 had four fixed bits. As mentioned above, the secret keys were in the set { 0xXX00, ... 0xXX0F }, not { 0xXXX0, ... 0xXXXF }. This is because keys were fed to the LFSR-16 via manual interaction, in the form of DIP switches. To attack a full LFSR-16, we would have to build more sophisticated off-board hardware to load randomly-generated keys and initialization vectors. This, combined with the PC software to drive it, is beyond the scope of this research. To attack LFSR-16, we simply fixed four key bits to 0, set four more in an operation-dependent manner and iterated through all 256 possibilities for the eight unfixed bits.

Inter-operation standard deviation peaks are observed later in Figure 5.14 than in Figure 5.4; this is due to the loading of secret keys { 0xXX00, ... 0xXX0F } and not { 0xF0XX, ... 0xFFXX }. The peaks start occurring at clock edge 8 instead of clock edge 0; this is precisely what we would expect if the differing key bits were loaded into the four right-most flip-flops in Figure 5.3.

5.3 Summary

In this chapter, we revealed the results of our initial experiments using the FlipFlopper and LFSR-16 circuits.

Using the FlipFlopper circuit, we were able to measure the power usage characteristics of the FPGA on SCAB. These characteristics led us to a power model to use

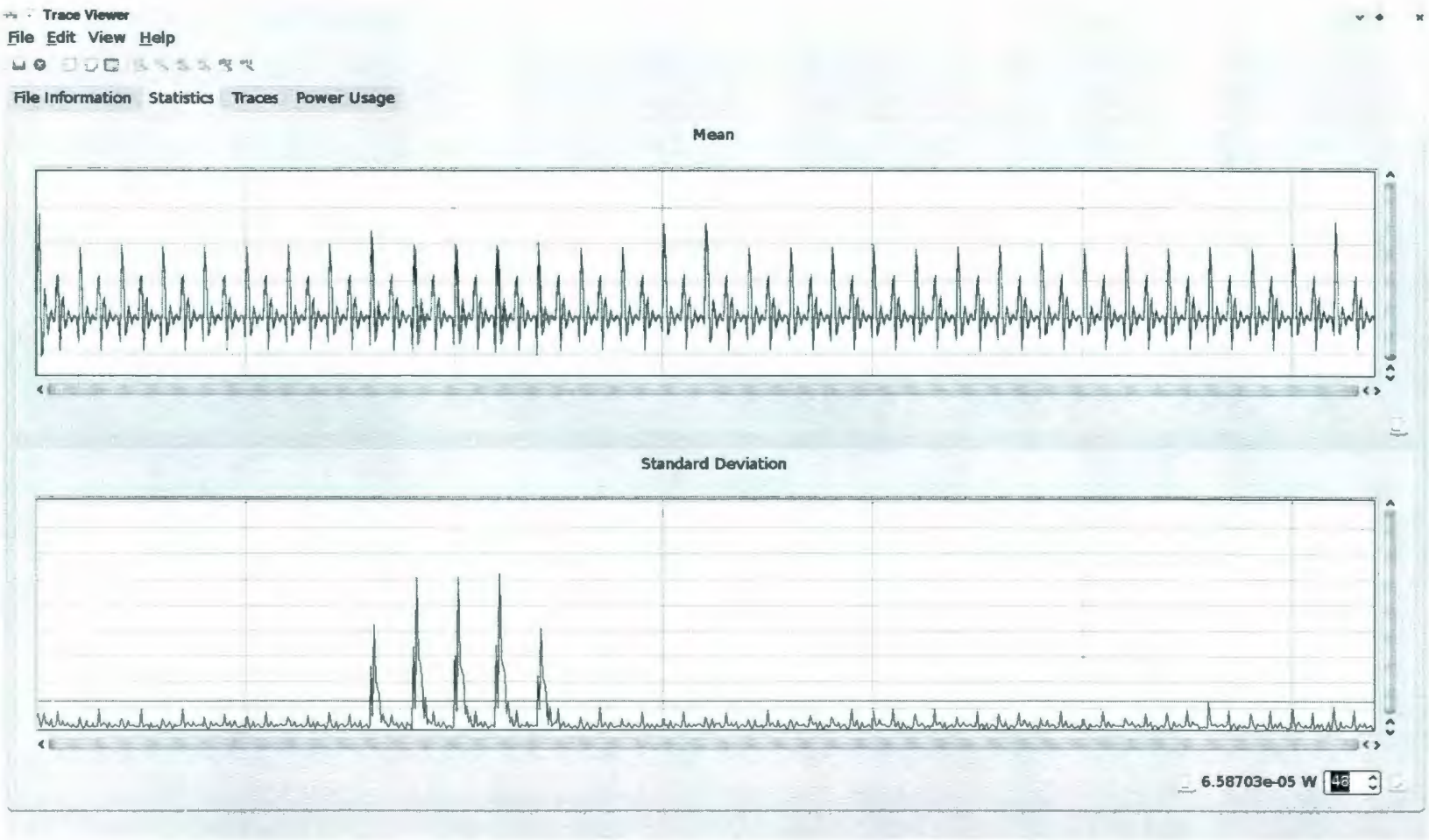

Figure 5.14: Hardware LFSR-16 statistics

for simulating hardware on the FPGA.

Using this power model, we simulated the operation of LFSR-16 hardware, and applied the template attack to its power usage. As expected, increased noise caused a decrease in classification success, but we were able to recover information about the secret key very successfully in many different noise conditions.

We then proceeded to apply the template attack to a real hardware implementation of LFSR-16. We were able to correctly guess secret key bits over 90% of the time, even with such small template sizes as $N = 12$.

Having successfully attacked LFSR-16 in both simulation and hardware, and having found good classification success with both, we proceeded to attack a simulated implementation of a real stream cipher: Trivium.

Chapter 6

Application of Template Attack to Trivium

Trivium is a candidate cipher for the eSTREAM stream cipher selection process (hardware profile) [5]. By applying Template Attacks, we were able to extract secret key material from a simulated version of this cipher.

6.1 Description

Trivium is a stream cipher that was developed for eSTREAM, a four-year effort to identify "promising new stream ciphers," some targeting software implementation and some targeting hardware [40]. Trivium is of the latter group, and it was designed "as an exercise in exploring how far a stream cipher can be simplified without sacrificing its security, speed or flexibility" [5].

Trivium has a 288-bit internal state which is updated through a combination of linear and non-linear feedback. It can generate up to 2^{64} bits of keystream from an 80-bit secret key and 80-bit initialization vector. It was designed to be implemented

in a parallel fashion: no state bit is used for 64 clock cycles after it is updated, so up to 64 iterations of the cipher can be calculated in parallel [5].

Precise specifications are given below, but intuitively, Trivium can be thought of as a collection of Feedback Shift Registers, as shown in Figure 6.1.

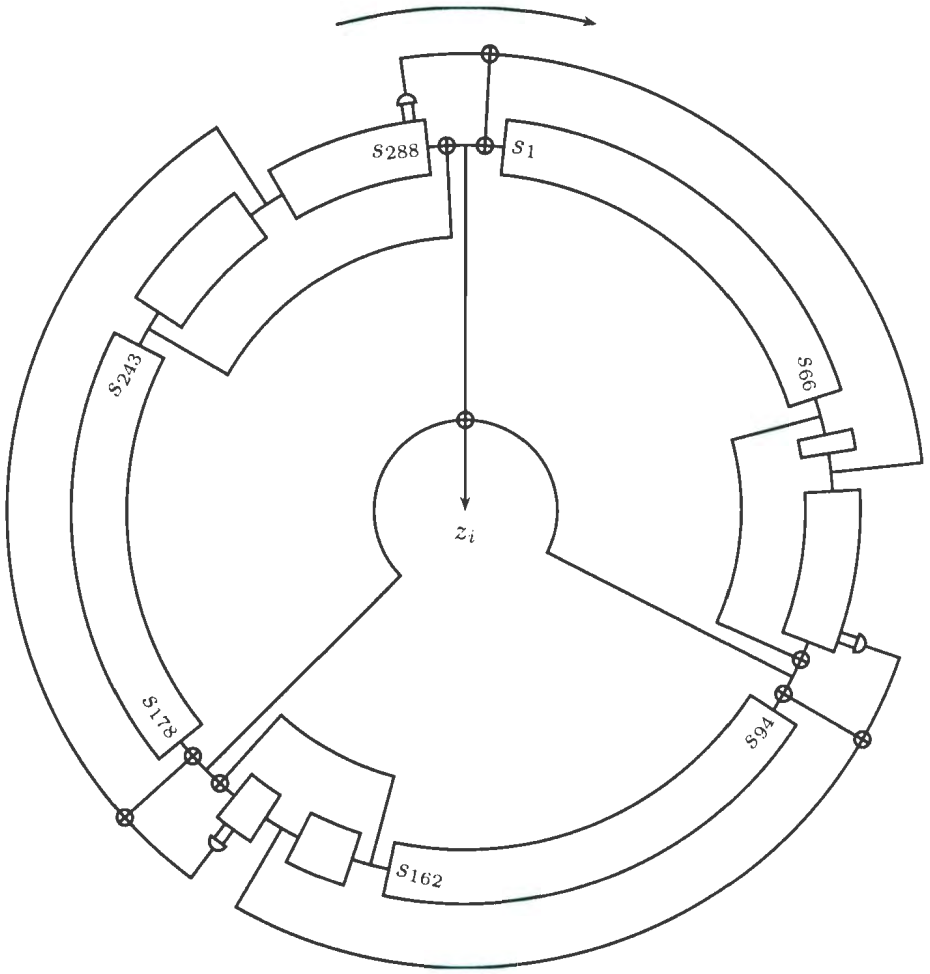

Figure 6.1: Trivium [5]

Before the keystream can be generated, the internal state has to be initialized. The state is initially loaded with the 80-bit secret key (state bits 0 - 79) and an 80-bit initialization vector (state bits 93 - 172). Three bits are then set to 1 (bits 286 - 288) and the remaining 125 bits are set to 0. The initialization procedure from Figure 6.2 is then followed, where `s[i]` is the i^{th} bit of the internal state and `t1`, `t2` and `t3` are

temporary variables.

```
for i = 1 to 4 * 288 do
   t1 = s[66] xor (s[91] and s[92]) xor s[93] xor s[171]
   t2 = s[162] xor (s[175] and s[176]) xor s[177] xor s[264]
   t3 = s[243] xor (s[286] and s[287]) xor s[288] xor s[69]

   (s[1],s[2],...,s[93]) = (t3,s[1],...,s[92])
   (s[94],s[95],...,s[177]) = (t1,s[94],...,s[176])
   (s[178],s[279],...,s[288]) = (t2,s[178],...,s[287])
end for
```

Figure 6.2: Trivium initialization

Keystream generation – shown in Figure 6.3 – is similar, but involves an output variable z, which is the current keystream output.

```
for i = 1 to N do
   t1 = s[66] xor s[93]
   t2 = s[162] xor s[177]
   t3 = s[243] xor s[288]

   z = t1 xor t2 xor t3

   t1 = t1 xor (s[91] and s[92]) xor s[171]
   t2 = t1 xor (s[175] and s[176]) xor s[264]
   t3 = t1 xor (s[286] and s[287]) xor s[69]

   (s[1], s[2], ..., s[93]) = (t3, s[1], ..., s[92])
   (s[94], s[95], ..., s[177]) = (t1, s[94], ..., s[176])
   (s[178], s[279], ..., s[288]) = (t2, s[178], ..., s[287])
end for
```

Figure 6.3: Trivium keystream generation

6.2 Simulation Results

Most Trivium simulations were performed with AWGN added, at a peak power noise of 10^{-7} W. This is a value which we found, for LFSR-16, produced success rates approximately equivalent to those obtained from hardware experimentation. In all cases, the right-most key bits were varied according to operation; the remaining bits were allowed to vary randomly.

6.2.1 Classification Success Rate vs. Template Size

Figure 6.4 shows our classification success rates for simulated Trivium versus template size. There are four lines on the graph:

- Maximum success rate
  - this is the highest classification success for any operation
  - e.g. if four operations $\{O^{(0)}, O^{(1)}, O^{(2)}, O^{(3)}\}$ had classification success rates $\{45\%, 32\%, 51\%, 29\%\}$, the maximum success rate would be 51%
- Average success rate
  - this is the average of classification success rates over all operations
  - e.g. if four operations $\{O^{(0)}, O^{(1)}, O^{(2)}, O^{(3)}\}$ had classification success rates $\{45\%, 32\%, 51\%, 29\%\}$, the average success rate would be 39.25%
- Minimum success rate
  - this is the lowest classification success for any operation
  - e.g. if four operations $\{O^{(0)}, O^{(1)}, O^{(2)}, O^{(3)}\}$ had classification success rates $\{45\%, 32\%, 51\%, 29\%\}$, the minimum success rate would be 29%

- "Guess" rate
  - this is how successful we would expect to be if we guessed randomly
  - this rate is $\frac{1}{2^n}$, where n is the number of bits included in the template
    * if we fixed four bits, we would have $2^4 = 16$ operations and the probability of a correct guess would be $\frac{1}{2^4} = 6.25\%$

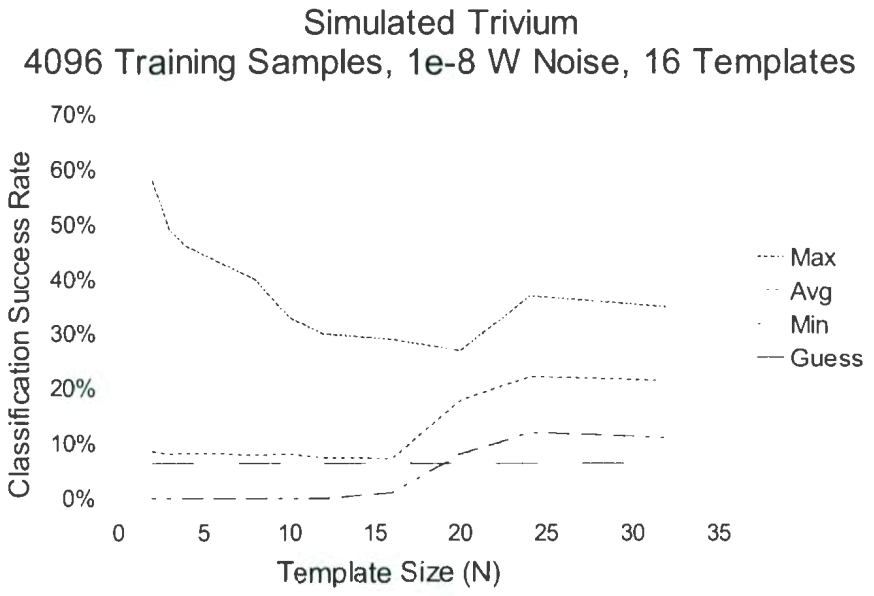

Figure 6.4: Classification success vs. template size Trivium

These success rates are lower than for LFSR-16 with the same amount of added noise, but average success rates as high as 22% were achieved better than the 6.25% that we would expect to achieve through random guessing.

6.2.2 Classification Success vs. Training Samples

As expected, increasing the number of training samples increased the probability of success, though success rates increased roughly linearly for exponentially increasing

numbers of samples. This is shown in Figure 6.5, where we can see that the maximum, average and minimum classification success rates are monotonically increasing with the number of training samples.

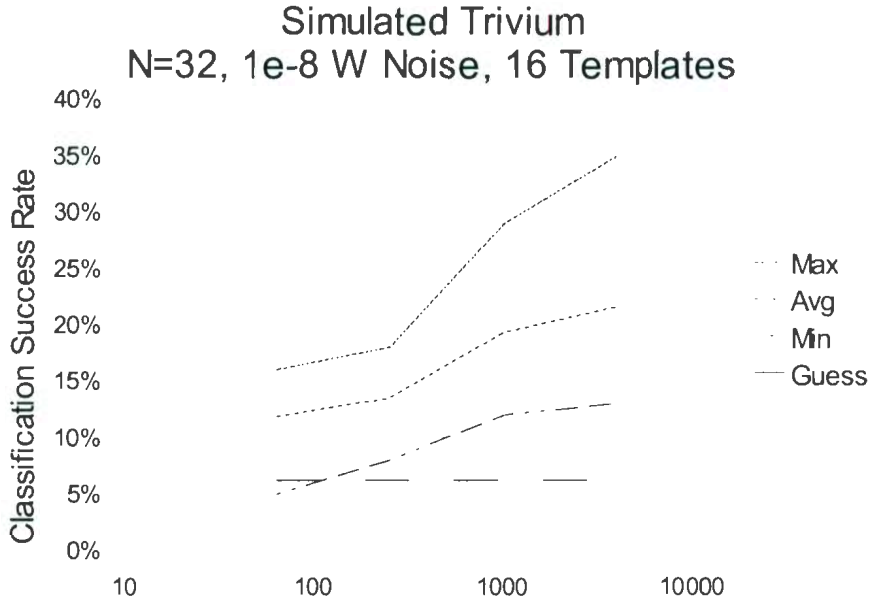

Figure 6.5: Classification success vs. training samples Trivium

For this research, we spent considerable time simulating the cipher under varying conditions. Simulating Trivium with 4,096 training samples per operation might only take an hour, but simulating the cipher's operation *and* performing analysis for varying template sizes might take a day. Thus, while using more than 4,096 training samples would be prohibitively time-consuming for this research, an individual or organization mounting a serious side channel attack could spend significant time and computational power - building templates from many training samples.

6.2.3 Classification Success Rate vs. Bits Under Attack

With Trivium simulations, we also varied the number of bits under attack, running simulations and analysis for one-bit templates ($2^1 = 2$ operations), two-bit templates

($2^2 = 4$ operations), four-bit templates ($2^4 = 16$ operations) and eight-bit templates ($2^8 = 256$ operations).

A linear increase in the number of bits under attack led to a exponential increase in the computation required to perform all simulation and analysis. On first inspection, however, maximum, average and minimum classification success rates all seem to vary exponentially with the *inverse* of the number of bits being attacked, as shown in Figure 6.6.

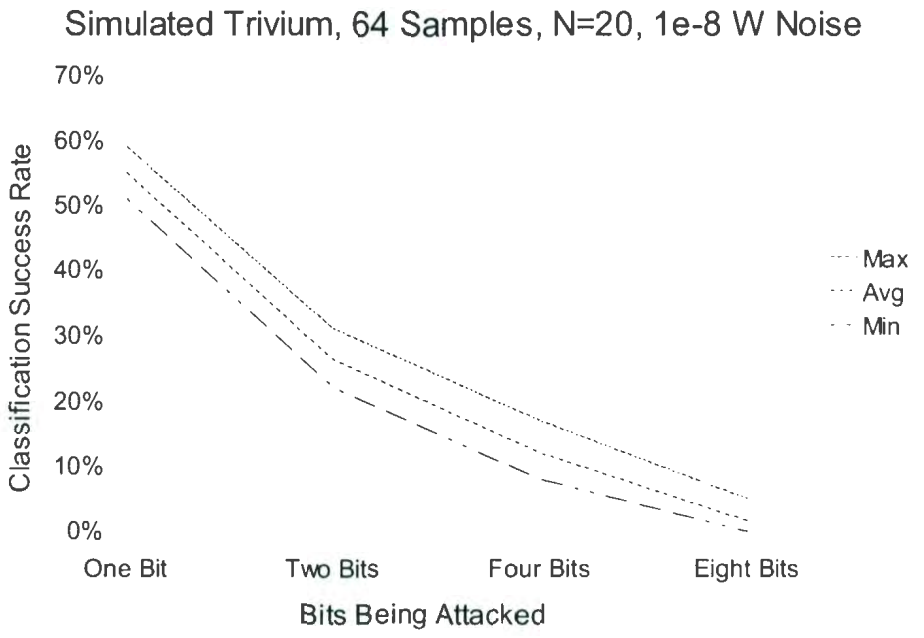

Figure 6.6: Trivium classification success vs. bits being attacked

What this graph does not reveal, however, is how the classification success compares to the *expected* classification success rate if we had no information about the cipher - i.e. if we guessed randomly. For n bits, we expect that random guessing would yield the correct subkey $\frac{1}{2^n}$ of the time. Our improvement over this rate tells us how much information each guess must reveal to enable as many correct guesses as we have made, and this information leakage can be calculated by Equation 6.1:

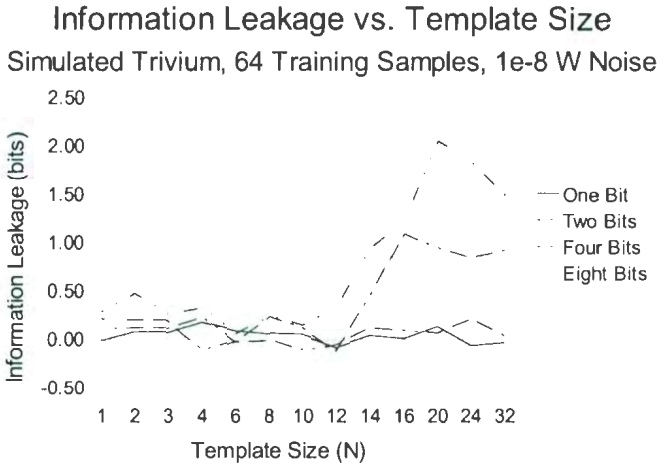

Figure 6.7: Trivium information leakage

$$l = B - \log_2\left(\frac{1}{s}\right), \tag{6.1}$$

where l is the information leakage, B is the number of bits being attacked and s is the classification success rate.

Figure 6.7 shows the information leakage for attacks on various numbers of bits. From this graph, we can see that the information obtained via attacking eight bits of key can be approximately twice that obtained from attacking four bits of key.

The attack is stronger as more bits are attacked, but this greatly increases computational complexity: if n is the number of bits being attacked, then 2^n templates must be generated, requiring $L2^n$ total template samples. As mentioned above, however, a serious side channel attack could be mounted on a system using resources such as computing clusters. This would make practical attack a very realistic possibility.

6.3 Trivium Hardware

We did not apply template attacks to the power usage of Trivium hardware, as we did with LFSR-16. The reason for this is entirely practical: feeding random key and IV values from attack software to the cipher hardware would require a more sophisticated experimental setup than we currently have. While this would be a logical attack for future work to implement, it is beyond the scope of this thesis.

Considering the similarity of our simulation and hardware results against LFSR-16, however, we conjecture that template attacks could be applied against real hardware implementations of Trivium.

6.4 Summary

Trivium is a very simple stream cipher which has withstood the rigours of the eSTREAM process, and is part of the final eSTREAM portfolio. Because of this, as well as its overwhelming popularity among stream cipher researchers [41], it is a very important cipher.

By applying template attacks, we were able to extract secret key material from a simulated version of Trivium. Our classification success rate was as high as 22% in noise conditions that we saw were reasonable in Chapter 5. This success rate could be increased by using more training samples per operation or by capturing multiple traces from the device under attack and exploiting the joint information contained in all of them.

We conjecture that these attacks could be realised against practical cryptosystems. Implementers of Trivium, and other practical stream ciphers, should take care to ensure that their implementations are not vulnerable to these attacks.

Chapter 7

Conclusions

As cryptography continues to be implemented in embedded systems such as smart cards and RFIDs, implementers of cryptographic systems must consider threat models that include adversaries having physical access to cipher hardware. This physical access enables attack via side channel analysis, including the powerful class of attacks known as template attacks.

In this thesis, we have demonstrated that template attacks can be applied to stream ciphers implemented not just via microcontrollers, but also in reconfigurable hardware. To this end, we have prepared an experimental setup that includes the Side Channel Analysis Board (SCAB), measurement equipment and software. SCAB is a custom PCB designed to support research in side channel analysis, with features to aid researchers in performing power analysis, electromagnetic analysis, fault analysis and timing analysis. In this research, we have used the power analysis features of SCAB, measuring the power used by stream cipher hardware with a PC-based oscilloscope called Cleverscope. We have also written 10,000 lines of C++ code to perform simulation and analysis of the power usage of cryptographic hardware.

Using this experimental setup, we measured the power usage characteristics of

FPGA-based hardware. Having found that these characteristics could be exploited for side channel analysis, we constructed a simple power usage model from them, which included the above characteristics and Additive White Gaussian Noise (AWGN). We simulated the operation of a stream cipher building block, a 16-bit Linear Feedback Shift Register (LFSR-16), and applied template attacks to its simulated power usage. We were able to recover secret key material from these simulated power traces success rates depended on the amount of AWGN present, but even with very high amounts of noise, success still exceeded the 6.25% rate that we would expect had we made random guesses at key bits. We then implemented LFSR-16 in hardware, measuring its power usage with the Cleverscope and analysing it with our software. We were able to recover secret key bits with success rates greater than 90%, even with small template sizes ($N < 20$).

From this success, we simulated the power usage of Trivium, a stream cipher that has been vetted by the eSTREAM initiative. For this complete stream cipher, we were able to retrieve four correct bits of key information for over 20% of our guesses, and our investigations indicate that higher success would be possible for a dedicated attacker with reasonable computational resources.

We thus conclude that side channel analysis is a very real threat to stream cipher hardware, and implementers of such hardware should take care to evaluate their implementations for susceptibility to this class of attacks.

Future Work

This thesis presents a black-box approach to attacking stream cipher hardware. Future work would include attacking different groups of bits within Trivium to determine the bits which are most or least susceptible to Template Attacks, as well as exploring

techniques to combine attacks so as to extract the maximum amount of key information possible.

Future work would also include more application of the method to physical hardware, especially the final eSTREAM portfolio ciphers (hardware focus) F-FCSR-H v2 [2], Grain v2 [3], MICKEY v2 [4] and Trivium [5]. This work will require a more elaborate experimental setup. The number of key and IV bits that must be determined randomly will be much larger - approximately 80 bits each which rules out the current method of IV generation: exhaustive search. Rather, unfixed key bits and all IV bits must be generated by hardware and/or software external to the device being tested - likely in software on the PC controlling the attack and exported to the hardware being analysed.

Other important future work is determining the effectiveness of traditional side channel countermeasures against the Template Attack. Many countermeasures were designed to defeat Differential Power Analysis, but the principles of the Template Attack are quite different. Whether or not they can be applied, and what techniques *are* effective at foiling the Template Attack, should be of particular interest to hardware designers.

Bibliography

[1] "Announcing the Advanced Encryption Standard," National Institute of Standards and Technology (NIST), Tech. Rep. FIPS 197, Nov. 2001.

[2] F. Arnault and T. Berger, "F-FCSR: design of a new class of stream ciphers," *Fast Software Encryption-FSE*, vol. 3557, pp. 83 97, 2005.

[3] M. Hell, T. Johansson, and W. Meier, "Grain a stream cipher for constrained environments," eSTREAM - ECRYPT Stream Cipher Project, Tech. Rep. 2005/010, 2005.

[4] S. Babbage and M. Dodd, "The stream cipher MICKEY-128," eSTREAM ECRYPT Stream Cipher Project, Tech. Rep. 2005/016, 2005.

[5] C. de Canniere and B. Preneel, "Trivium Specifications," *available from ESTREAM (http://www. ecrypt. eu. org/stream/triviump2. html)*.

[6] J. Muir, "Techniques of Side Channel Cryptanalysis," Master's thesis, University of Waterloo, 2001.

[7] S. Chari, J. Rao, and P. Rohatgi, "Template Attacks," in *Proceedings of Cryptographic Hardware and Embedded Systems*, vol. LNCS 2535, 2002, pp. 13 28.

[8] L. Smith, *Cryptography: The Science of Secret Writing.* Dover Publications, 1955.

[9] R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2001.

[10] N. Ferguson and B. Schneier, *Practical cryptography.* John Wiley & Sons, 2003.

[11] W. Diffie, P. Oorschot, and M. Wiener, "Authentication and authenticated key exchanges," *Designs, Codes and Cryptography*, vol. 2, no. 2, pp. 107–125, 1992.

[12] C. Shannon, "Communication theory of secrecy systems."

[13] "Data Encyption Standard (DES)," National Institute of Standards and Technology (NIST), Tech. Rep. FIPS 46-3, Oct. 1999.

[14] E. Foundation, M. Loukides, and J. Gilmore, *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design.* O'Reilly & Associates, Inc. Sebastopol, CA, USA, 1998.

[15] N. Mowlavi, "The Future of our Sun and Stars," *The Future of the Universe and the Future of our Civilization*, pp. 57 69, 2000.

[16] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[17] S. Garfinkel, *PGP: Pretty Good Privacy.* O'Reilly, 1995.

[18] D. Kahn, *The Codebreakers.* New York: Macruillan, 1967.

[19] J. Daemon, R. Govaerts, and J. Vendewalle, "A New Approach Towards Block Cipher Design," in *Fast Software Encryption, FSE 2003.* Springer-Verlag, 1993.

[20] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Advances in Cryptology: Proceedings of CRYPTO'96*, vol. 96. Springer-Verlag, 1996, pp. 104–113.

[21] H. Handschuh and H. M. Heys, "A Timing Attack on RC5," *Lecture Notes in Computer Science 1556: Selected Areas in Cryptography - SAC '98*, pp. 306 318, 1999.

[22] D. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of aes," *CT-RSA*, pp. 1–20, 2006.

[23] E. Biham and A. Shamir, "Differential Fault Analysis," in *Advances in Cryptology: Proceedings of CRYPTO '97*, vol. LNCS 1294. Springer-Verlag, 1997, pp. 513–525.

[24] D. Boneh, "On the Importance of Eliminating Errors in Cryptographic Computations," *Journal of Cryptology*, vol. 14, no. 2, pp. 101 119, 2001.

[25] J. Blömer and J.-P. Seifert, "Fault based cryptanalysis of the advanced encryption standard (aes)," *LNCS 2742: FC 2003*, pp. 162 181, 2003.

[26] S. Skorobogatov and R. Anderson, "Optical fault induction attacks," *Proceedings of CHES '02*, pp. 2–12, 2002.

[27] J.-J. Quisquater and D. Samyde, "Eddy current for magnetic analysis with active sensor," *Proceedings of Int. Conf. on Research in SmartCards (E-Smart 2002)*, pp. 185–194, 2002.

[28] R. Anderson and M. Kuhn, "Low Cost Attacks on Tamper Resistant Devices," in *5th International Workshop on Security Protocols*, vol. LNCS 1361. Springer-Verlag, 1997, pp. 125 126.

[29] ——, "Tamper Resistance - A Cautionary Note," in *Proceedings of the Second USENIX Workshop on Electronic Commerce*, 1996.

[30] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, 1999, pp. 388–397.

[31] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic Analysis: Concrete Results," *Cryptographic hardware and embedded systems-CHES 2001: Third International Workshop, Paris, France, May 14-16, 2001: Proceedings*, 2001.

[32] C. Rechberger and E. Oswald, "Stream Ciphers and Side-Channel Analysis," in *Workshop on the State of the Art in Stream Ciphers*, 2004, pp. 320–326.

[33] *Identification cards – Integrated circuit cards – Part 1: Physical characteristics*, International Standards Organization (ISO), Oct. 1998.

[34] S. F. Arnold, *The Theory of Linear Models and Multivariate Analysis*, ser. Wiley Series in Probability and Mathemetical Statistics. Wiley, 1981.

[35] C. S. Davis, *Statistical Methods for the Analysis of Repeated Measurements*, ser. Springer Texts in Statistics. Springer, 2002.

[36] R. Duda, P. Hart *et al.*, *Pattern classification and scene analysis*, 1973.

[37] A. Menezes, P. Van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.

[38] C. Su, T. Lin, C. Huang, and C. Wu, "A high-throughput low-cost AES processor," *Communications Magazine, IEEE*, vol. 41, no. 12, pp. 86–91, 2003.

[39] "Cleverscope." [Online]. Available: http://www.cleverscope.com/

[40] The eSTREAM Project. [Online]. Available: http://www.ecrypt.eu.org/stream/

[41] S. Babbage, C. D. Cannière, A. Canteaut, C. Cid, H. Gilbert, T. Johansson, M. Parker, B. Preneel, V. Rijmen, and M. Robshaw, "The eSTREAM Portfolio," ECRYPT, Tech. Rep., Apr. 2008.

[42] Qt cross-platform application framework. [Online]. Available: http://trolltech.com/products/qt/

Appendix A

Detailed Results

A.1 Simulation

A.1.1 LFSR-16

All LFSR-16 data is for an attack against four key bits.

16 Training Samples per Operation

Table A.1 contains the classification success rates when the peak power noise was 10^{-8} W.

N	Minimum	Average	Maximum
1	0	10.3%	51%
2	0	25.8%	56%
3	17%	48.5%	71%
4	40%	94.2%	100%
6	0	12.5%	100%
8	0	43.8%	100%
10	0	56.1%	100%
12	0	0	0
14	0	0	0

Table A.1: 16 training samples per operation (10^{-8} W noise)

Table A.2 contains the classification success rates when the peak power noise was 10^{-7} W.

N	Minimum	Average	Maximum
1	0	9.8%	59%
2	2%	25.2%	56%
3	17%	52.1%	70%
4	42%	94.1%	100%
6	50%	94.4%	100%
8	48%	94.9%	100%
10	54%	93.6%	100%
12	58%	89.5%	100%
14	24%	75.8%	100%

Table A.2: 16 training samples per operation (10^{-7} W noise)

Table A.3 contains the classification success rates when the peak power noise was 10^{-6} W.

N	Minimum	Average	Maximum
1	0	10.3%	59%
2	0	12.4%	30%
3	1%	14.9%	30%
4	9%	20.7%	39%
6	9%	21.8%	39%
8	11%	22.6%	36%
10	9%	22.9%	43%
12	4%	22.9%	41%
14	4%	23.6%	34%

Table A.3: 16 training samples per operation (10^{-6} W noise)

Table A.4 contains the classification success rates when the peak power noise was 10^{-5} W.

N	Minimum	Average	Maximum
1	0	6.1%	31%
2	0	6.6%	26%
3	0	7.7%	24%
4	0	8.1%	25%
6	0	9.8%	27%
8	0	10.7%	25%
10	0	12.8%	34%
12	0	12.4%	35%
14	0	12.6%	44%

Table A.4: 16 training samples per operation (10^{-5} W noise)

Table A.5 contains the classification success rates when the peak power noise was 10^{-4} W.

N	Minimum	Average	Maximum
1	0	6.7%	49%
2	0	7.9%	33%
3	0	7.9%	26%
4	0	8.8%	28%
6	0	10.2%	30%
8	0	11.6%	27%
10	0	12.3%	32%
12	0	12.8%	37%
14	0	12.5%	36%

Table A.5: 16 training samples per operation (10^{-4} W noise)

Table A.6 contains the classification success rates when the peak power noise was 10^{-3} W.

N	Minimum	Average	Average
1	0	6.5%	34%
2	0	6.6%	29%
3	0	6.4%	22%
4	0	8.1%	29%
6	0	9.4%	26%
8	0	9.5%	24%
10	0	11.3%	27%
12	0	12.3%	31%

Table A.6: 16 training samples per operation (10^{-3} W noise)

Table A.7 contains the classification success rates when the peak power noise was .01 W.

N	Minimum	Average	Maximum
1	0	7.8%	46%
2	0	7.4%	31%
3	0	7.7%	25%
4	0	8.0%	21%
6	0	9.1%	21%
8	0	11.7%	23%
10	0	12.7%	28%
12	0	13.2%	30%

Table A.7: 16 training samples per operation (.01 W noise)

Table A.8 contains the classification success rates when the peak power noise was .1 W.

N	Minimum	Average	Maximum
1	0	6.8%	59%
2	0	8.0%	56%
3	0	8.6%	70%
4	0	10.4%	100%
6	0	11.5%	100%
8	0	14.3%	100%
10	0	16.8%	100%
12	0	19.8%	100%

Table A.8: 16 training samples per operation (.1 W noise)

Table A.9 contains the classification success rates when the peak power noise was 1 W.

N	Minimum	Average	Maximum
1	0	6.6%	48%
2	0	8.5%	26%
3	0	9.1%	23%
4	0	9.6%	21%
6	0	11.8%	23%
8	0	12.6%	21%
10	0	16.8%	22%
12	0	19.3%	28%

Table A.9: 16 training samples per operation (1 W noise)

32 Training Samples per Operation

Table A.10 contains the classification success rates when the peak power noise was 10^{-8} W.

N	Minimum	Average	Maximum
1	0	8.2%	55%
2	0	28.2%	66%
3	16%	51.7%	77%
4	50%	95.0%	100%
6	53%	95.7%	100%
8	56%	95.9%	100%
10	54%	95.5%	100%
12	64%	96.2%	100%
14	72%	96.6%	100%
16	0	0	0
20	0	0	0
24	0	10.3%	89%

Table A.10: 32 training samples per operation (10^{-8} W noise)

Table A.11 contains the classification success rates when the peak power noise was 10^{-7} W.

N	Minimum	Average	Maximum
1	0	10.1%	65%
2	0	28.0%	56%
3	14%	53.2%	78%
4	50%	94.8%	100%
6	53%	94.8%	100%
8	58%	95.3%	100%
10	59%	95.0%	100%
12	53%	94.8%	100%
14	51%	94.4%	100%
16	51%	94.3%	100%
20	50%	92.6%	100%
24	54%	90.4%	100%

Table A.11: 32 training samples per operation (10^{-7} W noise)

Table A.12 contains the classification success rates when the peak power noise was 10^{-6} W.

N	Minimum	Average	Maximum
1	0	10.7%	47%
2	0	14.8%	38%
3	2%	19.5%	43%
4	9%	24.8%	41%
6	14%	26.3%	39%
8	16%	27.0%	42%
10	15%	28.4%	41%
12	18%	29.6%	40%
14	26%	31.4%	43%
16	20%	31.2%	40%
20	14%	32.6%	40%
24	7%	33.9%	40%

Table A.12: 32 training samples per operation (10^{-6} W noise)

Table A.13 contains the classification success rates when the peak power noise was 10^{-5} W.

N	Minimum	Average	Maximum
1	0	7.3%	43%
2	0	8.0%	24%
3	0	9.0%	23%
4	0	9.1%	24%
6	0	10.3%	26%
8	0	11.1%	32%
10	0	13.8%	32%
12	0	14.8%	37%
14	0	16.7%	40%
16	0	18.1%	44%
20	0	19.6%	45%
24	0	20.1%	43%

Table A.13: 32 training samples per operation (10^{-5} W noise)

64 Training Samples per Operation

Table A.14 contains the classification success rates when the peak power noise was 10^{-8} W.

N	Minimum	Average	Maximum
1	0	11.7%	40%
2	0	25.8%	58%
3	27%	53.5%	73%
4	52%	94.2%	100%
6	64%	95.8%	100%
8	62%	95.4%	100%
10	59%	95.1%	100%
12	56%	94.6%	100%
14	58%	94.9%	100%
16	56%	95.0%	100%
20	51%	94.9%	100%
24	41%	94.1%	89%

Table A.14: 64 training samples per operation (10^{-8} W noise)

Table A.15 contains the classification success rates when the peak power noise was 10^{-7} W.

N	Minimum	Average	Maximum
1	0	12.4%	63%
2	0	28.2%	56%
3	14%	55.6%	70%
4	50%	94.1%	100%
6	53%	95.3%	100%
8	58%	96.4%	100%
10	59%	96.8%	100%
12	53%	96.6%	100%
14	51%	96.8%	100%
16	51%	97.1%	100%
20	50%	97.2%	100%
24	54%	97.9%	100%

Table A.15: 64 training samples per operation (10^{-7} W noise)

Table A.16 contains the classification success rates when the peak power noise was 10^{-6} W.

N	Minimum	Average	Maximum
1	0	9.8%	48%
2	0	14.1%	36%
3	5%	18.3%	33%
4	3%	23.1%	38%
6	13%	26.0%	39%
8	10%	26.4%	42%
10	11%	29.6%	42%
12	13%	31.6%	49%
14	11%	32.8%	49%
16	12%	34.7%	50%
20	12%	38.1%	54%
24	8%	40.9%	61%

Table A.16: 64 training samples per operation (10^{-6} W noise)

Table A.17 contains the classification success rates when the peak power noise was 10^{-5} W.

N	Minimum	Average	Maximum
1	0	7.6%	46%
2	0	8.8%	34%
3	0	9.7%	32%
4	0	9.5%	24%
6	2%	11.0%	22%
8	5%	13.6%	23%
10	2%	15.2%	28%
12	2%	18.6%	32%
14	2%	20.6%	30%
16	4%	24.1%	34%
20	2%	32.3%	47%
24	2%	39.0%	55%

Table A.17: 64 training samples per operation (10^{-5} W noise)

Beyond the full data sets we collected, we also collected partial sets at 64 training samples for different noise values.

Table A.18 contains the classification success rates when the peak power noise was 10^{-4} W.

N	Minimum	Average	Maximum
24	1%	24.5%	57%

Table A.18: 64 training samples per operation (10^{-4} W noise)

Table A.19 contains the classification success rates when the peak power noise was 10^{-3} W.

N	Minimum	Average	Maximum
24	1%	37.4%	53%

Table A.19: 64 training samples per operation (10^{-3} W noise)

Table A.20 contains the classification success rates when the peak power noise was 10^{-2} W.

N	Minimum	Average	Maximum
24	1%	26.8%	53%

Table A.20: 64 training samples per operation (10^{-2} W noise)

Table A.21 contains the classification success rates when the peak power noise was .1 W.

N	Minimum	Average	Maximum
24	3%	37.4%	53%

Table A.21: 64 training samples per operation (.1 W noise)

Table A.22 contains the classification success rates when the peak power noise was 1 W.

N	Minimum	Average	Maximum
24	2%	27.0%	52%

Table A.22: 64 training samples per operation (1 W noise)

128 Training Samples per Operation

Table A.23 contains the classification success rates when the peak power noise was 10^{-6} W.

N	Minimum	Average	Maximum
1	0	10.8%	61%
2	0	14.4%	39%
3	5%	18.8%	33%
4	1%	22.1%	41%
6	7%	26.7%	46%
8	14%	28.9%	47%
10	14%	30.1%	52%
12	15%	31.1%	52%
14	11%	33.0%	58%
16	11%	34.2%	59%
20	11%	38.3%	65%
24	10%	42.8%	66%

Table A.23: 128 training samples per operation (10^{-6} W noise)

Table A.24 contains the classification success rates when the peak power noise was 10^{-5} W.

N	Minimum	Average	Maximum
1	0	8.1%	47%
2	0	8.5%	38%
3	0	10.2%	36%
4	0	10.1%	34%
6	0	11.0%	31%
8	0	11.7%	30%
10	1%	12.9%	28%
12	1%	13.6%	29%
14	1%	14.9%	32%
16	0	16.9%	37%
20	0	22.1%	55%
24	0	26.3%	57%
24	1%	34.9%	75%

Table A.24: 128 training samples per operation (10^{-5} W noise)

256 Training Samples per Operation

Table A.25 contains the classification success rates when the peak power noise was 10^{-7} W.

N	Minimum	Average	Maximum
1	0	10.5%	69%
2	1%	27.4%	62%
3	15%	54.6%	68%
4	58%	94.9%	100%
6	70%	96.5%	100%
8	70%	96.8%	100%
10	69%	96.3%	100%
12	70%	96.3%	100%
14	61%	95.9%	100%
16	60%	95.9%	100%
20	62%	96.4%	100%
24	66%	97.2%	100%

Table A.25: 256 training samples per operation (10^{-7} W noise)

Table A.26 contains the classification success rates when the peak power noise was 10^{-5} W.

N	Minimum	Average	Maximum
1	0	7.3%	36%
2	0	8.0%	36%
3	0	7.2%	31%
4	0	6.6%	25%
6	2%	6.9%	18%
8	2%	7.1%	12%
10	4%	7.8%	20%
12	3%	7.8%	18%
14	2%	7.4%	19%
16	3%	7.8%	22%
20	2%	8.1%	24%
24	0	8.1%	29%

Table A.26: 256 training samples per operation (10^{-5} W noise)

A.1.2 Trivium

All Trivium simulations, unless otherwise specified, were performed with 256 training samples and a peak power noise of 10^{-8} W.

A.1.2.1 One Key Bit

The results of attacking one key bit (79 bits randomly assigned) are given in Table A.27.

N	Minimum	Average	Maximum
1	43%	49.5%	56%
2	43%	53.0%	63%
3	44%	53.0%	62%
4	48%	57.0%	66%
6	46%	53.5%	61%
8	50%	52.5%	55%
10	48%	52.0%	56%
12	45%	47.5%	50%
14	44%	51.5%	59%
16	44%	50.5%	57%
20	51%	55.0%	59%
24	39%	48.0%	57%
32	35%	49.0%	49%

Table A.27: Trivium results - attacking one key bit

A.1.2.2 Two Key Bits

The results of attacking two key bits (78 bits randomly assigned) are given in Table A.28.

N	Minimum	Average	Maximum
1	1%	27.0%	42%
2	9%	27.3%	48%
3	10%	27.3%	44%
4	19%	29.5%	41%
6	21%	24.8%	29%
8	22%	25.0%	30%
10	17%	23.3%	27%
12	22%	24.3%	27%
14	24%	27.3%	30%
16	26%	26.8%	27%
20	22%	26.3%	31%
24	26%	29.0%	37%
32	23%	25.8%	39%

Table A.28: Trivium results - attacking two key bits

A.1.2.3 Four Key Bits

When attacking four key bits (76 bits randomly assigned), simulations were performed with power noise of 10^{-7} and 10^{-8} for several numbers of training samples.

64 Training Samples per Operation The attack results when the power noise is 10^{-8} W are given in Table A.29.

N	Minimum	Average	Maximum
1	0%	7.25%	47%
2	0%	7.19%	29%
3	1%	7.19%	24%
4	0%	5.81%	25%
6	1%	6.19%	17%
8	3%	7.38%	12%
10	2%	6.81%	14%
12	3%	5.81%	11%
14	1%	8.56%	16%
16	8%	13.3%	20%
20	8%	12.1%	17%
24	5%	11.3%	18%
32	5%	11.9%	16%

Table A.29: Trivium results - attacking four key bits, 64 samples, 10^{-8} peak noise

The attack results when the power noise is 10^{-7} W are given in Table A.30.

N	Minimum	Average	Maximum
1	0%	6.88%	32%
2	0%	7.00%	21%
3	1%	7.31%	21%
4	1%	6.00%	14%
6	2%	6.31%	12%
8	1%	5.94%	13%
10	1%	7.00%	14%
12	4%	6.50%	12%
14	1%	6.44%	13%
16	3%	6.69%	13%
20	1%	7.50%	13%
24	2%	6.25%	10%
32	2%	6.13%	10%

Table A.30: Trivium results - attacking four key bits, 64 samples, 10^{-7} peak noise

256 Training Samples per Operation The attack results when the power noise is 10^{-8} W are given in Table A.31.

N	Minimum	Average	Maximum
1	0%	7.2%	52%
2	0%	7.2%	34%
3	0%	7.2%	36%
4	0%	5.8%	28%
6	0%	6.2%	24%
8	1%	7.4%	22%
10	2%	6.8%	14%
12	2%	5.8%	14%
16	10%	13.3%	24%
20	6%	12.1%	21%
24	7%	11.3%	18%
32	8%	11.9%	18%

Table A.31: Trivium results - attacking four key bits, 256 samples, 10^{-8} peak noise

The attack results when the power noise is 10^{-7} W are given in Table A.32.

N	Minimum	Average	Maximum
1	0%	7.8%	32%
2	0%	8.5%	21%
3	0%	7.0%	21%
4	0%	7.8%	14%
6	1%	7.9%	12%
8	1%	7.7%	13%
10	2%	6.7%	14%
12	2%	7.6%	12%
16	3%	11.6%	13%
20	6%	10.2%	13%
24	4%	9.3%	10%
32	2%	8.5%	10%

Table A.32: Trivium results - attacking four key bits, 256 samples, 10^{-7} peak noise

1024 Training Samples per Operation The attack results when the power noise is 10^{-8} W are given in Table A.33.

N	Minimum	Average	Maximum
1	0%	6.9%	55%
2	0%	7.1%	43%
3	0%	7.7%	36%
4	0%	7.5%	29%
6	1%	7.4%	23%
8	2%	7.0%	21%
10	1%	7.4%	17%
12	1%	7.0%	17%
16	6%	16.6%	27%
20	10%	16.4%	28%
24	11%	20.1%	32%
32	12%	19.3%	29%

Table A.33: Trivium results - attacking four key bits, 1024 samples, 10^{-8} peak noise

4096 Training Samples per Operation The attack results when the power noise is 10^{-8} W are given in Table A.31.

N	Minimum	Average	Maximum
1	0%	8.6%	58%
2	0%	8.0%	49%
3	0%	8.2%	46%
4	0%	8.1%	43%
6	0%	7.8%	40%
8	0%	8.1%	33%
10	0%	7.4%	30%
12	1%	7.2%	29%
16	8%	17.8%	27%
20	12%	22.1%	37%
24	11%	22.4%	35%
32	13%	21.6%	35%

Table A.34: Trivium results - attacking four key bits, 4096 samples, 10^{-8} peak noise

A.1.2.4 Eight Key Bits

The results of attacking eight key bits (72 bits randomly assigned) are given in Table A.35. 64 training samples were used in all cases.

N	Minimum	Average	Maximum
1	0%	0.48%	21%
2	0%	0.54%	21%
3	0%	0.47%	9%
4	0%	0.49%	8%
6	0%	0.41%	9%
8	0%	0.46%	17%
10	0%	0.43%	4%
12	0%	0.49%	3%
14	0%	0.75%	5%
16	0%	0.92%	5%
20	0%	1.63%	5%
24	0%	1.39%	6%
32	0%	1.10%	5%

Table A.35: Trivium results - attacking eight key bits

A.2 Physical Measurement

Physical measurement was performed of the LFSR-16 cipher building block for 256 training samples. Results are given in Table A.36

N	Minimum	Average	Maximum
1	0	11.3%	40.6%
2	0	26.7%	59.0%
3	29.7%	52.3%	58.6%
4	60.2%	65.9%	73.4%
6	65.2%	73.0%	80.5%
8	72.3%	85.6%	95.7%
10	72.7%	88.2%	96.1%
12	77.7%	91.6%	96.5%
16	85.2%	94.1%	97.7%
20	93.4%	97.4%	99.6%
24	95.7%	98.5%	100%
32	97.7%	99.4%	100%
40	98.8%	99.7%	100%

Table A.36: Physical measurement results

Appendix B

Software Data Formats

B.1 Cleverscope Text Files

The Cleverscope text-based format has a header, beginning with the line "`[Sample Definition]`" and a body, beginning with the line "`[Data]`". An example of this format is shown in Figure B.1.

B.1.1 Header

The header of a Cleverscope text file contains several pieces of information important to our analysis:

- Usage of digital traces
  - If digital traces were captured by the Cleverscope unit, the `UseDig` parameter is `TRUE`; otherwise, it is `FALSE`.
- Analog scale, offset

```
[Sample Definition]
Type=Time
UseBuffer=FALSE
UseDig=TRUE
ChAscale=1.000000
ChAoffset=0.000000
ChBscale=1.000000
ChBoffset=0.000000
delta=0.0000000100
start=0.0006427900
nsample=7047
offset=0
Save Time =      8/20/2007 2:35:46 PM
[Data]
Time              Chan A          Chan B          Dig
0.00064279        1.50011814      1.48075295      240.00000000
0.00064280        1.49992914      1.47955595      240.00000000
0.00064281        1.50005514      1.48058195      240.00000000
0.00064282        1.49879514      1.47915695      240.00000000
0.00064283        1.49948814      1.47927095      240.00000000
0.00064284        1.49911014      1.47852995      240.00000000
0.00064285        1.49961414      1.47898595      240.00000000
0.00064286        1.49904714      1.47972695      240.00000000
0.00064287        1.49929914      1.47938495      240.00000000
...
```

Figure B.1: Cleverscope text file example

  - Each analog channel (A and B) has a scale and an offset associated with it; these values must be multiplied with and added to, respectively, the analog channel data specified below.

- Sampling period

  - The time between samples is given by the `delta` parameter. While the sampling period does not affect template attacks directly, we do read and store it to ensure that we only attempt to add or multiply traces with the

same sampling period.

- Number of samples
  - The number of sample points in the trace is given by the `nsample` parameter. After loading sample points from the file, we ensure that the entire file was loaded by comparing the number of loaded points to `nsample`.

Other parameters, such as "`Save Time`", are not important for the research, but are nonetheless parsed and saved.

B.1.2 Body

The body of a Cleverscope text file contains tab-delimited lines of data in four columns:

1. Time: the time, in seconds, that the data was sampled
2. Chan A: the voltage measured by Channel A
3. Chan B: the voltage measured by Channel B
4. Dig: digital trace values
   (a) This number varies between 0 and 255, and represents the values of all eight digital traces
   (b) Retrieving a particular trace's value is a matter of bit masking:

```
for(int j = 0; j < 8; j++)
    digitalTraces[j]->append(value & (1 << j));
```

B.2 Analog Trace Files

An AnalogTrace C++ object has six attributes:

Name	Type	Description
myName	QString	Name of the trace (e.g. "Channel A")
unit	Unit*	Unit of trace values (e.g. Volts, Watts)
timeDivision	double	Time between samples
trace	QList<double>	Actual trace values
minValue	double	Smallest value in the trace
maxValue	double	Largest value in the trace

QString and QList are data structures from the Qt C++ toolkit [42], `double` is the 64-bit IEEE-standard C++ primitive and Unit is a class that we wrote to manage trace units (e.g. dissimilar units cannot be added, multiplying an Amp by a Volt produces a Watt).

Such a trace can be written to two types of files: text-based or binary.

B.2.1 Text

When writing small traces to file, we may choose to write them in a text-based format that facilitates direct inspection. This is accomplished via the Qt class QTextStream. A QTextStream object, which is associated with a QFile object, can be used to read or write primitives such as strings and double-precision floating-point numbers. An example of the output is shown in Figure B.2.

```
AnalogTrace ("Mean Power Usage for Unnamed Trace", W, 192
   values, 1e-06s apart, range [1.11306e-05:0.000221819]){
    0.000221806 1.11333e-05 1.11326e-05 0.000221819
   1.11322e-05 1.11331e-05 0.000221815 1.11339e-05 1.11316
   e-05 0.000221804 1.11328e-05 1.11315e-05 0.000215942
   ...  }
```

Figure B.2: Example of a text-based AnalogTrace file

B.2.2 Binary

When writing files that are large or will be read many times, it is more efficient to write AnalogTrace objects in a binary format. Such a format is smaller than the equivalent text-based format, and it saves the computational effort required to parse floating-point numbers from text.

Writing a AnalogTrace to a binary file or reading it back is accomplished using the Qt class QDataStream. Like QTextStream above, QDataStream object can be used to read or write primitives such as strings and double-precision floating-point numbers. The binary format includes a "magic" number used to recognize the format and a binary format version (currently version 2). The process of writing such a file is shown in Figure B.3, and a sample trace as viewed in a hex editor is shown in Figure B.7.

B.3 Digital Trace Files

Digital trace files are much simpler than analog traces, as they contain a binary trace there are no units or minimum/maximum values to be concerned with. An DigitalTrace C++ object has just two attributes:

```
QDataStream& power::operator << (QDataStream& d,
                                 AnalogTrace& trace)
{
   d << (quint32) 0x5CAB00A7; //  magic : SCAB  AT  ( Analog Trac
   d << (quint32) 2;          //           binary  format  version
   d << trace.name();
   d << trace.units().toString();
   d << trace.period();
   d << trace.values().size();

   for(long int i = 0; i < trace.values().size(); i++)
      d << trace.values()[i];

   return d;
}
```

Figure B.3: Writing a binary AnalogTrace file

Name	Type	Description
timeDivision	double	Sampling period
trace	QList<bool>	Binary trace

Digital traces are are also simpler to parse than analog traces – there is only one floating-point number per file – and in our usage, they are also much smaller, since we only use them for subtrace masking, and subtraces are much smaller than full traces (see Section 4.4.2). Thus, we only write digital trace files in a text-based format, though a binary representation is required when writing digital traces as part of power usage files.

B.3.1 Text

The text-based digital trace file format is quite simple, as shown in Figure B.5, incorporating just the trace length, sampling period and actual trace values.

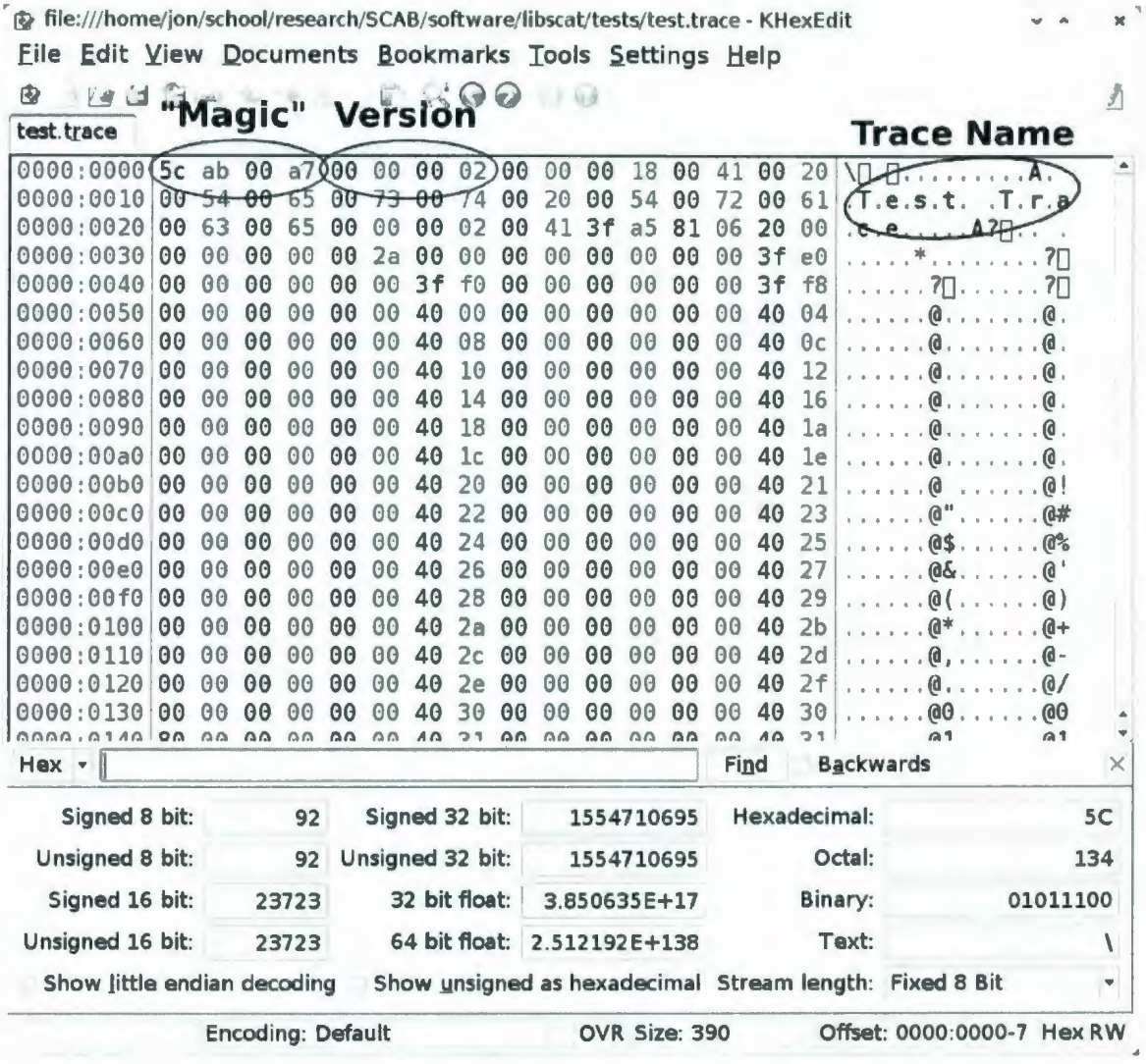

Figure B.4: Example of a binary AnalogTrace file

B.3.2 Binary

Like that of an AnalogTrace, the DigitalTrace's binary representation uses a "magic" value for the purposes of format recognition and a format version – currently version 1. The code to write such a file is shown in Figure B.6.

```
DigitalTrace (192 values, 1s apart){
   1001001001001001001 00...0000 }
```

Figure B.5: Example of a text-based DigitalTrace file

```
QDataStream& power::operator << (QDataStream& ds,
                                 DigitalTrace& t)
{
   ds << (quint32) 0x5CAB00D7; //  magic : SCAB  DT  ( Digital Tr
   ds << (quint32) 1;          //           binary  format  versio
   ds << t.period();           //            sampling  period
   ds << t.size();             //              size

   const QList<bool> values = t.values();
   ds << values;

   return ds;
}
```

Figure B.6: Writing a binary AnalogTrace file

B.4 Power Usage Files

A PowerUsage file is a binary representation of two things:

- an AnalogTrace containing a power trace
- a DigitalTrace that partitions the trace into subtraces (see Section 4.4.2

This file consists of another "magic" number, a version (current version 1), two binary values (to indicate the presence of an analog and digital trace, respectively) and then the binary representations of the power trace and partitioning trace. An example of this format is shown in Figure B.8.

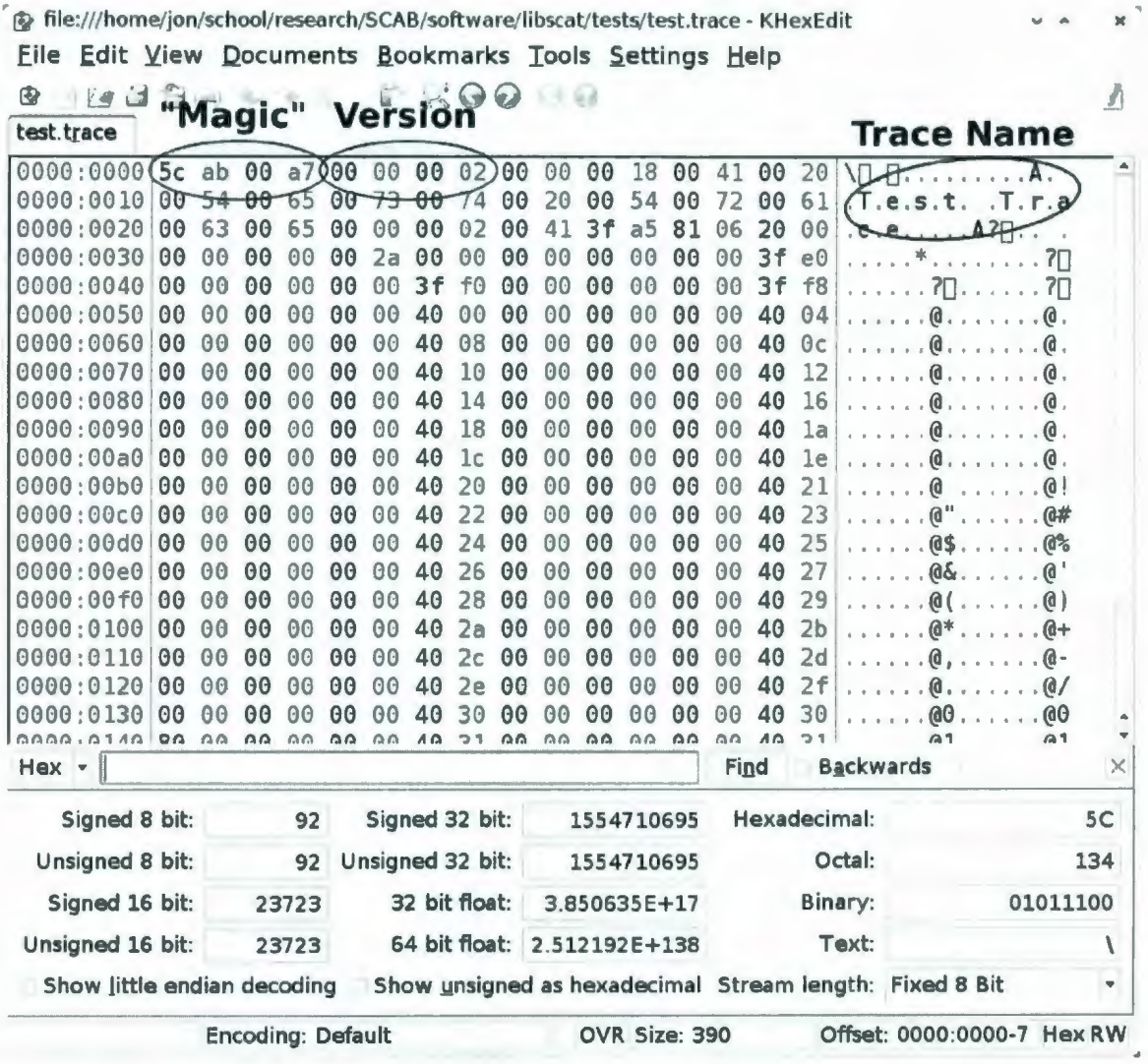

Figure B.7: Example of a binary AnalogTrace file

B.5 Power Simulation

Hardware was modeled in this work in two parts: first the hardware itself was characterized, then hardware power usage was simulated using these characteristics as a model.

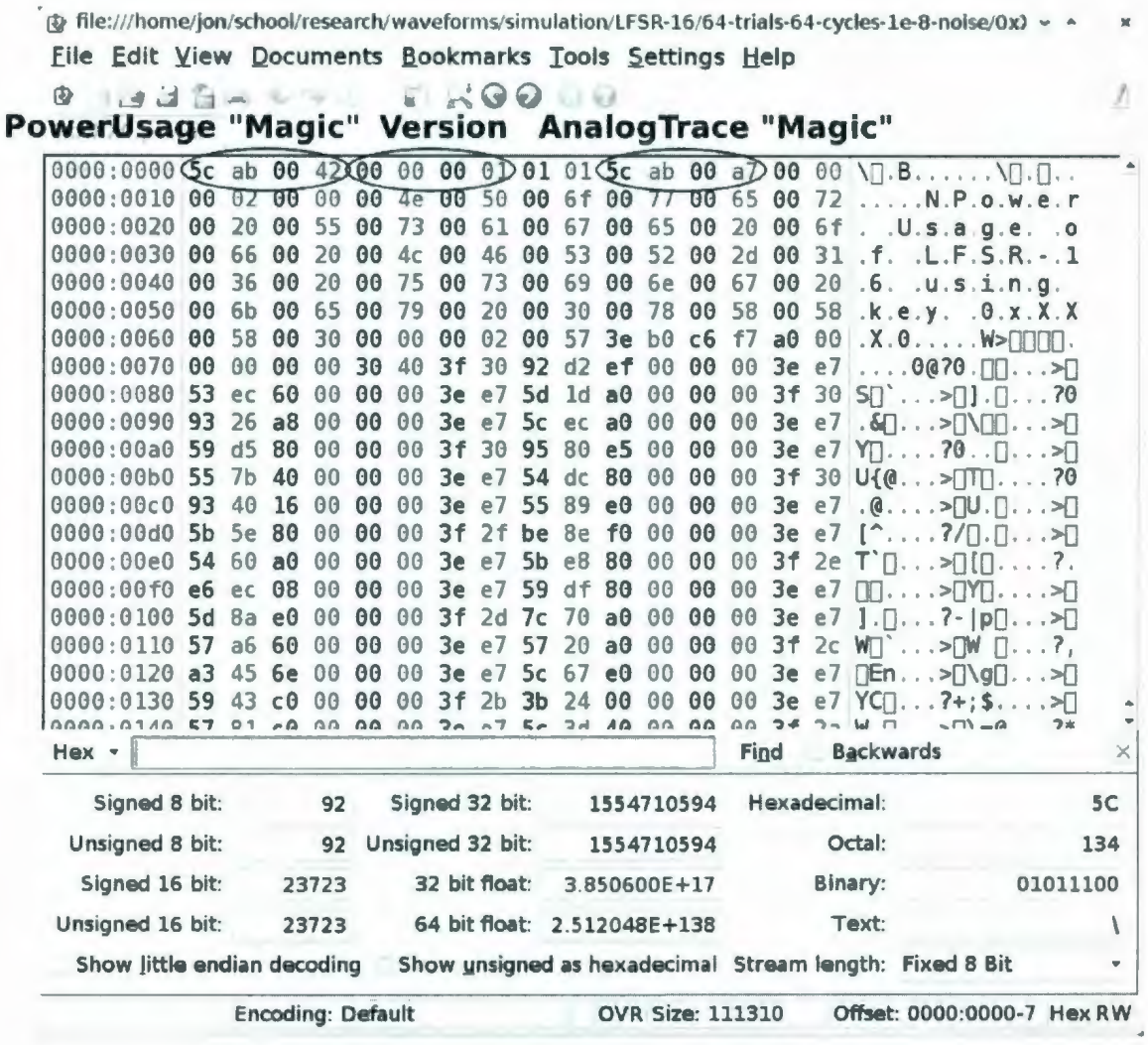

Figure B.8: Example of a binary PowerUsage file

```
class PowerUsageModel
{
   public:
   double noiseLevel() const;        //        !< Amount   of   AWGN   in   p
   void setNoiseLevel(double);       //       !< Set   amount   of   AWGN   i

   virtual float basic() const = 0;
   virtual float zeroToOne() const = 0;
   virtual float oneToZero() const = 0;

   protected:
   float noise() const;              //              !< Noise
   float noise(float scale) const;   //  !< Noise   in   a   specific   +-

   private:
   double myNoiseLevel;              //              !< Amount   of   AWGN
};
```

Figure B.9: PowerUsageModel interface

B.5.1 Power Model

Once hardware has been characterized, a C++ class can be written which implements the PowerUsageModel interface, which is shown in Figure B.9.

B.5.2 Cipher Model

Simulating hardware requires simulating the number of high-low and low-high transitions of a cipher. A Qt/C++ class (a C++ class using the Qt class library and preprocessed by Qt's Meta Object Compiler MOC) must be written which inherits from the abstract class Cipher, shown in Figure B.10.

```
class Cipher : public QObject
{
   public:
   //! Represents what happens when an cipher changes state
   struct StateChange
   {
      // ...
      int ll; int lh; int hl; int hh;
   };


   //! The cipher 's name
   virtual QString name() const = 0;

   //! Current cipher state ( should be human - readable)
   virtual QString stateString() const = 0;

   virtual int minimumKeySize() const = 0;
   virtual int maximumKeySize() const = 0;
   virtual int minimumIVSize() const = 0;
   virtual int maximumIVSize() const = 0;

   //! Initialize the cipher for use
   virtual void initialize(const Cryptovariable& key,
                           const Cryptovariable& iv) = 0;

   virtual void initialize(const QList<bool>& key,
                           const QList<bool>& iv) = 0;

   /**
    * Cycle the clock .
    *
    * @returns a StateChange class , representing the number of
    *          low          -> high and high -> low transitions , etc
    */
   virtual StateChange clock() = 0;

   signals:
   //! The internal state has changed
   void newState(QString state);
};
```

Figure B.10: Cipher interface