Formalizing Secure Computation for Embedded Systems

Jason Waddle

December 21, 2004

Chapter 1

Introduction

For any discussion of the security of a system to be meaningful, it must be framed in a context of assumptions—whether explicit or implicit—that restrict, among other things, the nature of any adversary. These assumptions can generally be lumped into two categories: those that specify how the adversary interacts with the system and those that limit the adversary's computational power.

As an illustration of these assumptions, consider the case of creating a secure channel over an insecure channel. In the traditional presentation we have two participants, Alice and Bob. They can communicate over a public, insecure channel. By "insecure," we mean that there is a third player, Eve, who listens to the channel between Alice and Bob. This situation is depicted in Figure 1.1. Alice and Bob wish to to communicate in such a way that Eve cannot tell what they are saying. Unless we allow some extra secret known only to Alice and Bob, their only hope for achieving their goal is by utilizing randomness and assuming some limit on Eve's computational ability (e.g., her inability to solve a discrete logarithm problems in a reasonable amount of time).

In this case, the nature of Eve is dictated by two assumptions. First, she can interact with the system only by observing messages passing between Alice and Bob; she cannot intercept or modify messages, and she certainly cannot observe or influence Alice or Bob's internal actions. Second, she is computationally limited in some particular way.

Obviously, security guarantees are stronger when we consider the most general types of adversaries by weakening both assumptions about how Eve interacts with the system and assumptions about Eve's computational limitations. Traditionally, much of the work has gone into weakening the latter sorts of assumptions since they are frequently more contentious and generally come in the form: "we do not know how to do it, so it's safe to assume Eve doesn't know how to do it either"; clearly, these types of arguments require ample justification to be effective.

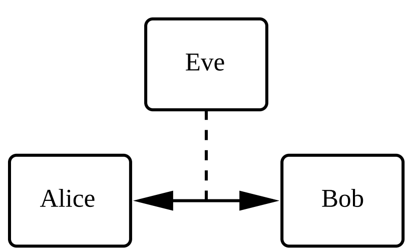

Figure 1.1: The classic model for cryptography. Alice and Bob (good) compute privately and communicate across a public channel. Eve (bad) can read from and possibly write to the channel.

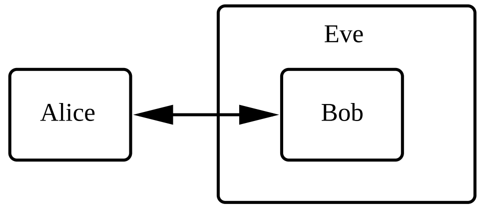

Figure 1.2: A possible representation of the embedded computing situation. While Alice enjoys private computation, Bob is somewhat exposed to Eve. Not only does Eve control the channel between Alice and Bob, but she may observe artifacts of Bob's computation and may even be able to modify Bob's operation. The nature and extent of Eve's power over Bob are details to be specified by the particular model of embedded computing.

Assumptions of the former type, those that specify how Eve interacts with the system, are typically motivated by more practical concerns, and there is a tendency to take them for granted. Some obvious variations on the standard assumptions are practically motivated and well studied. For instance, if Eve can tap the public channel, it might be reasonable to assume she can also intercept, corrupt, and send her own messages. This fatally complicates the task of setting up a secure channel between Alice and Bob since Eve can now do at least as much (i.e., send to the channel) as either Alice or Bob.

What has not received as much attention, however, are variations on the assumption of Alice and Bob's *private* computation—the assumption that all that Eve learns about the system is what she observes on the public channel. The sanctity of private computation is usually justified by the following argument: with the exception of timing information, all reasonable interaction with Alice or Bob's private computation seems to require physical proximity to Alice or Bob, and this can be prevented with mundane, real-world techniques.

Unfortunately, it seems that the other motivation for imagining that all computation is private is more convenient than practical. It's simply the case that "nonprivate computation" is a very slippery concept to capture in a formal model. What does it mean for Eve to have access to Bob's inner workings? Clearly, if Bob is just Eve's puppet, private communication with Bob doesn't make sense. Then how do we model restricted access to Bob's inner workings? How can we get a grasp on what Eve can learn about by measuring some by-product of his computation, or, worse yet, by slightly modifying Bob's computation? And how do we do all of this while making minimal assumptions about the nature of Alice, Bob, and Eve's physical realization? Figure 1.2 attempts to depict the situation where Bob is, at least to some extent, in Eve's control.

1.1 Embedded versus Private Computation

A practical motivation for dismissing the assumption of private computation comes from the increasing use of *embedded* computing. Rather than computation occurring on a traditional computer in some isolated, controlled environment, applications are increasingly depending on specialized computational devices deployed in the wild. Examples of such devices include smart cards, cell phones, digital ID tokens, and increasingly sophisticated microcontrollers in everything from televisions to kitchen appliances to automobiles.

While it can be argued that what most people imagine is the primary focus of much conventional computer security research—servers safely humming away under fluorescent lights in air-conditioned, locked rooms—meet the private computing assumptions of the standard models, these same assumptions are implicitly applied in many situations where they are not appropriate. One class of systems that certainly do not deserve to enjoy private computing assumptions are embedded devices, which need to operate in environments largely out of the control of their designers, in the hands of operators (e.g., Eve herself) who don't have same interests in mind as the designer.

Experience has shown that Eve can leverage her physical control in a variety of ways, but they tend to fall into two categories: *passive* attacks, where Eve observes some unintended by-product of Bob's computation, and *active* attacks, where Eve purposefully modifies Bob's computation in order to learn something from the possibly erroneous output.

1.1.1 Passive Attacks

The most obvious passive attacks (though the name *passive* seems less appropriate here) are ones where the attacker exposes the circuitry of the device in question and simply probes the values on the wires. This type of attack requires some specific knowledge of the device under attack and can also be thwarted by physical countermeasures that, for example, zeroize the memory of the device when the physical integrity of the casing is compromised. Other defenses (such as some we will discuss later) use randomness to mask the useful data carried across a single wire, forcing a probing attacker to measure several wires simultaneously; due to physical constraints, this might be considerably more difficult or expensive for the attacker.

Of course non-intrusive attacks also exist, and defenses that rely on detection do not work against them. A popular example of a non-intrusive passive attack is *power analysis*. Embedded devices typically rely on either an external power source or a replaceable battery, and in either case, the channel that carries electrical power to the device is exposed. It is fairly simple and inexpensive to measure the instantaneous current drawn by the device without affecting its operation. Depending on the design, knowledge of the instantaneous current consumption can reveal quite a bit about any secret data used in the computation. Careful designs will not exhibit dramatically differing behavior based on secret bits, but even the unavoidable task of loading a bit value into a register can cause a tiny data dependence in the current consumption.

Other locally observable artifacts of modern computation are fluctuations in the nearby electric and magnetic fields. Using a simple induction coil, for example, an attacker can observe these fluctuations in the magnetic field around the device during computation. By mapping the degree of response at various locations around the device, the attacker can map the layout, locating busses, memory, processors, etc., without depackaging or even touching the device. Aside from just learning about the implementation, however, the attacker can potentially exploit the data dependence of the EM field fluctuations in the same manner as the data dependent power consumption. Although measurements of the EM field tend to be more noisy than measurements of current consumption, the choice of physical location for the EM measurement can potentially *select* its source within the circuit—circumventing some countermeasures that alleviate data dependence of the total (i.e., whole circuit) current consumption.

1.1.2 Active Attacks

While passive attacks tend to resemble each other—they all tend to read some intermediate value in the computation, *active* attacks can be expected to have only one common feature: the attacker intentionally modifies the operation of the device. If the attack is successful, the resulting behavior of the device, and possibly its erroneous output, reveal something about the secret data.

Active attacks are frequently *ad-hoc* in nature, and are therefore difficult to classify; in order to appreciate active attacks, it is best to consider some examples.

Probably the simplest technique for introducing error is the *power glitch*. As with power analysis, with power glitching, the attacker is leveraging her control over the power source to the device. Instead of (or possibly in addition to) measuring current consumption, the attacker intentionally alters the voltage outside of the range required for normal operation during some cleverly chosen time interval. The effect may be as focused as to cause the device to skip exactly one security-critical operation or as blunt as to just cause the device to compute the wrong value during the evaluation of some extended arithmetic procedure (e.g., modular exponentiation).

While active attacks designed to cause the device to produce erroneous output that reveals secret data can be the most impressively effective—sometimes immediately revealing some or all bits of the key—they are often straightforward to defend against, even with general techniques. For instance, to prevent the presentation of erroneous output resulting from a single induced error, a device can simply compute the desired function twice, and if the results do not match, output error. This exemplifies the general active attack countermeasure of *error-detection*.

As we shall see in Chapter 5, however, a device that alters its behavior upon detecting errors may reveal secret bits if the attacker is capable of injecting data dependent errors. For this reason, *error-correction* is a more robust, though more costly, generic countermeasure for active attacks.

A generic method for producing efficient designs that are simultaneously resistant to power analysis attacks (i.e., use balanced logic) and provide error correction capability is an elusive goal. As we argue below, a more thorough understanding and rigorous model of semiprivate or embedded computation is needed before we can seriously consider achieving any such goal.

1.2 The Case for a Theoretical Approach

Much of the research in the area of security for embedded systems has been very applied, and the majority of the techniques in the literature are *ad-hoc*. As we discuss in this section, this does not provide a good foundation for good, principled, secure designs.

1.2.1 Attacking vs. Designing for Security

One of the major benefits of a theoretical approach is most obvious when contrasting analysis of attacks versus analysis of a system's ability to thwart attacks. Attacks are frequently *ad-hoc* in nature; they seize on a particular aspect of the target system and exploit it. It can be helpful to have generalized attacks, but it is not really necessary. Attacks also do not require rigorous analysis; a simple demonstration of the efficacy of an attack is sufficient to cast doubt on the security of a system, even in the absence of an explanation of the attack's mechanism.

Designing for security, on the other hand, requires both generality and rigor. Generality is necessary because the designer is required to defend against a variety of types of attacks from attackers with varying capabilities; it is often not realistic to expect the designer to anticipate the specifics of his adversaries. The cost of such generality, however, is the complementary requirement of increased rigor; a vague adversary does not permit a vague analysis. On the contrary, assumptions, implications, and details need to be explicitly identified, their relations thoroughly examined, and all consequences checked. This is where the theoretical approach shines.

1.2.2 Research

In a more practical direction, a theoretical methodology also facilitates research in a couple of important ways.

First, by explicitly establishing assumptions in the form of well-defined models, differing results may be more easily compared. Even if the results are the product of analyses in slightly different models, it is usually straightforward to compare the differences in assumptions as well as the differing characteristics of the designs in their respective models, or to transplant one design into the other model and modify the analysis. With ad-hoc results, on the other hand, comparing results can be very difficult and can lead to confusion or, worse, falsely justify misleading claims. It is entirely possible to come up with entirely contradictory results when using two differing sets of assumptions. When these assumptions are implicit as opposed to being part of a well-formulated model, it can be difficult to track down and understand the fundamental differences, and this can waste a lot of time.

Secondly, real-world phenomena frequently consist of many influencing effects. If we are lucky, we can manage the complicated effects by either ignoring them if their effect is sufficiently minute (e.g., relativistic effects can be safely ignored in many real-world problems in mechanics), or replacing them with a simpler approximation (e.g., treating many small, independent sources of noise as a single, large, Gaussian-distributed source). As long as we accurately capture the salient effects, our idealized model should yield interesting and practical results when scrutinized with standard analytic tools.

All investigations, principled or not, assume some sort of model. What characterizes a principled, theoretical approach is the explicit and detailed nature of the assumptions and derivations. Thus, even a flawed model that ignores a relevant effect can provide useful results when utilized in a principled manner: since the conclusion (for example, the "proven" security of some design) logically follows from one or more faulty assumptions, attackers will know that it is on these assumptions where they should focus their efforts. A result of the form "Any attack that has property X will fail against system S" is a warm invitation for researchers to look for attacks against system S that do not have property X.

In short, while a theoretical approach does not directly solve the problems of engineering scientific investigation, it can certainly serve as a guide and save time in recovering from errors, as well as allowing researchers to compare and build on each others' work.

1.2.3 Our Contribution

Currently, there is not a strong theoretical foundation for work in the security of embedded systems. The theoretical foundation is being built at this time, and this thesis represents some of the early thought in this direction.

We make a case for a rigorous, theoretical approach to the design of secure embedded systems, and we give examples of a principled approach through a couple of results in the area.

Chapter 2

Background

In this section, we discuss in detail the types of attacks that apply to embedded systems and attempt to categorize these attacks by identifying key distinguishing features. After discussing the details and their important features, we turn our attention to capturing the essence of these attacks in theoretical models.

In order to classify attacks, we consider two things: how the attack obtains information (i.e., what is the *channel*) and the utility of the information to the adversary.

2.1 Utility of Information

By carrying out an attack against a system, the adversary almost never attempts to learn everything about the computation. Frequently, a single iteration of an attack yields a bit or less of useful extra information about any secret internal state: the utility of this information is restricted. Here we seek to qualify the types of restrictions on the utility of information to an adversary. We focus on three limiting factors:

scope The scope of the information can be limited to the state of a some small subset of the internals of the device, maybe even just one wire or gate.

resolution The information obtained may be a confused amalgam of the states of a variety of sources in the device; greater resolution implies less uncertainty as to contributions of the various components.

noise Measurements made by the adversary may contain a noise component as well as a component that is correlated with the internal state of the attacked device; the greater the noise, the less certainty the adversary has as to the internal state.

These classifications are not mutually exclusive. For example, low resolution means that many parts of a device are contributing to the information, and the component contributed by parts that act independently of the desired secret can be considered noise.

In the sequel, we use the nature of the restricted utility of gleaned information will help us categorize attacks.

2.2 Taxonomy of Attacks

We divide the types of attacks into two broad categories: *passive* and *active* attacks. We define passive attacks as those that do not noticeably interfere with the operation of the device; the attacker gains some information about the device's operation, but the device behaves exactly as if no attacker were present. In active attacks, on the other hand, the adversary exerts some influence on the behavior of the device. While the actively attacked device may or may not be able to detect such influence, an outside observer would notice a difference in the operation of the device.

It is important to note that the distinction between active and passive attacks has more to do with the intrinsic nature of the attack than the intrusiveness of any physical implementation of the attack. For instance, a passive probing attack may involve destructive removal of the packaging before placing a probe directly on a copper trace in the device's innards, while the much less intrusive approach of a power glitching attack, which might involve nothing more than failing to provide sufficient power supply voltage, is classified as an active attack.

2.3 Passive Attacks

As we mentioned above, passive attacks yield information for the adversary without affecting the operation of the attacked device. For this reason, the task of classifying passive attacks is essentially the same as that of classifying the *channel*—the source of side-information—that enables the attack. Known passive attacks utilize channels that tend to fall roughly into three categories, though their distinction is not as clear as that between passive and active attacks. Here we will refer to the three categories of attacks (and the respective channels) as *probing*, *aggregate*, and *timing*.

2.3.1 Probing

Probing attacks are typified by the the placement of a probe directly on a trace in a device in order to learn the logic value of a wire; these attacks characteristically yield information with narrow scope, high resolution, and very little noise.

Although the scope is limited, learning the value of even a single wire for each of a relatively small number of sample computations is frequently sufficient to recover any secret internal state. Unfortunately, probing attacks are very specific to the details of the attacked device and tend to be very intrusive, often requiring specialized equipment and skills to carry out.

2.3.2 Aggregate

Aggregate attacks, on the other hand, characteristically yield information with large scope, low resolution, and varying noise. Power analysis, an approach that involves measuring the instantaneous power consumption of a device by observing current draw from the power supply, is a typical aggregate attack.

In general, an aggregate channel provides a signal that is correlated with some aspect of the operation of several components of the target system. A simple example that is often used in theoretical models of power analysis is where the aggregate channel is the total hamming weight of the logic values on all the wires in a system at a particular instant in time. Another example is a slightly more complicated but perhaps a more realistic representation of power consumption in the pervasive CMOS logic family, where the aggregate channel is the total number of gate inputs that have changed from one instant to the next. Of course both of the previous examples were *theoretical* models, and they attempt to capture the physical aggregate channel of power consumption: the rate at which the device dissipates energy at a particular instant in time.

In its basic form, power analysis is fairly cheap and requires relatively little specialized skill or equipment. As depicted in Figure 2.1, the attacker typically places a small R Ohm resistor in series with the device's power supply and uses a digital scope to measure the instantaneous voltage drop

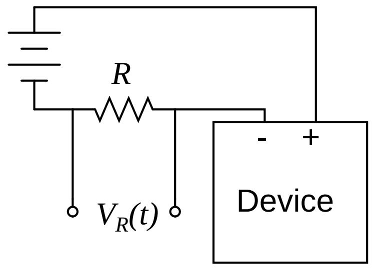

Figure 2.1: A typical configuration for performing power analysis. The device's instantaneous power consumption can be determined by measuring the voltage drop $V_R(t)$ across a small introduced resistor R.

$V_R(t)$ across the resistor many times during the computation. This voltage drop is proportional to the current draw by Ohm's law: $I(t) = V_R(t)/R$. The current, in turn, is proportional (by a factor of the total voltage V_T applied to the device) to the instantaneous power used by the device: $P(t) = V_T \cdot I(t)$. The main complications for the attacker are measuring with sufficient precision to detect the small component of total power consumption that is correlated with the secret and sampling at a high enough rate to isolate this component sufficiently from other signals present at around the same time.

Other aggregate channels exist, as well. Just as electrical switching components in the device cause differences in the current draw that can be detected as a voltage across an introduced resistor, the same switching components can cause disturbances in the electric and magnetic fields *around* the device that can induce voltage in a nearby inductor. The attacker may sample the voltage change on this inductor and acquire an electromagnetic radiation trace of the computation analogous to the power trace discussed above. This electromagnetic channel affords a signal to noise ratio that may be somewhat weaker than direct power analysis, but it has the advantage of requiring no physical interference with the device. Additionally, the attacker can choose the physical position of the inductor relative to the device, possibly amplifying the signal from the region in the device that handles the secret; in this way, the attacker may be able to tune the scope of the electromagnetic radiation channel.

As a final surprising example of an aggregate channel, consider an indicator LED on the outside of the device (e.g., a "power" indicator) that is unintentionally coupled with switching components that handle secret data. Perhaps the LED picks up power transients from a nearby data bus, and the intensity of the LED varies with the data present on the bus. These fluctuations are probably small, but even indirect reflections of the emitted light can be detected through a nearby window over some considerable distance.

In comparison with probing attacks, aggregate attacks generally yield less useful information to the adversary. Due to the typically high noise levels, these attacks also tend to be statistical in nature, often requiring large numbers of samples. On the other hand, aggregate attacks can be entirely non-intrusive, require very little knowledge of the attacked device, and can be relatively cheap to implement.

Of particular interest in considering passive aggregate attacks is the technique of *Differential Power Analysis (DPA)* [10]. DPA is a statistical approach that permits devastating attacks with minimal requirements on the attacker. Perhaps most significantly, an attacker employing DPA does not need to know much of anything about the implementation of the device beyond the choice of algorithm. We discuss DPA in more depth and propose some variants in Section 6.

2.3.3 Timing

Though certainly passive in nature, timing attacks differ markedly from both aggregate and probing attacks; aggregate and probing attacks can be seen as ends of a spectrum of attacks that attempt to determine interesting bits of internal state that are intimately related to some discrete components of the device at some discrete stage of the computation. In contrast, timing attacks, which obtain their information by measuring the time used by the device to compute with the secret data, depend on higher-level behavior of the device. This fact means that timing attacks can often be employed outside the realm of embedded devices; as long as the time between input and output can be measured with sufficient accuracy, timing attacks can work against remote devices [13].

Embedded devices are particularly vulnerable to timing attacks, however, since physical proximity usually permits far more accurate measurement of time than remote access, and augmenting timing information with other side-channels such as power consumption can provide more landmarks in the computation between which to measure time other than *start* and *stop*.

If we consider the restrictions on utility of information in the time domain, we can say that timing attacks typically yield information with varying scope, low resolution, and high noise.

2.4 Active Attacks

In active attacks, the adversary interferes with the normal operation of the device in hopes that the resulting anomalous behavior will yield information about the secret internal state. The amount of information an attacker can hope to to recover depends on the nature of the interference and the nature of the device; in many realistic threat models, the full spectrum of vulnerability is possible, from devices that are totally secure to those that will yield the entire secret.

2.4.1 Glitching

Probably the simplest means for an attacker to introduce error is by altering the device's supplied voltage above or below the range required for normal operation at some cleverly chosen time during the computation. This technique is known as *power glitching*. In some attacks, carefully timed power-glitching can be employed to cause a processor to skip (i.e., treat as NOP) a specific security-critical instruction.

It is not difficult to come up with some realistic scenarios in which a simple well-timed NOP-inducing power glitch can cause the device to divulge whole sections of the key. Alternatively, a power glitch can be a simple way to induce a general fault in the computation—this can suffice to reveal the whole key[4].

2.4.2 Targeted Faults

In another type of active attack, sometimes called *zapping*, the adversary induces faults in particular gates on or particular wires. In Chapter 4, we discuss *optical fault induction* as a possible mechanism for targeted errors. These types of attacks can be quite difficult and expensive to defend against. We elaborate more on targeted faults and investigate dual-rail encoding as a possible countermeasure in Chapter 5.

Chapter 3

Models

3.1 Philosophy of Selecting Models

Several decisions need to be made when selecting a model for the system to be analyzed. The decisions made have an enormous effect on the outcome and credibility of the resulting analysis. A disingenuous designer can "prove" just about any design secure by using an accommodating model. On the other hand, a prudent designer can build in resilience against unanticipated attacks that arise far in the future after the design process by selecting a model that captures a sufficiently broad range of attack classes.

There are no hard and fast rules, but we can provide some general direction to help make those decisions in such a way as to select a model that is useful and productive, and not irrelevant or misleading.

3.1.1 A Conservative Approach

In approaching the issue of security in embedded systems from a designer's point of view, we want to make conservative decisions; that is, we will prefer to overestimate potential adversaries rather than to underestimate them. Failure to follow the principle of conservative design, at least as much as possible, can completely eliminate the benefits of a theoretical approach, and the resulting analysis will be a dangerous farce.

Our motivation for employing a theoretical approach was to take advantage of the benefits of rigor, and in keeping with that end, we suggest making the most conservative assumptions (i.e., giving the benefit to the adversary) when possible, and explicitly and completely quantifying the deviations from this principle (i.e., imposing limitations on the adversary) when necessary.

Indulge in a metaphor. Attackers wish to climb the fortifications, and in order to do so, they need a ledge of sufficient width to stand on. Designers of the fortifications wish to eliminate the ledges or at least make them as narrow as possible. Well, eliminate the ridges, or make them narrow? Obviously the former is preferable, but it may be difficult, expensive, or impossible. Forced to settle for the latter choice, then, some designer must finally decide that the ledge is narrow enough: "they cannot stand on this, they are clumsy and their feet are too fat." This is the metaphorical equivalent of hand-waving, and while this type of argument is sometimes necessary, it should come with the strong and explicit caveat that it is a critical link in the security of the design. "Our fortification baffles clumsy attackers with fat feet; if we anticipate any other types of attackers, we had better have another plan."

3.1.2 Theory in Practice

> *The difference between theory and practice in theory is much less than the difference between theory and practice in practice.*
>
> - Randal L. Schwartz

It is not always possible to perfectly defend a system. Sometimes it is necessary to settle for some level of security that is deemed sufficient, where the cost of overcoming that security is expected to be out of reach for all anticipated adversaries for the foreseeable future. Designers that find themselves playing such games need to expect their designs to be broken in that period just after the foreseeable future. The effective lifetime of security for a given design is proportional to the effort and resources expended on that security.

Leverage

Here a major apparent difference between classical and embedded security stands out. Most cryptographic primitives (e.g., block ciphers and public key systems) afford the designer with exponential leverage: adding just one bit of key can potentially double the difficulty for the adversary. Embedded security techniques do not all seem to provide such leverage, and the costs of increasing the level of security in a device can be quite large: consider doubling or tripling the number of gates used in a design in order to prevent fault attacks or balance power consumption.

Designers of embedded systems seem to be at a relative disadvantage, then, and should expect an effective security lifetime much less than what is assumed for conventional cryptographic approaches (e.g., eighteen months for embedded versus thirty years for a conventional design that enjoys private computing guarantees). Reverse engineering and analyzing a deployed system does take some time, but it is nothing like what is required in most known attacks of cryptographic primitives.

3.2 Our Threat Models

In the following sections we will develop some models for security in embedded computation that we will later use to derive some particular results in Chapter 5 and Chapter 6.

We attempt to incorporate the philosophy outlined in the previous discussion, commenting on the relevant decisions as we make them. We also attempt to highlight the major drawbacks of each of the models we present and discuss why they would be inappropriate in other applications.

3.2.1 Considerations

A primary consideration in selecting a model is its intended use. Our work attacks several security designs (dual-rail redundant representation and masking), so we pick models that capture just the relevant aspects of those security designs. The applications here are attacks; they are not necessarily intended to be practical, but rather to test the assumptions behind the designs.

Models chosen to prove countermeasures secure, on the other hand, would have to be a bit more robust to be credible. More specific discussion along these lines follows the description of each of our models.

3.2.2 Common Assumptions

The assumptions that we appropriate in this section will be common to any derived models, so we attempt to make them as lightweight and unimposing as possible; their purpose is more to unify and simplify the presentation rather than affect the results.

Target System

The target system is assumed to compute some function $F : I \times K \rightarrow O$, where I is the set of possible inputs (supplied by the user, or in the case of attacks, by the adversary), K is the set of secret keys (supplied by the designer), and O is the set of outputs. The key is assumed to be hard-coded into the device; although we frequently assume that the attacker knows everything about the implementation of the target system, we do not mean to say that he knows the specific value of the hard-coded key.

Without loss of generality, we may refer to the computation of the function F as *encryption*, and we can alternatively refer to the *key* or to the *secret* to refer to the bits that the adversary is after.

Adversary's Behavior

The adversary's goal is to extract from the device either the key or an equivalent key. The adversary may always use the device as a *black-box*, relying solely on its designed input and output behavior; this adversary would be operating in a more classical model.

In considering the security of embedded systems, however, the adversary may also be able to somehow break the private computing assumption of the device. The usefulness of this extra influence can be measured by how much less work the adversary needs to do when compared to an adversary that uses the device in a purely black-box manner. The usual measure of the cost of an attack is the number of interactions with the device (compare to black-box queries) that the adversary needs. If exploitation of the extra influence or side-channel imposes some additional cost, this will be indicated in the model.

3.3 Active Attack Models

Obviously, the most important feature of a an active attack model is its ability to capture the nature of the adversary's influence on the target device. As always, it is also desirable for models to keep all of the other aspects as simple as possible. What we seek, then, is the simplest computational model with respect to which we can define a "fault" in the context of the active attack or class of active attacks in consideration.

Later, in Chapter 5, we will examine the use of dual-rail encoding as a countermeasure for active attacks. In order to carry out that analysis, we develop here a simple but conservative (i.e., it does not underestimate adversaries) attack model, which specifies both a computation model and defines the induced "faults" with respect to this model.

3.3.1 An Example Fault Induction Model

In our analysis of dual-rail encoding in Chapter 5, we are concerned with the behavior of systems when subjected to single arbitrary faults. In practice, these faults are in induced in the transistors at the physical level, so we model them as deviations in the logic level at the inputs or outputs of logic gates. That is, the value at the particular gate input (or output) under attack differs from what it would be if it were not under attack. Note that the resulting faulty behavior of the device under attack can immediately leak the key [4], and it is not always necessary for a fault-induction attack to compare a faulty and non-faulty run (i.e., one interaction can be enough).

Model of Computation

In studying the effects of single faults on the target system, we find it useful to think of the target system as a network of logic gates: AND, OR, NOT, and C-elements.[1] There does not need to be a one-to-one correspondence between logic gates in the model and gates in the implementation. For example, it may be helpful to unroll sequential logic into a combinatorial (loop-free) network of logic gates. In the analysis of such a network, the adversary's ability to affect a particular gate in the unrolled network corresponds to his ability to affect the corresponding implementation gate at a precise time in the computation.

In keeping with our conservative approach, we assume that the timing in these circuits is deterministic and that the adversary has (almost) complete knowledge of every aspect of the implementation of the circuit—the obvious exception being the exact values in the key-storage gates.

Computation occurs in a straightforward manner: input is provided to the network, intermediate values are computed in the gates, and these values are propagated to subsequent gates. We do not concern ourselves with timing and synchronization issues. The initial states of the wires in the network may be specified by the design; for example, dual-rail designs call for all wires to be pulled to logic zero before a round of computation.

Fault Model

We think of an induced fault in a circuit as a deviation in the value at the output or input of a single logic gate.[2]

We allow exactly one fault per computation and we assume the adversary has complete control over the absolute timing of the fault and the location (i.e., which gate) of the fault. Since the adversary has complete knowledge of the device layout and timing, he is potentially capable of devastatingly effective attacks.

What is left to determine is the exact nature of the induced faults. Our classification of faults is partially motivated by a similar one used by Blömer and Seifert in their fault-attack analysis of AES [7]. We classify faults using two parameters: the *type* of fault and the *persistence* of the fault.

Fault Type The type of fault indicates what kind of deviation the fault induces at its target. We consider three types:

- A *set*-fault forces its target to carry a 1 value.
- A *reset*-fault forces its target to carry a 0 value.
- A *flip*-fault inverts the value carried by the target.

[1] The C-element gate is explained in Section 5.2.

[2] Except where fanout is concerned, this is similar to fault models where there is a possible change in the value carried on a single wire. Physical considerations suggest deviations at inputs and outputs of gates as the most reasonable fault model, at least with CMOS. See Chapter 4 for details.

Fault Persistence The persistence of a fault models the duration of the effect of a fault as well as the level of control the attacker has over when the effect occurs. We consider two levels of fault persistence:

- A *transient* fault affects its target in an erratic manner over a stretch of time, sometimes causing it to deviate to the fault state and sometimes back to its natural state. Our only assumption is that the faulty state occurs at least once during the time interval in which the attacker applies a transient fault.

- A *steady* fault forces the target to assume the fault state for the entire interval during which the attacker applies the steady fault.

In our discussion in Chapter 4, we will show how some of these types of faults correspond to proven practical threats.

Shortcomings

The typical objection to this model is that it makes strong assumptions about adversary's control over the location and timing of induced faults. It is certainly true that these are strong assumptions, but to claim that this somehow invalidates the model is missing the point; security in this model implies security against less capable (and more realistic) adversaries. The onus is on system designers to either meet this high bar or make explicit their claims of security against fault induction (i.e., propose an alternative model).

That said, it would certainly be desirable to parameterize this model in some way to capture probabilistic faults, locality, and uncertain timing. Such a weakened model would then be useful not only for testing designs, but also for analyzing realistic attacks.

3.4 Passive Attack Models

As mentioned in Section 2.3, there is a varied spectrum of passive attacks, and the primary differentiating characteristics are locality and resolution of the determined information. At the ends of the spectrum are probing attacks, with very precise locality and resolution, and aggregate attacks, which typically obtain very non-local, conglomerate information about the computation.

Analysis of a probing attack would require a model that captures details of the implementation at the gate-level, as in our active attack model from Section 3.3; a good example of such a model for analyzing probing is given by Ishai, Sahai, and Wagner in a proposal for securing systems against such attacks [15].

To analyze an aggregate attack, on the other hand, it suffices to specify a few characteristics of the implementation of the circuit without necessarily specifying the details at the gate level; one must identify the channel (e.g., power consumption), determine the aspects of the system implementation that affect this channel (e.g., representation of 1 versus 0 bit values), and determine the nature of the uncertainty in the adversary's view of this channel (e.g., Gaussian noise due to imprecise measurement and contributions to power consumption by unrelated parts of the circuit).

Later, in Chapter 6, we will analyze some preprocessing techniques that attempt to overcome masking techniques that are commonly proposed as countermeasures for power analysis. In order to do that, we develop here a model for passive power analysis attacks.

3.4.1 An Example Power Consumption Model

Power analysis aims to detect the often minute influences of interesting data on the current drawn by the device while computing with that data. What must be specified is how the data in the device affects that consumption.

The Hamming model gives a simple approach to this question: at any given time, each bit represented in the circuit (present at a gate input or output, say) contributes an amount $\pm\delta$ to the power consumption depending on its value, $+\delta$ for 0 and $-\delta$ for 1.[3] It is also helpful to assume that the contribution of a particular bit always occurs at the same predictable time in each computation.[4]

Notation

Thus, if we let $X(t)$ be the set of bit values represented in the device at time t, we can define the *power trace* $T(t)$:

$$T(t) = \delta \sum_{b \in X(t)} (-1)^b. \tag{3.1}$$

Thus each trace containing m samples is a function of time with a discrete range $\{1, \ldots, m\}$. Power analysis attacks frequently rely on statistical techniques where many traces are acquired and analyzed. Different traces are denoted with subscripts or superscripts. For example, in an attack that takes n traces, trace i can be denoted T_i, and the average power consumption at time s is $\frac{1}{n}\sum_{i=1}^{n} T_i(s)$.

Various attacks may further group the traces, and these grouping may be denoted by a more complicated subscript or superscript scheme (see Chapter 6, for example).

Signal and Noise

Of course only a few bits are of security interest; we call their contributions to the instantaneous power consumption the *signal*. Uninteresting independent bits and measurement noise also make their contributions to the power traces, and it is called *noise*. We assume that this noise is distributed as a Gaussian, and it often helps to assume that the traces have been scaled and offset so that the noise components are normally distributed (i.e., $T(s) \sim \mathcal{N}(0, 1)$ when there is only noise at time s), and that the signal magnitude, δ, is in units of the standard deviation of the noise. It is often helpful to name the noise components, so we call them $S()$:

$$\begin{aligned} T(t) &= (\text{signal at time } t) + (\text{noise at time } t) \\ &= (\text{signal at time } t) + S(t). \end{aligned}$$

As a final example, consider a trace T consisting of m samples from a system where there is only one bit of interest, with value b, that contributes to the power consumption only at time s. We have:

[3] This assignment of $\pm\delta$ to $\{0, 1\}$ permits the convenient notation for the contribution of one bit with value b: $\delta \cdot (-1)^b$.

[4] Countermeasures such as random timing and randomized order of evaluation have been proposed for use against power analysis, but can be overcome by statistical or signal-processing techniques, and are largely ineffective.

$$T(t) = \begin{cases} S(t) & \text{if } t \neq s \\ S(t) + \delta \cdot (-1)^b & \text{if } t = s \end{cases}$$

so that $T(t) \sim \mathcal{N}(0,1)$ for $t \neq s$ and $T(s) \sim \mathcal{N}(\delta \cdot (-1)^b, 1)$.

The Adversary

While we assume that the adversary knows the algorithm and can predict the timing in the device, it is not always necessary that he knows exactly when the signal occurs; as we will see in Chapter 6, Differential Power Analysis provides an elegant solution to this problem.

Shortcomings

Models, such as the Hamming Model, that consider the instantaneous power consumption to be a function only of the current state (i.e., independent of previous states), are relatively simple. Unfortunately, they are not very accurate representations of power consumption in real-world implementations, which are usually based on CMOS digital logic. In CMOS, almost all of the current (and therefore power) is consumed during state transitions. An accurate model for CMOS applications would take this into account.

The significance of whether or not a model accurately reflects the behavior of the system with respect to the side-channel in question depends on whether it is being examined from the perspective of an attacker or a designer. For example, in Chapter 6, we use our Hamming-type model in analyzing higher order power analysis methods; in this case, it does not matter whether the power consumption is proportional to the Hamming weight or the number of transitions, only that there is some data dependence in the power consumption. On the other hand, a designer may propose a system where the number of 0s and 1s is maintained equal in any valid intermediate state of a computation. This could be argued secure in a Hamming model, but utterly fail to prevent attacks in real-world CMOS implementations unless it also took transitions into account. The designer would clearly benefit from a model that captured all relevant real-world effects.

This model says nothing about localized information, so it does not apply to passive attacks from other parts of the passive spectrum, such as probing or EM analysis. There are no practical reasons, however, that these attacks could not be combined simultaneously with power analysis. Our model assumes that the adversary attempts only power analysis without attempting to coordinate it with any other techniques, and there is no practical justification for this. Again, this is not a problem in using the model for an attack, but it does make this model, as it is, a dubious choice for use in arguing the real-world security of a system.

Finally, this model completely ignores timing information. Changing the requirement of predictable timing in this model can present to the adversary either a help (e.g., data dependent timing leads to timing attacks) or a hindrance (e.g., randomized timing requires data alignment before statistical methods can be employed.)

Chapter 4

Physical Motivation for Attack Models

4.1 Optical Faults

The powerful yet surprisingly practical optical fault injection attack [1] provides the major motivation for our threat models. While previous work will convince the reader that fault injection is definitely a problem worth considering in the design of secure embedded systems, we wish to go a little further into the physics behind optical attacks.

In exploring a variety of fault types, we try to cover a sufficiently large spectrum of potential fault injection methods against a variety of digital logic families. In this section, we attempt to give some of the background required to understand how optical fault injection works and, as an example, to justify that at least one of our threat models captures the effect of optical faults against CMOS digital logic.

4.1.1 MOSFET Basics

Enhancement-mode MOSFETs (Metal Oxide Semiconductor Field Effect Transistors) are the basic components of CMOS (Complementary MOSFET) digital logic. Ideally, a MOSFET behaves like a voltage-controlled switch: the electrical potential difference between the the electrically-isolated metal[1] gate and the bulk determines whether or not the device will allow charge to flow between the source and drain.

Enhancement-mode MOSFETs come in two basic varieties, n-channel and p-channel, depending on the type of majority charge carrier (negative or positive) that is active when the device is conducting. The two types behave in a complementary manner: n-channel enhancement-mode MOSFETs conduct when the gate voltage is positive relative to the bulk, while the p-channel variety conducts when the gate voltage is negative relative to the bulk. Figure 4.1 gives commonly-used schematic representations of MOSFETs.

In each type of MOSFET, the source terminal is connected to the supply of charge carriers: n-channel devices have their source connected to the more negative input and p-channel devices have their source connected to the positive input.

[1]Modern devices frequently have a non-metal polysilicon gate, and are alternatively referred to as IGFETs (Isolated Gate Field Effect Transistors).

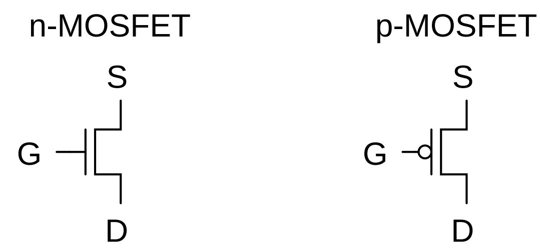

Figure 4.1: Common schematic representations of MOSFETs. The terminals are labelled: G is Gate, S is Source, and D is Drain. The inverter dot on the gate of the p-channel MOSFET suggests the p-channel behavior: it conducts when the gate voltage is low.

Mechanism

The behavior of semiconductors is a product of the interplay between the two types of semiconductor material: n-type and p-type. These types of semiconductor material are created by *doping* a relatively pure non-conductive crystal (e.g., a crystal of silicon or germanium atoms) with different atoms (e.g., phosphorus, arsenic, aluminum, or gallium). The doped material has a surplus of charge carriers: n-type material has an abundance of electrons that are not bound in its crystal structure while p-type material has a dearth of electrons—an abundance of *holes* in its crystal structure. Each type of doped material will conduct charge.

The most interesting and useful behavior occurs when the different types of material are joined in various configurations. The basic effect utilized by semiconducting devices is that, under normal conditions, current will flow across a *pn junction* (a junction between p-type and n-type materials) only when the n-type material is sufficiently negatively charged relative to the p-material. For example, a diode, which permits current flow in only one direction, is nothing more than a pn junction.

MOSFET Configuration

An enhancement-mode MOSFET has one type of semiconductor material at each of the source and drain terminals, and a bulk (or substrate) of the other type of material separating these regions. For example, an n-channel MOSFET has n-type material at the source and drain inputs and a bulk composed of p-type material. See Figure 4.2 for a depiction of an n-channel enhancement-mode MOSFET.

This configuration would normally not allow current to flow between the source and drain. To force the device in Figure 4.2 to conduct, a positive charge is loaded onto the gate. Although the gate is electrically isolated from the bulk by a layer of non-conducting oxide, the charge on the gate induces an electric field that permeates the the p-type material below. This electric charge attracts electrons (negative charge carriers) to the region in the p-type material near the gate. If enough charge is present on the gate to induce a sufficiently strong electric field, so many electrons will be present in the affected region of the p-type material to make it behave as if it were n-type material. Thus a *channel* is created between the n-type regions at the source and drain and current can flow: electrons are the majority charge carrier.

The situation is similar in a p-channel MOSFET, except the type of materials and charge carriers are reversed. When a negative charge is present on the gate of a p-channel device, electrons in the underlying n-type material are repulsed, resulting in an abundance of holes—the affected region behaves like p-type material and a channel is formed.

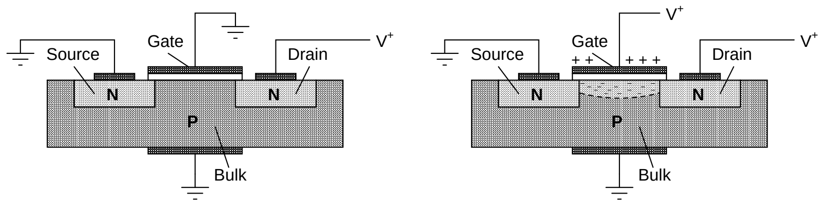

Figure 4.2: An n-channel enhancement-mode MOSFET in normal operation.

4.1.2 Optically Induced Failure in n-Channel MOSFETs

The intended method for causing n-channel enhancement-mode MOSFET to conduct is by moving a positive charge to the gate, resulting in an electric field that attracts free electrons to the underlying p-type material. However, the intended method is not the only way to create a channel of majority charge carrier electrons in the p-type material.

Electrons and Photons

When electrons move through a crystal lattice, they pop in and out of the covalent bonds that bind the atoms in the crystal structure. An electron that is part of a bond has lower potential energy than a free electron, so a free electron that joins a covalent bond *loses* some energy. Where does this energy go? It is released as a *photon*, a unit or *quantum* of electromagnetic radiation. The wavelength of the emitted photon is determined by the amount of energy lost by the electron: the more energy lost, the shorter the wavelength. This phenomenon, known as the *photoelectric effect*, is utilized in LEDs (Light Emitting Diodes), where the semiconductor materials are specifically chosen so that the emitted photons have a desired wave-length of light (e.g., visible green, infrared, etc.)

What is important for us, however, is the opposite effect: if an electron in a covalent bond of a crystal structure absorbs a photon with sufficient energy, it will jump out of the bond and briefly become a free electron. When illuminated by a sufficiently intense source of light (e.g., a laser), even p-type material will have a large number of free electrons due to photon absorption. Illuminating an n-channel MOSFET with light of the appropriate frequency and intensity to penetrate the gate and oxide insulator can induce a channel of electron majority charge carriers between the source and drain, causing the device to conduct regardless of the charge on the gate.

Thus, a clever adversary may use a well-aimed laser with the proper intensity to affect the switching behavior of n-channel MOSFETs in the target device.

4.1.3 Example: an Optically Induced Fault in CMOS

How does the injection of faults in the transistors affect the digital logic gates realized by these transistors? We consider the case of CMOS, one of the most commonly used digital logic families. CMOS uses electric potential on to indicate logic values. Typically, positive voltage represents a logical 1 and 0V represents a logical 0.

Skorobogatov and Anderson [1] have successfully used light to set the value of bits in SRAM[2]. For our example, however, we would like to use the same technique to influence the output of the circuit depicted in Figure 4.3.

[2]memory based on CMOS digital logic

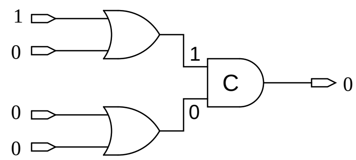

Figure 4.3: The logical schematic of our target circuit.

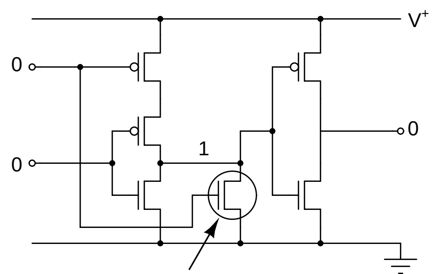

Figure 4.4: The CMOS schematic of the target OR gate. The n-channel MOSFET indicated by the arrow and circle is the target of our optical attack.

In Figure 4.3, only one of the inputs to the C-element is 1, so the C-element has a 0 output. Suppose we wish to change that output to a steady 1. All that we really need to do is cause the lower OR gate driving the second input of the C-element to go from 0 to 1, even if only for a short time (once the C-element has a 1 output, it will not drop back to a 0 output until *both* of its inputs drop to 0). With that in mind, we concentrate our efforts on the lower OR gate in Figure 4.3.

Figure 4.4 gives the transistor-level schematic for the target OR gate. The left four transistors constitute a CMOS NOR gate, while the two rightmost transistors form a CMOS inverter. The target transistor, the n-channel MOSFET indicated in Figure 4.4, is the last transistor of the NOR half of the circuit. If it were to conduct, it would cause the output voltage of the NOR half to drop sufficiently to cause the output of the inverter half (and hence the whole OR gate) to go high.

By illuminating this transistor with a flash of light of the appropriate frequency and intensity, we can cause it to briefly drop the voltage of the line going into the inverter half of the circuit, thus creating a brief high-voltage spike from the output of the OR gate to the second input of the C-element (see Figure 4.3), ultimately causing the target circuit to output 1.

4.1.4 Practical Considerations

In order to optically flip bits of SRAM, Skorobogatov and Anderson [1] had to depackage (expose) the digital logic in the target device, an older MicroChip PIC16F84, sufficiently for their light to be able to affect the transistors. It would probably be significantly more difficult to accomplish the same thing in the logic (rather than RAM) part of a more modern (especially multilayered) design. Furthermore, in a dual-rail design that incorporated phototransistor tripwires as suggested by Skorobogatov and Anderson [1], depackaging could cause the circuit to always halt in the alarm state unless the adversary is able to circumvent this countermeasure.

The example in the previous section demonstrates something close to our *transient clear fault model* that we introduced in Chapter 3. It is not obvious whether the other fault models introduced in that chapter are applicable to CMOS. In following a principled, theoretical approach, however, it would behoove a designer to consider all of our fault-induction models; a sysem that is robust against all would be secure against single-faults not only as a stadard CMOS implementation, but likely also in unanticipated implementations.

Chapter 5

Fault Attacks on Dual-Rail Encoded Systems

5.1 Introduction

Securing embedded systems is exceedingly difficult due to the fact that potential adversaries have physical access. In this chapter, we focus on fault-injection attacks. Recently, dual-rail logic, a scheme previously used in asynchronous circuit designs, has been suggested as a countermeasure for fault-injection attacks. In this article, we analyze, in a variety of threat models, the effectiveness of dual-rail encoding as a countermeasure against single-fault injection attacks.

5.1.1 Summary of Results

In Section 5.4, we describe a series of attacks against dual-rail encoded systems in the presence of adversaries capable of inducing a variety of types of faults. Each of the attacks attempts to recover an n-bit secret key from the dual-rail encoded device. We measure the efficiency of our attacks by the number of trials required to recover the entire key. Table 5.1 summarizes the effectiveness of our attacks in the various threat models.

As these attacks are so effective, we conclude that designs for secure embedded systems must take these threat models into account. The system should either be designed to be robust in our threat models or some other explicit countermeasures should be incorporated for the purpose of invalidating our models.

Table 5.1: Summary of our attacks for extracting an n-bit key from a dual-rail encoded system.

Fault Type	Fault Persistence	Required Trials	
		Expected	Worst-Case
Set	Steady	n trials	n trials
Reset	Steady	$1.125n$ trials	$1.5n$ trials
Flip	Steady	n trials	n trials
Set	Transient	n trials	n trials
Reset	Transient	*(no attack)*	*(no attack)*
Flip	Transient	n trials	n trials

Table 5.2: Logical values (0,1), metadata values (quiet ,alarm), and their corresponding dual-rail encodings.

Logical or Metadata Value	Dual-rail Encoding
0	(0,1)
1	(1,0)
quiet	(0,0)
alarm	(1,1)

5.1.2 Overview

In Section 5.3 and Section 5.4, we construct the framework in which we analyze the dual-rail encoded systems and present our attacks against these systems in that framework.

We suggest some directions for future research into securing embedded systems against fault-injection attacks in Section 5.5 and conclude with Section 5.6.

5.2 Background

5.2.1 Dual-Rail Encoding

Dual-rail encoding is an alternate method for encoding bits in hardware. In contrast with classical encoding, where each wire carries a single bit-value, dual-rail encoded circuits use two wires to carry each bit. Table 5.2 summarizes the encoding.

Notation

Throughout this text, we will use lowercase variables such as x and y to denote logical values and subscripted lowercase variables such as x_1, x_0, y_1, and y_0 to denote the values on the dual-rail lines. For example, when $x = 1$, its dual-rail encoding is $(x_1, x_0) = (1, 0)$.

Asynchronous Circuits

Dual-rail encoding seems unnecessarily more complicated and expensive than classical encoding, but dual-rail circuits have the advantage of being able to carry the metadata (in particular the quiet state) necessary to realize *asynchronous* circuits. In classical circuits, there is no way to tell when a wire is carrying valid data, so components have to be coordinated by a common clock: components have an additional “clock” input that indicates when valid data is present on the inputs. In dual-rail circuits, on the other hand, it is possible to tell when inputs contain valid data—when they are no longer in a quiet (0,0) state—and so the components synchronize automatically.

Asynchronous circuits are attractive since they have some inherent robustness against side-channel and fault attacks. For this reason they have been suggested for use in secure embedded systems [2, 3].

Generic Transformation

We can think of dual-rail logic as a generic transformation that converts a high-level logical description of a circuit in terms of AND, OR, and NOT gates into a dual-rail encoded circuit where

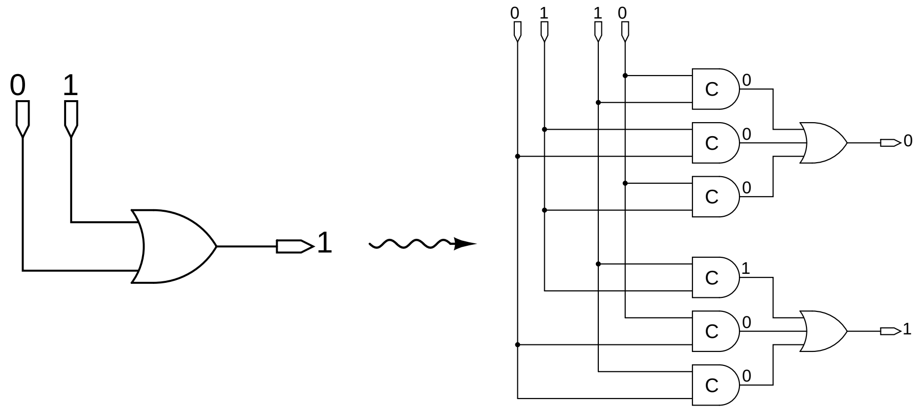

Figure 5.1: A logical OR gate and its dual-rail implementation. While this is not the simplest way to implement OR in dual-rail, this design suppresses data-dependent power consumption.

each high-level logic gate is realized by a collection of OR gates and C-elements (explained later in this section) that compute the appropriate logic function in dual-rail encoding. Figure 5.1 gives an example transformation of a logical OR gate into its dual-rail implementation.

The dual-rail implementation of a logical AND gate is similar: one is depicted in Figure 5.2. While the logical AND and OR gates have dual-rail implementations that require several C-elements and OR gates, it is easy to verify that a logical NOT can be performed on dual-rail encoded bits by simply crossing the wires: in our notation, if x is encoded as (x_1, x_0), then $\overline{x}$ is (x_0, x_1).

It is instructive to consider the similarities between the dual-rail implementations of the logical AND and OR gates in Figure 5.2. In particular, note that one can easily be obtained from the other by applying DeMorgan's laws and crossing the dual-rail wires to implement the NOTs.

C-Elements

The gates that look like AND gates with a letter 'C' in the middle are *C-elements*. Unlike typical logic gates, they maintain some state: when their output is 0, they act like AND gates, and when their output is 1, they act like OR gates. C-elements are used in asynchronous circuits to address the problem of unsteady input signals, and we will leverage their stateful behavior in some of our attacks.

Dual-Rail Circuit Operation

In a typical computation in a dual-rail circuit, all wires start in a quiet state. When valid inputs are supplied to the device, the gates start to compute and intermediate wires driven by these gates go from the quiet state to a valid data state. Finally, when all inputs are supplied and the data has had time to propagate, the outputs will be in valid data states.

If something unexpected happens during the computation (such as an attacker introducing a fault), the alarm state might appear in some intermediate value. A secure implementation will propagate this value to all of the outputs in order to prevent a possible attacker from learning some partial results of a faulty computation.

Finally, to prepare the circuit for another computation, the quiet state is again applied to the inputs; this state should propagate through the device and return all internal values back to quiet as well.

Robustness Against Side-channel Attacks

Due to their effectiveness and the relative ease with which they can be mounted, power analysis attacks are a major motivation for the use of dual-rail encoded systems.

The power consumption of a device using CMOS digital logic is mostly a function of the number of transitions in the values on wires and at the inputs and outputs of its gates. A carefully designed dual-rail circuit (with balanced gate design and equalized wire-lengths) can minimize or eliminate the data-dependence of the power consumption during a computation: just as many transitions are required with a dual-rail value (0,1) as with (1,0).

On the other hand, at least one empirical test suggests that electromagnetic profiles of computation in dual-rail circuits may be more highly correlated to data than in clocked circuits. This surprising result is possibly due to the absence of noise from clock lines [3].

Robustness Against Fault Attacks

Dual-rail encoding is a particular instance of *m-of-n encoding*.[1] In m-of-n coding, valid data (as opposed to metadata) codewords have exactly m 1's. Such codes are called immutable: flipping any single bit in a valid data codeword results in invalid data [2].

In the case of dual-rail logic, changing one bit of a valid data encoding (either (0,1) or (1,0)) results in one of the metadata states: quiet (0,0) or alarm (1,1). Thus single faults are detected. The device can halt with an error indication instead of giving the attacker an erroneous output that might leak secret information.

Thus dual-rail encoding provides some security against the general fault glitching attacks mentioned in Section 2.4.1. If faults are not detected and the device outputs an erroneous result, an attacker can quickly (with just two computations and only one injected error) deduce the entire secret. [4] A dual-rail implementation would only give the correct output or an indication that an error occurred—not much for an attacker to work with.

At least that is the hope. An important concern, however, is that errors are *detected* and not *corrected*. As we will show, an attacker can observe whether carefully injected faults result in valid or erroneous computations and thereby deduce some information on internal values (e.g., cryptographic key bits).

5.3 Assumptions

Target Systems

The target systems we consider are the dual-rail implementations of logical circuits consisting of AND, OR, and NOT gates. In particular, they consist of the dual-rail gates depicted in Figure 5.2 along with the simple wire-swap for logical NOTs.

Assumed Behavior of Dual-Rail Encoded Systems

We assume that the dual-rail systems perform a computation on some given input and output the result if no error is detected. In the case of an error, whether it is the result of the presence of an alarm (1,1) state or a deadlock condition (quiet $(0,0)$ after some timeout), the only output is an error indicator. A system that indicated the type of error would only make our attacks easier.

[1] In particular, "dual-rail logic" is another name for 1-of-2 encoding.

While the deadlock and alarm states may be indistinguishable for users who follow the system's intended interface, we assume it is possible to distinguish these states when also monitoring the power consumption of the system during the faulty computation.

We assume that the timing of the computation is deterministic and known to the adversary. While randomized timing may complicate some of our attacks, its effectiveness as a countermeasure is highly implementation-specific. It is also possible that an adversary could use a side-channel such as power consumption or electromagnetic radiation to determine the computation timing in real-time.

The Secret

An attacker would like to extract the key from the device. Typically, a key is hard-coded in the circuitry or stored in program memory. While attacks on the stored key can be quite devastating [5], we prefer a more general approach and make no assumptions about how the key is stored in the device. Our attacks focus instead on the initial computations involving the secret bits.

Finally, we assume that each bit of the secret is used in every computation, as is usual with a block cipher key.

5.4 Attacks on Dual-Rail Encoded Systems

As mentioned in Section 5.3, the goal of the attacker is to deduce an n-bit secret held in the system in question. The attacker observes the external behavior of the system when given known or chosen inputs, with the option of inducing a single fault per computation. We measure of efficiency of an attack by the number of computations required to deduce the secret.

5.4.1 Attacker's Strategy

In all threat models, the attacker attempts to learn a secret bit by deducing an internal value in the dual-rail implementation of the first logical gate that takes that secret bit as an input. In particular, he will induce a fault in a well-chosen OR or C-element gate, and by observing the result of the computation, learn something about the key bit in question. Repeating this process for each of the n secret bits, the attacker will learn the entire secret.

Figure 5.2 shows the encoding-level implementation of both an AND and an OR gate for dual-rail logic. Sites relevant to our attacks are labeled in the figure.

5.4.2 Set and Flip Faults

We show how an attacker who can induce set or flip faults can recover the secret. The trick is to use the C-elements intended to propagate alarm signals to help discover the logical values of the inputs of the dual-rail. Due to the stateful nature of the C-elements, both steady and transient flip-faults and transient set-faults are utilized in the same way as the steady set-faults to recover the secret.

We first describe the attack in the steady set fault threat model, then argue that this attack works in the other threat models.

Steady Set Faults

Suppose the attacker wants to learn bit i of the secret. She will focus her attack on the first gate that takes bit i as an input. In particular, if bit i is the a-input to the gate, she will induce a

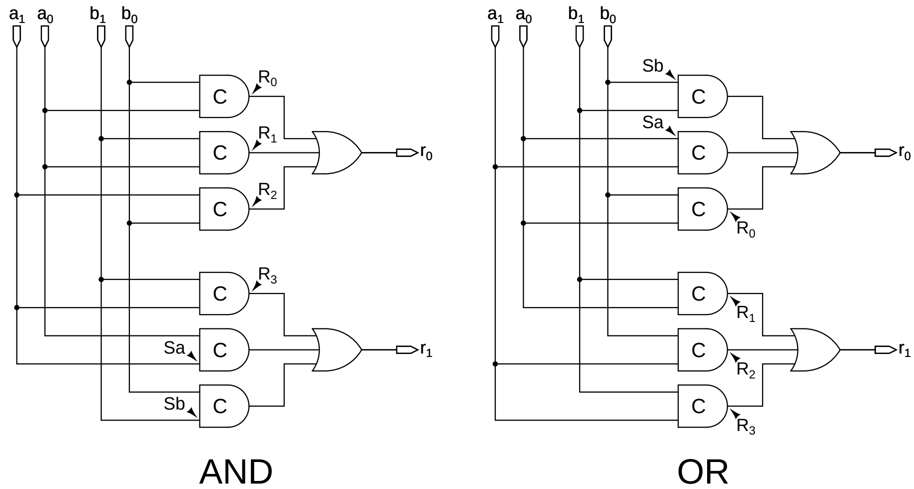

Figure 5.2: Dual-rail AND and OR gates. Set/Flip attack target sites are labeled Sa and Sb. Reset attack target sites are R_0 through R_3.

fault at the C-element input marked Sa in Figure 5.2 for the appropriate type of dual-rail gate. Similarly, if bit i is the b input, she will induce the fault at the site labeled Sb.

Suppose both the a and b inputs contain valid data, that is, each is in either a logical 0 (dual-rail encoded $(0, 1)$) or a logical 1 (dual-rail encoded $(1, 0)$) state.[2] Furthermore, suppose that the attacker is attempting to learn the logical value of the a input to a dual-rail logical AND gate (the b input and OR gate cases are similar). In this case, the suggested attack is to set to 1 the Sa input to the C-element, as indicated in Figure 5.2. Let us consider the resulting output of the dual-rail AND gate. There are two possibilities, according to the two possible logical values of a:

- Case 1: The a input has logical value 1. Then $(a_1, a_0) = (1, 0)$.

  Since a_1 is already 1, setting the Sa input to 1 has no effect, and the entire computation terminates normally with a valid output.

- Case 2: The a input has a logical value 0. Then $(a_1, a_0) = (0, 1)$.

  Since a_0 is 1, the output value r_0 is 1 regardless of the b input's logical value. Furthermore, we have set to 1 the C-element input marked Sa. The other input to that C-element is a_0. Thus, that C-element will have output 1, driving the output $r_1 = 1$. At this point, the output of this dual-rail AND gate is the $(1, 1)$ alarm state, and this state propagates to the rest of the system. Therefore, the computation halts in an error state.

See Table 5.3. Notice that the computation halts with an error if and only if the logical value of a was originally 0. Thus, by observing the computation to see whether it halts, the attacker can learn on key bit—the logical value of a. In short, by inducing exactly one well-chosen set-fault, the attacker can deduce the value of key bit i. Repeating this attack once for each key bit allows the attacker to learn the entire n-bit secret with n set-faults.

[2]Our fault models require that at most one fault happens per computation, and since we have not introduced a fault yet, both inputs will contain valid data, so our assumption is reasonable.

Table 5.3: Summary of the steady set-fault attack.

		Before set-fault at Sa		After set-fault at Sa		
a	b	(a_1, a_0)	(b_1, b_0)	Input to C-element	(r_1, r_0)	Result of Computation
0	0	(0,1)	(0,1)	(1,1)	(1,1)	error
0	1	(0,1)	(1,0)	(1,1)	(1,1)	error
1	0	(1,0)	(0,1)	(1,0)	(0,1)	no error
1	1	(1,0)	(1,0)	(1,1)	(1,0)	no error

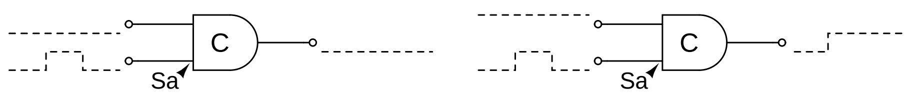

Figure 5.3: Depending on the value of the other input, a C-element may latch high from a transient signal at the Sa input.

Transient Set, Steady and Transient Flip Faults

We will argue that inducing a steady flip, transient flip, or transient set fault at the same target site as in the steady set-fault attack above also allows the attacker to deduce the logical input value.

Suppose, as above, that bit i of the secret is first used as input a to an AND gate. Referring again to Figure 5.2, we would like to induce a fault at the Sa site that will result in either a valid computation or halting in an error state depending on the value of the C-element's other input, a_0. Of course, once a_0 is determined, the logic value of a follows.

It is easy to see that the above attack works with a steady flip fault:

- If $a = 1$, then $(a_1, a_0) = (1, 0)$ and the target C-element has output 0. Flipping the input at Sa makes both C-element inputs 0, and the output remains 0. The expected, error-free, computation follows.

- If $a = 0$, then $(a_1, a_0) = (0, 1)$ and the target C-element has output 0. Flipping the input at Sa make both C-element inputs 1, and the output also goes to 1, also driving the r_1 output to 1. Since $a_0 = 1$, we already have $r_0 = 1$. Thus, this AND gate has a $(1, 1)$ alarm state output, and the computation results in an error.

As for the transient set and transient flip fault models, we need only point out that the only case where the fault must effect a change in the output value of the C-element is when $a = 0$ (and $(a_1, a_0) = (0, 1)$). With $a_0 = 1$, any brief setting of a_1 to 1 will cause the C-element to give a 1 output, and it will remain 1 even after a_1 drops back to 0 as long as a_0 remains at 1. This effect occurs in both the transient flip and transient set threat models.

Steady Flip Model and Power Consumption

In general, the problem with a single steady flip is that it will always induce one of the two types of error states, quiet or alarm . Typical interaction with the device will not reveal which error state resulted from the induced fault. However, if an adversary were to couple the induced fault with (for example) a power consumption trace, it should be clear whether the induced fault caused a quiet

Table 5.4: Outputs marked in Figure 5.2, corresponding minterms, and what we can deduce about the inputs if the corresponding minterm is known to be active.

Active Output	Active Minterm	Logical Implication
R_0	a_0b_0	$a = b = 0$
R_1	a_0b_1	$(a = 0) \wedge (b = 1)$
R_2	a_1b_0	$(a = 1) \wedge (b = 0)$
R_3	a_1b_1	$a = b = 1$

or alarm state due to the likely significantly different levels of power consumption between quiet , where all gates retain (0,0) inputs, and alarm , where (1,1) states cascade throughout the circuit.

By using a side-channel to distinguish between the resulting quiet and alarm error states, the attacker can effectively use steady flip-faults to probe the logic values at the inputs and outputs of gates.

Summary of Attack in Set and Flip Models

The attack is quite efficient: each experiment yields a bit of the secret. Thus, an attacker can recover an n-bit secret by observing and inducing faults in just n computations. We see that dual-rail logic is not secure against steady set, steady flip, transient set, or transient flip threat faults.

5.4.3 Steady Reset Fault Model

As we saw in Section 5.4.2, an attacker who can cause value to change from 0 to 1 can leverage the alarm propagation aspects of the dual-rail gate design to cause the system to divulge the logical values of gate inputs—and secret bits, in particular.

In the reset fault models, however, the induced faults can only change a 1 to a 0, and this present the attacker with a slightly trickier problem. The reason for this problem is that the proposed dual-rail gates are *monotonic*: once there is an input present that causes an output to be 1, setting more bits of that input to 1 will never cause the output to go to 0. Likewise, clearing bits of an input that is producing a 0 output will not cause that output to become 1. Thus, the attacker can only hope to alter a computation by having it result in a deadlock ($(0, 0)$ quiet state) error.[3] The trick is to produce these errors in a data-dependent way that allows the attacker to deduce internal logical values.

Suppose the attacker wishes to learn bit i of the secret. Again, assume that bit i is fed as an input to a dual-rail AND gate, as depicted in Figure 5.2. The attacker will attempt to learn the logic values of the input by clearing the output of one of the four C-elements with outputs labeled R_0, R_1, R_2, or R_3. Notice that when both the a and b inputs contain valid dual-rail data values ($(0, 1)$ or $(1, 0)$), exactly one of these C-elements will be producing a 1 output. This is because each of the C-elements is responsible for detecting one *minterm*. As summarized in Table 5.4, knowing which of the $R_0, \ldots, R_3$ outputs is 1, and therefore which minterm is active, reveals the logic values of the a and b inputs.

[3]This limitation is mostly an artifact of our chosen model. Physical and implementation considerations suggest that the ability to induce reset faults implies the ability to induce set faults. See Chapter 4 for details.

When both the a and b inputs are valid, exactly one of the $R_0, \ldots, R_3$ outputs is 1. Suppose the attacker guesses which of these outputs is active and and clears that output. If he guessed incorrectly, the fault will have no effect because that bit was already fixed at 0. However, if he guesses correctly, he will change the operation of the circuit: what would have been a 1 is now replaced by a 0. This will cause the dual-rail AND gate to output $(0, 0)$, indicating no data present and ultimately causing the computation to deadlock, confirming his guess. Therefore, by repeatedly clearing one of the R_i sites and then observing whether the computation deadlocks or not, the attacker can learn the logical values carried on both inputs to any dual-rail gate.

Now all that is left is for the attacker to decide in what order to attack the four output sites. The most auspicious choice of the order for attacking the output sites depends on what the attacker knows about the inputs.

In the case that the logical value of one of the inputs is known (as when a gate is mixing a key bit with a known plaintext bit), he can narrow the candidate active outputs down to two (e.g., if he knows $b = 1$, he need only determine whether R_1 or R_3 is 1), and only one trial is necessary to determine which is active.

On the other hand, if both inputs are unknown, the attacker may be unlucky and run up to three computations before discovering the active output site and deducing the inputs' logic values. As a consolation prize, he learns both input bits simultaneously. This immediately suggests that in the worst case, an adversary has to run at most $\frac{3}{2}n$ computations to discover n bits. If the attacker attacks gates with two unknown inputs by randomly selecting the order of attack on the $R_0, \ldots, R_3$ sites, however, he can expect to run fewer tests:

$$\begin{aligned}
\text{E[trials to deduce two bits]} &= \sum_{i=1}^{3} i \cdot \Pr[\text{active site known after } i \text{ trials}] \\
&= 1 \cdot \frac{1}{4} + 2 \cdot \frac{3}{4} \cdot \frac{1}{3} + 3 \cdot \frac{1}{2} \\
&= 2\frac{1}{4}.
\end{aligned}$$

Thus, even if none of the secret bits are taken as inputs to gates along with known bits (where the attacks can learn the secret bit in 1 trial), the attacker can expect to run only about $1\frac{1}{8}$ trials per secret bit.[4]

Reset Fault Model Summary

We have show that dual-rail logic is not secure against steady reset faults. It takes only about $1.125n$ trials to learn n secret key bits. For some circuit configurations, the number of trials can be reduced to n.

We are unable to find any workable attack in our transient reset fault model.

5.5 Recommendations

Our attacks utilize induced faults to effectively probe internal logic values, and their success depends on the deterministic nature of the target systems. This observation immediately suggests that

[4] Here was assume that all unknown bits are interesting (i.e., only the n secret bits are unknown). If uninteresting unknown bits are present, it may take twice as many trials for the adversary to learn all secret bits.

randomization, a commonly suggested defense against probing attacks, may perhaps be employed to defend against our fault attacks.

Alternatively, one may note that dual-rail logic is effectively computing with an *error-detecting code*. This allows dual-rail logic to detect errors and halt in an error state rather than divulging an incorrect output that might give the attacker enormous information [8]. Unfortunately, just the two cases of successful versus unsuccessful computation can leak a bit of information. Our attacks and others [9] exploit this bit. With this in mind, one could consider representing logic values with an *error-correcting code* (ECC) rather than with an error detecting code. If logic values were represented in an ECC with a minimum distance of at least three, single-fault errors in a logic value's representation could be *corrected* rather than just *detected*. Thus, single injected faults could reliably result in valid computations rather than the data-dependent behavior of possibly ending in an error state, and this could thwart our attacks. Unfortunately, computing with an ECC would certainly be more expensive than using dual-rail logic: an ECC would require at least 3 wires to carry each bit, and even more wires would be necessary to guarantee balanced power consumption.

Finally, we recommend that a secure embedded device count computations that result in error states. After a number of such suspicious occurrences, the device should, at a minimum, no longer perform computations; if possible, it should also zeroize its secret data, forcing an adversary to acquire another device that uses the same key in order to continue with his attack. By requiring an attacker to acquire several devices that use the same secret key, this type of self-destructive countermeasure could significantly complicate the task of mounting our attacks.

Some of these suggestions might be prohibitively expensive, so, as an alternative, we note that our principled approach permits another solution: namely, changing the assumptions. While dual-rail designs may not be secure in our strong attack models, they may be sufficiently robust in the real world, where adversaries are not so powerful. If a convincing model of limited fault induction (e.g., with less spacial and temporal precision) could be found, the very dual-rail transformation that we examine here may be shown secure. Presumably, our strong model would be a special case of this limited model (one with maximally tight precision parameters), and research could focus on finding efficient transformations that remain robust with attack-model precision parameters approaching ours.

5.6 Conclusion

We have analyzed the efficacy of dual-rail logic as a countermeasure against a variety of types of induced faults. In particular we have found efficient attacks against dual-rail encoded systems in some reasonable threat models. We must conclude that dual-rail encoding, on its own, is not a sufficient countermeasure against single-fault attacks.

Chapter 6

Towards Efficient Second-Order Power Analysis

Power analysis is a major concern for designers of smartcards and other embedded cryptosystems. The advance of Differential Power Analysis (DPA) in 1998 by Paul Kocher [10] made power analysis attacks even more practical since an attacker using DPA did not need to know very much about the device being attacked.

The technique of masking or duplication is commonly suggested as a way to stymie first-order power attacks, including DPA. In order to defeat masking, attacks would have to correlate the power consumption at multiple times during a single computation. Attacks of this sort were suggested and investigated (for example, by Thomas Messerges [11]), but it seems that the attacker was once again required to know significant details about the device under analysis.

It was a comment by Coron and Goubin [12] on a masking proposal of Messerges [14] that prompted the investigation in this chapter. Coron and Goubin found fault with the masking technique employed by Messerges. At the crux of the problem was an assumption about the power consumption model: was it linear? Messerges assumed so while Coron and Goubin (implicitly) assumed not. This was a problem in mismatched models; a principled theoretical approach would have bypassed the confusion introduced by incompatible assumptions.

In this Chapter, we follow a principled theoretical approach in presenting our results by stating the assumptions of our model. In investigating the discrepancy between linearity and nonlinearity assumptions with respect to masking, we find that even in a linear model, an attacker can introduce the necessary nonlinearity with preprocessing techniques and still, relatively efficiently, deduce intermediate secret bits. The technique does not come free of cost, however, as the preprocessing steps significantly decrease the signal to noise ratio.

In Section 6.1, we give some background and contrast first-order and second-order power analysis techniques. We also discuss the apparently inherent costs of higher-order attacks.

In Section 6.2, we discuss our model and give the intuition behind our techniques.

In Section 6.3, we give some techniques for second-order power analysis. In particular, we present some algorithms and analyze them in terms of limitations and requirements: generality, runtime, and number of required traces.

Section 6.4 contains some closing remarks, and Section 6.5, which we consider optional, gives the formal derivations for the noise amplifications that are behind the limitations of the attacks in Section 6.3.

6.1 First-Order and Second-Order Power Analysis

Power analysis is an important class of passive attack. As mentioned in Chapter 2, it is an aggregate attack: it provides information about the entire system, ignoring locality. In this section we specific approaches to power analysis, from the straightforward Simple Power Analysis (SPA) to a powerful statistical variant, Differential Power Analysis (DPA). We also hint at higher-order attacks, which introduce nonlinearity in order to defeat statistical balancing (i.e., masking) countermeasures. In later sections, we develop our own higher-order DPA variants.

6.1.1 First-Order Power Analysis Attacks

First-order attacks are characterized by the property that they exploit highly local correlation of the secret with the power trace. Typically, the secret-correlated power draw occurs at a consistent time during the encryption and has consistent sign and magnitude.

Simple Power Analysis (SPA)

In simple first-order power analysis attacks, the adversary is assumed to have some fairly explicit knowledge of the analyzed cryptosystem. In particular, he knows the time at which the power consumption is correlated with part of the secret. By measuring the power consumption at this time (and perhaps averaging over a few encryptions to reduce the ambiguity introduced by noise), he gains some information about the key.

As a simple example, suppose the attacker knows that the first bit of the key k_0 is loaded into a register at 100μs into the encryption. The average power draw at 100μs is m, but when the k_0 is 0 this average is $m - \delta$ and when k_0 is 1, this average is $m + \delta$. Given enough samples of the power draw at 100μs to distinguish these means (where the number of samples required depends on the level of noise relative to δ), he can determine the first bit of the key.

Differential Power Analysis (DPA)

One of the most amazing and troublesome features of differential power analysis is that, unlike with SPA, the attacker does not need such specific information about how the analyzed device implements its function. In particular, she can be ignorant of the specific times at which the power consumption is correlated with the secret; it is only necessary that the correlation is reasonably consistent.

In differential power analysis attacks, the attacker has identified some intermediate value in the computation that is 1) correlated with the power consumption, and 2) dependent only on the plaintext (or ciphertext or both) and some small part of the key. She gathers a collection of *power traces* by sampling power consumption at a very high frequency throughout a series of encryptions of different plaintexts. If the intermediate value is sufficiently correlated with the power consumption, the adversary can use the power traces to verify guesses at the small part of the key.

In particular, for each possible value of relevant part of the key, the attacker will divide the traces into groups according to the intermediate value predicted by current guess at the key and the trace's corresponding plaintext (or ciphertext); if the averaged power trace of each group differs noticeably from the others (the averaged differences will have a large difference at the time of correlation), it is likely that the current key guess is correct. Since incorrectly predicted intermediate value will not be correlated with the measured power traces, incorrect key guesses should result in all groups having very similar averaged power traces.

6.1.2 Higher-Order Attacks

A higher-order attack addresses a situation where there is some intermediate value (or set of values) that depends only on the plaintext and some small part of the key, but it is not correlated directly with the power consumption at any particular time. Instead, this value contributes to the joint distribution of the power consumption at a few times during the computation.

An important example of such a situation comes about when the *masking* (or duplication) technique is employed to protect against first-order attacks. As a typical example of masking, consider an implementation that wishes to perform a computation using some intermediate, key-dependent bit b. Rather than computing directly with b and opening itself up to DPA attacks, however, it performs the computation twice: once with a random bit r, then with the *masked bit* $(r + b)$.[1] The implementation is designed to use these two masked intermediate results as inputs to the rest of the computation.

In this case, knowledge of either r or $r + b$ alone is not of any use to the attacker. Since the first-order attacks look for local, linear correlation of b with the power draw, they are stymied. If, however, an attack could correlate the power consumption at the time r is present and the time $r + b$ is present (e.g., by multiplying the power consumptions at these times), it could gain some information on b.

For example, suppose a cryptographic device employs masking to hide some intermediate bit b that is derived directly from the key, but displays the following behavior: at 100μs, the average power draw is $m + \delta(-1)^r$ and at 210μs it is $m + \delta(-1)^{(r+b)}$. An attacker aware of this fact could multiply the samples at these times for each trace and obtain a product value with expected value[2]

$$\text{E[product of samples]} = \begin{cases} m^2 + \delta^2 & \text{if } b = 0 \\ m^2 - \delta^2 & \text{if } b = 1 \end{cases} \tag{6.1}$$

Summing the samples over n encryptions, the means would be $n(m^2 + \delta^2)$ for $b = 0$ and $n(m^2 - \delta^2)$ for $b = 1$. By choosing n large enough to reduce the relative effect of noise, the attacker could distinguish these distributions and deduce b. An attack of this sort is the second-order analog of an SPA attack.

But how practical is this really? A higher-order attack seems to face two major problems:

- How much does the process of correlation amplify the noise, thereby increasing standard deviation and requiring more samples to reliably differentiate distributions?

- How does it identify the times when the power consumption is correlated with an intermediate value?

The first issue is apparent when calculating the standard deviation of the product computed in the above attack. If the power consumption at times 100μs and 210μs both have standard deviation σ, then the product has standard deviation

$$\sigma_{\text{product}} = \begin{cases} \sqrt{\sigma^4 + 2\sigma^2 m^2 + 4\delta^2 m^m + 2\delta^2\sigma^2} & \text{if } b = 0, \\ \sqrt{\sigma^4 + 2\sigma^2 m^2 + 2\delta^2\sigma^2} & \text{if } b = 1 \end{cases} \tag{6.2}$$

[1]Though we use the symbol '+' to denote the masking operation, we require nothing from it other than that $c = a + b$ implies $(-1)^c = (-1)^{a+b}$; for our purposes, it is convenient to just assume that '+' is exclusive-or.

[2]Here the expectation is taken over both the random noise and the value of the masking bit r, and the noise components at times 100μs and 210μs are assumed independent.

effectively squaring the standard deviation of zero-mean noise. This means that substantially many more samples are required to distinguish the $b = 0$ and $b = 1$ distributions than would be required in a first-order attack, if one were possible.

The second issue is essentially the higher-order analog of the problem with SPA: attackers require exact knowledge of the time at which the intermediate value and the power consumption are correlated. DPA resolves this problem by considering many samples of the power consumption throughout an encryption. Unfortunately, the natural generalization of this approach to even second-order attacks, where a product would be accumulated for each (t_1, t_2) time pair, is extremely computationally taxing. The second-order attacks discussed in this chapter avoid this overhead.

6.2 The Power Analysis Model

Both of the attacks we present are second-order attacks which are essentially preprocessing steps applied to the power traces followed by standard DPA.

In this section, we build on the general model presented in Section 3.4.1 and and present standard DPA in this framework, both as a point of reference and as a necessary subroutine for our attacks, which are described in Section 6.3.

6.2.1 The Model

In this section, we augment the model of Section 3.4.1 in order to account for masking techniques. We also fill in the details (e.g., noise distribution and trace construction) necessary for our analysis in this Chapter.

We assume that the attacker has guessed part of the key and has predicted an intermediate bit value b for each of the power traces, grouping them into a $b = 0$ and a $b = 1$ group. For simplicity, we assume there are n traces in each of these groups: trace i from group b is called T_i^b, where $0 \leq i < n$. Each trace contains samples at m evenly spaced times; the sample at time t from this trace is denoted $T_i^b(t)$, where $0 \leq t < m$.

Each sample has a *noise* component and possibly a *signal* component, if it is correlated with b. We assume that each noise component is Gaussian with equal standard deviation and independent of the noise in other samples in its own trace and other traces. For simplicity, we also assume that the input has been normalized so that each noise component is a 0-mean Gaussian with standard deviation one (i.e., $\sim \mathcal{N}(0, 1)$). The random variable for the noise component in trace i from group b at time t is $S_i^b(t)$, for $0 \leq t < m$.

We assume that the device being analyzed is utilizing masking so that there is a uniformly distributed independent random variable for each trace that corresponds to the masking bit; it will be more convenient for us to deal with $\{\pm 1\}$ bit values, so if the random bit in trace i from group b is r, we define the random variable $R_i^b = (-1)^r$.

Finally, if the guess for b is correct, the power consumption is correlated with the random masking bit and the intermediate value b at the same times in each trace. Specifically, we assume that there is some parameter d (in units of the standard deviation of the noise) and times c_0 and c_1 such that the random bit makes a contribution of dR_i^b to the power consumption at time c_0 and the masked bit makes a contribution of $d(-1)^b R_i^b$ at time c_1.

We can now characterize the trace sample distributions in terms of these noise and signal components:

- If the guess of the key is correct, then for $0 \leq i < n$, $0 \leq t < m$, and $b \in \{0,1\}$, we have:

$$T_i^b(t) = \begin{cases} S_i^b(t) + dR_i^b & \text{if } t = c_0 \\ S_i^b(t) + d(-1)^b R_i^b & \text{if } t = c_1 \\ S_i^b(t) & \text{otherwise} \end{cases} \tag{6.3}$$

- If the key is predicted incorrectly, however, then the groups are not correlated with the true value of b in each trace and hence there is no correlation between the grouping and the power consumption in the traces, so, for $0 \leq i < n$, $0 \leq t < m$, and $b \in \{0,1\}$:

$$T_i^b(t) = S_t^b(t) \tag{6.4}$$

Given these traces as inputs, the algorithms try to decide whether the groupings (and hence the guess for the key) are correct by distinguishing these distributions.

6.2.2 The Generic DPA Subroutine

Both algorithms use a subroutine DPA after their preprocessing step. For our purposes, this subroutine simply takes the two groups of traces, T^0 and T^1, a threshold value τ, and determines whether the groups' totalled traces differ by more than τ at any sample time. If the difference of the totalled traces is greater than τ at any point, DPA returns 1, indicating that T^0 and T^1 have different distributions; if the difference is no more than τ at any point, DPA returns 0, indicating that it thinks T^0 and T^1 are identically distributed.

DPA(T^0, T^1, τ)
    for each $t \in \{0, \ldots, m-1\}$:
        $s \leftarrow 0$
        for each $i \in \{0, \ldots, n-1\}$:
            $s \leftarrow s + T_i^0(t) - T_i^1(t)$
        if $|s| > \tau$ return 1
    return 0

When using the DPA subroutine, it is most important to pick the threshold, τ, appropriately. Typically, to minimize the impact of false positives and false negatives, τ should be half the difference. This is perhaps unexpected since false positives are actually far more likely than false negatives when using a midpoint threshold test since false positives can occur if *any* of the m times' samples sum deviates above τ, while false negatives require exactly the correlated time's samples to deviate below τ. The reason for not choosing τ to equalize the probabilities is that false negatives are far more detrimental than false positives: an attack suggesting two likely subkeys is more helpful than an attack suggesting none.

An equally important consideration in using DPA is whether τ is large enough compared to the noise to reduce the probability of error. Typically, the samples' noise components will be independent and the summed samples' noise will be Gaussian, so we can can achieve negligible probability of error by using n large enough that τ is some constant multiple of the standard deviation.

DPA runs in time $\Theta(nm)$. Each run of DPA decides the correctness of only one guessed grouping, however, so an attack that tries l groupings runs in time $\Theta(nml)$.

6.3 Our Second-Order Attacks

The two second-order variants of DPA that we discuss are Zero-Offset 2DPA and FFT 2DPA. The former is applied in the special but not necessarily unlikely situation when the power correlation times for the two bits are coincident (i.e., the random bit r and the masked bit $r+b$ are correlated with the power consumption at the same time). The latter attack applies to the more general situation where the attacker does not know the times of correlation; it discovers the correlation with only slight computational overhead but pays a price in the number of required samples.

6.3.1 Zero-Offset 2DPA

Zero-Offset 2DPA is a very simple variation of ordinary first-order DPA that can be applied against systems that employ masking in such a way that both the random bit r and the masked intermediate bit $r+b$ correlate with the power consumption at the same time. In the language of our model, $c_0 = c_1$.

The coincident effect of the two masked values may seem to be too specialized of a circumstance to occur in practice, but it does come up. The motivation for this attack is the claim by Coron and Goubin [12] that some techniques suggested by Messerges [14] were insecure due to some register containing the multi-bit intermediate value a or its complement $\overline{a}$. Since Messerges assumes a power consumption model based on Hamming weight, it was not clear how a first-order attack would exploit this register. However, we observe that such a system can be attacked (even in the Hamming model) by a Zero-Offset 2DPA that uses as its intermediate value the exclusive-or of the first two bits of a. Another example of a situation with coincident power consumption correlation is in a paired circuit design that computes with both the random and masked inputs in parallel.

Combining $c_0 = c_1$ with Equation (6.3), we see that in a correct grouping:

$$T_i^b(t) = \begin{cases} S_i^b(t) + dR_i^b + d(-1)^b R_i^b & \text{if } t = c_0 \\ S_i^b(t) & \text{otherwise} \end{cases} \tag{6.5}$$

In an incorrect grouping, $T_i^b(t)$ is distributed exactly as in the general uncorrelated case in Equation (6.4).

Note that in a correct grouping, when $b = 1$, the influence of the two bits cancel, leaving $T_i^1(c_0) = S_i^1(c_0)$, while when $b = 0$, the influences of the two bits combine constructively and we get $T_i^0(c_0) = S_i^0(c_0) + 2dR_i^0$. In the former case, there appears to be no influence of the bits on the power consumption distribution, but in the latter case, the bits contribute a bimodal component. The bimodal component has mean 0, however, so it would not be apparent in a first-order averaging analysis.

Zero-offset 2DPA exploits the bimodal component for the $b = 0$ case by simply squaring the samples in the power traces before running straight DPA.

`Zero-Offset-2DPA`(T^0, T^1)
    for each $b \in \{0,1\}$, $i \in \{0, \ldots, n\}$, $t \in \{0, \ldots, m\}$:
        $T_i^b(t) \leftarrow (T_i^b(t))^2$
    return `DPA`$(T^0, T^1, 2nd^2)$

Why does this work? Suppose we have a correct grouping and consider the expected values for the sum of the squares of the samples at time c_0 in the two groups:

- if $b = 0$,

$$\begin{aligned} \mathrm{E}\left[\sum_{i=0}^{n-1}[T_i^0(c_0)]^2\right] &= \sum_{i=0}^{n-1}\mathrm{E}[(S_i^0(c_0))^2 + 4dS_i^0(c_0)R_i^0 + 4d^2(R_i^0)^2] \\ &= \sum_{i=0}^{n-1}\left(\mathrm{E}[(S_i^0(c_0))^2] + \mathrm{E}[4dS_i^0(c_0)R_i^0] + \mathrm{E}[4d^2(R_i^0)^2]\right) \\ &= \sum_{i=0}^{n-1}(1 + 0 + 4d^2) \\ &= 4nd^2 + n \end{aligned} \tag{6.6}$$

- if $b = 1$,

$$\begin{aligned} \mathrm{E}\left[\sum_{i=0}^{n-1}[T_i^1(c_0)]^2\right] &= \sum_{i=0}^{n-1}\mathrm{E}[(S_i^1(c_0))^2] \\ &= \sum_{i=0}^{n-1}1 \\ &= n \end{aligned} \tag{6.7}$$

The above derivations use the fact that if $S \sim \mathcal{N}(0,1)$ then $S^2 \sim \chi^2(1,0)$ (i.e., S^2 has χ^2 distribution with $\nu = 1$ degree of freedom and non-centrality parameter $\delta^2 = 0$), and the expected value of a $\chi^2(\nu,\delta^2)$ random variable is $\nu + \delta^2$.

Thus, the expected difference of the sum of products for the c_0 samples is $4nd^2$, while the expected difference for incorrect groupings is clearly 0. In Section 6.5.1, we show that the difference of the groups' sums of products is essentially Gaussian with standard deviation

$$\sigma = \sqrt{n(16d^2 + 4)}. \tag{6.8}$$

For an attack that uses a DPA threshold value at least k standard deviations from the mean, we will need at least $k^2 \cdot \frac{(4d^2+1)}{4d^4}$ traces. This $\frac{(4d^2+1)}{4d^4}$ blowup factor may be substantial; recall that d is in units of the standard deviation of the noise, so it may be significantly less than 1.

The preprocessing in `Zero-Offset-DPA` takes time $\Theta(nm)$. After this preprocessing, each of l subsequent guessed groupings can be tested using `DPA` in time $\Theta(nm)$, for a total runtime of $\Theta(nm + nml) = \Theta(nml)$. It is important to keep in mind when comparing these run times that the number n of required traces for `Zero-Offset-DPA` can be somewhat larger than would be necessary for first-order DPA—if a first-order attack were possible.

A Natural Variation: Known-Offset 2DPA

If the difference $s = c_1 - c_0$ is non-zero but known, a similar attack may be mounted. Instead of calculating the squares of the samples, the adversary can calculate the lagged product:

$$L_i^b(t,s) = T_i^b(t) \cdot T_i^b(t+s), \tag{6.9}$$

where the addition $t + s$ is intended to be cyclic in $\{0,\ldots n-1\}$.

This lagged product at the correct offset $s = c_1 - c_0$ has properties similar to the squared samples discussed above, and can be used in the same way.

6.3.2 FFT 2DPA

Fast Fourier Transform (FFT) 2DPA is useful in that it is more general than Zero-Offset 2DPA: it does not require that the times of correlation be coincident, and it does not require any particular information about c_0 and c_1.

To achieve this, it uses the FFT to compute the *correlation* of a trace with itself—an *autocorrelation*. The autocorrelation A_i^b of a trace T_i^b is also defined on values $t \in \{0, \ldots, m-1\}$, but this argument is considered an *offset* or *lag* value rather than an absolute time. Specifically, for $b \in \{0, 1\}$, $0 \leq i < n$, and $0 \leq t < m$,

$$A_i^b(t) = \sum_{j=0}^{m-1} T_i^b(j) \cdot T_i^b(j+t) \tag{6.10}$$

The argument $t+j$ in $T_i^b(j+t)$ is understood to be cyclic in $\{0, \ldots, m-1\}$, so that $A_i^b(t) = A_i^b(m-t)$, and we really only need to consider $0 \leq t \leq m/2$.

To see why $A_i^b(t)$ might be useful, recall Equation (6.3) and notice that most of the terms of $A_i^b(t)$ are of the form $S_i^b(j) \cdot S_i^b(j+t)$; in fact, the only terms that differ are where j or $j+t$ is c_0 or c_1. This observation suggests a way to view the sum for $A_i^b(t)$ by splitting it up by the different types of terms from Equation (6.3), and in fact it is instructive to do so. To simplify notation, let $Q = \{c_0 - t, c_0, c_1 - t, c_1\}$, the set of "interesting" indices, where the terms of $A_i^b(t)$ are "unusual" when $j \in Q$. Assuming $t \neq c_1 - c_0$,

$$\begin{aligned}
A_i^b(t) &= S_i^b(c_0 - t) \cdot [S_i^b(c_0) + dR_i^b] \\
&+ [S_i^b(c_0) + dR_i^b] \cdot S_i^b(c_0 + t) \\
&+ S_i^b(c_1 - t) \cdot [S_i^b(c_1) + d(-1)^b R_i^b] \\
&+ [S_i^b(c_1) + d(-1)^b R_i^b] \cdot S_i^b(c_1 + t) \\
&+ \sum_{j \notin Q} S_i^b(j) \cdot S_i^b(j+t)
\end{aligned} \tag{6.11}$$

and we can distribute and recombine terms to get

$$\begin{aligned}
A_i^b(t) &= [S_i^b(c_0 - t) + S_i^b(c_0 + t)] \cdot dR_i^b \\
&+ [S_i^b(c_1 - t) + S_i^b(c_1 + t)] \cdot d(-1)^b R_i^b \\
&+ \sum_{j=0}^{m-1} S_i^b(j) \cdot S_i^b(j+t).
\end{aligned} \tag{6.12}$$

Using Equation (6.12) and the fact that $\mathrm{E}[XY] = \mathrm{E}[X] \cdot \mathrm{E}[Y]$ when X and Y are independent random variables, it is straightforward to verify that $\mathrm{E}[A_i^b(t)] = 0$ when $t \neq c_1 - c_0$; its terms in that case are products involving some 0-mean independent random variable (this is exactly what we show in Equation (6.15)). On the other hand, $A_i^b(c_1 - c_0)$ involves terms that are products of dependent random variables, as can be seen by reference to Equation (6.10). We make frequent use of Equation (6.12) in our derivations in this section and in Section 6.5.2.

This technique requires a subroutine to compute the autocorrelation of a trace:

```
Autocorrelate(T)
    F ← FFT(T)
    for each t ∈ {0, ..., m − 1}:
```

```
3 :         F(t) ← |F(t)|^2
4 :     return Inv-FFT(F)
```

The $|F(t)|^2$ in line 3 is the squared $\mathcal{L}_2$-norm of the complex number $F(t)$ (i.e., $|F(t)|^2 = \overline{F(t)} \cdot F(t)$, where $\overline{\alpha}$ denotes the complex conjugate of α).

The subroutine FFT computes the usual Discrete Fourier Transform:

$$(\texttt{FFT}(T))(x) = \sum_{j=0}^{m-1} T(j) \cdot \omega^{-xj} \tag{6.13}$$

and Inv-FFT computes the Inverse Discrete Fourier Transform:

$$(\texttt{Inv-FFT}(T))(y) = \frac{1}{m} \sum_{j=0}^{m-1} T(j) \cdot \omega^{xj} \tag{6.14}$$

In the above equations, ω is a complex primitive mth root of unity (i.e., $\omega \in \mathbb{C}$, $\omega^m = 1$, and $\omega^k \neq 1$ for all $0 < k < m$).

The subroutines FFT, Inv-FFT, and therefore Autocorrelate itself all run in time $\Theta(m \log m)$.

We can now define the top-level FFT-2DPA algorithm:

```
FFT-2DPA(T^0, T^1, τ)
    for each b ∈ {0,1}, t ∈ {0,...,m−1}:
        Z^b(t) ← 0
    for each b ∈ {0,1}, i ∈ {0,...,n−1}:
        A^b ← Autocorrelate(T_i^b)
        for each t ∈ {0,...,m−1}:
            Z^b(t) ← Z^b(t) + A^b(t)
    return DPA(Z^0, Z^1, nd^2)
```

What makes this work? Assuming a correct grouping, the expected sums are:

- $t \neq c_1 - c_0$:

$$\begin{aligned}
\mathrm{E}[Z^b(t)] &= \mathrm{E}\left[\sum_{i=0}^{n-1}\sum_{j=0}^{m-1}(T_i^b(j) \cdot T_i^b(j+t))\right] \\
&= n \sum_{j=0}^{m-1} \mathrm{E}[T_0^b(j) \cdot T_0^b(j+t)] \\
&= n\,\mathrm{E}\left[[S_0^b(c_0 - t) + S_0^b(c_0+t)] \cdot dR_0^b\right] \\
&+ n\,\mathrm{E}\left[[S_0^b(c_1 - t) + S_0^b(c_1+t)] \cdot d(-1)^b R_0^b\right] \\
&+ nm\,\mathrm{E}\left[S_0^b(0) \cdot S_0^b(0+t)\right] \\
&= 0
\end{aligned} \tag{6.15}$$

- $t = (c_1 - c_0)$:

$$\begin{aligned}
\mathrm{E}[Z^b(t)] &= \mathrm{E}\left[\sum_{i=0}^{n-1}\sum_{j=0}^{m-1}(T_i^b(j)\cdot T_i^b(j+t))\right] \\
&= n\sum_{j=0}^{m-1}\mathrm{E}[T_0^b(j)\cdot T_0^b(j+t)] \\
&= n\,\mathrm{E}\left[[S_0^b(c_0-t)+S_0^b(c_1)]\cdot dR_0^b\right] \\
&+ \; n\,\mathrm{E}\left[[S_0^b(c_0)+S_0^b(c_1+t)]\cdot d(-1)^b R_0^b\right] \\
&+ \; n\,\mathrm{E}[d^2(R_0^b)^2(-1)^b] \\
&+ \; nm\,\mathrm{E}\left[S_0^b(0)\cdot S_0^b(0+t)\right] \\
&= 0+0+n\,\mathrm{E}[d^2(R_0^b)^2(-1)^b]+0 \\
&= nd^2(-1)^b \qquad (6.16)
\end{aligned}$$

So in a correct grouping, we have

$$\mathrm{E}[Z^0(t)-Z^1(t)] = \begin{cases} 2nd^2 & \text{if } t=c_1-c_0 \\ 0 & \text{otherwise.}\end{cases} \qquad (6.17)$$

In incorrect groupings, however, $\mathrm{E}[Z^0(t)-Z^1(t)]=0$ for all $t\in\{0,\ldots,m-1\}$.

In Section 6.5.2, we see that this distribution is closely approximated by a Gaussian with standard deviation $\sigma=\sqrt{n(8d^2+2m)}$, so that an attacker who wishes to use a threshold at least k standard deviations away from the mean needs n to be at least about $k^2\cdot\frac{(4d^2+m)}{2d^4}$.

Note that the noise from the other samples contributes significantly to the standard deviation at $Z^b(c_1-c_0)$, so this attack would only be practical for relatively short traces and a significant correlated bit influence (i.e., when m is small and d is not much smaller than 1).

The preprocessing in `FFT-2DPA` runs in time $\Theta(nm\log m)$. After this preprocessing, however, each of l guessed groupings can be tested using `DPA` in time $\Theta(nm)$, for a total runtime of $\Theta(nm\log m+nml)$, amortizing to $\Theta(nml)$ if $l=\Omega(\log m)$. Again, when considering this runtime, it is important to keep in mind that the number n of required traces can be substantially larger than would be necessary for first-order DPA—if a first-order attack were possible.

FFT and Known-Offset 2DPA

It might be very helpful in practice to use the FFT in second-order power analysis attacks for attempting to determine the offset of correlation. With a few traces, it could be possible to use an FFT to find the offset s of repeated computations, such as when the same function is computed with the random bit r at time c_0 and with the masked bit $r+b$ at time c_0+s.

With even a few values of s suggested by an FFT on these traces, a Known-Offset 2DPA attack could be attempted, which could require far fewer traces than straight FFT 2DPA since Known-Offset 2DPA suffers from less noise amplification.

6.4 Conclusion

We explored two second-order attacks that attempt to defeat masking while minimizing computation resource requirements in terms of space and time.

The first, Zero-Offset 2DPA, works in the special situation where the masking bit and the masked bit are coincidentally correlated with the power consumption, either canceling out or contributing a bimodal component. It runs with almost no noticeable overhead over standard DPA, but the number of required power traces increases more quickly with the relative noise present in the power consumption.

The second technique, FFT 2DPA, works in the more general situation where the attacker knows very little about the device being analyzed and suffers only logarithmic overhead in terms of runtime. On the other hand, it also requires many more power traces as the relative noise increases.

In summary, we expect that Zero-Offset 2DPA and Known-Offset 2DPA can be of some practical use, but FFT 2DPA probably suffers from too much noise amplification to be generally effective. However, if the traces are fairly short and the correlated bit influence fairly large, it can be effective.

6.5 Analysis: Noise Amplification

In this section, we attempt to characterize the distribution of the estimators that we use to distinguish the target distributions. In particular, we show that the estimators have near-Gaussian distributions and we calculate their standard deviations.

6.5.1 Zero-Offset 2DPA

As in Section 6.3.1, we assume that the times of correlation are coincident, so that $c_0 = c_1$. From this, we get that the distribution of the samples in a correct grouping follows Equation (6.5):

$$T_i^b(t) = \begin{cases} S_i^b(t) + dR_i^b + d(-1)^b R_i^b & \text{if } t = c_0 \\ S_i^b(t) & \text{otherwise} \end{cases} \tag{6.18}$$

The sum

$$\sum_{i=0}^{n-1} [T_i^0(c_0)]^2 = \sum_{i=0}^{n-1} [S_i^0(c_0) + 2dR_i^0]^2 \tag{6.19}$$

is then a $\chi^2(\nu, \delta^2)$-distributed random variable with $\nu = n$ degrees of freedom and non-centrality parameter $\delta^2 = \sum_{i=0}^{n-1}(2dR_i^0)^2 = 4nd^2$. It has mean $\nu + \delta^2 = 4nd^2 + n$ and standard deviation $\sqrt{2(\nu + 2\delta^2)} = \sqrt{2(n + 8nd^2)} = \sqrt{n(16d^2 + 2)}$.

A common rule of thumb is that χ^2-distributed random variables with over thirty degrees of freedom are closely approximated by Gaussians. We expect $n \gg 30$, so we say

$$\sum_{i=0}^{n-1} [T_i^0(c_0)]^2 \sim \mathcal{N}\left(4nd^2 + n, \sqrt{n(16d^2 + 2)}\right). \tag{6.20}$$

Similarly, we obtain $\sum_{i=0}^{n-1} [T_i^1(c_0)]^2 \sim \chi^2(n, 0)$, which, since $n \gg 30$, we approximate with

$$\sum_{i=0}^{n-1} [T_i^1(c_0)]^2 \sim \mathcal{N}\left(n, \sqrt{2n}\right). \tag{6.21}$$

The difference of the summed squares is then

$$\sum_{i=0}^{n-1} \left[(T_i^0(c_0))^2 - (T_i^1(c_0))^2\right] \sim \mathcal{N}\left(4nd^2, \sqrt{n(16d^2 + 4)}\right). \tag{6.22}$$

6.5.2 FFT 2DPA

Recalling our discussion from Section 6.3.2, we want to examine the distribution of

$$Z^b(t) = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} T_i^b(j) \cdot T_i^b(j+t). \tag{6.23}$$

when $t = c_1 - c_0$. Its standard deviation should dominate that of $Z^b(t')$ for $t' \neq c_1 - c_0$ (for simplicity, we assume $c_1 - c_0 \neq c_0 - c_1$).

In Section 6.3.2, we saw that $\mathrm{E}[Z^b(t)] = nd^2(-1)^b$. We would now like to calculate its standard deviation.

In the following, we liberally use the fact that

$$\mathrm{Var}[X+Y] = \mathrm{Var}[X] + \mathrm{Var}[Y] + 2\,\mathrm{Cov}[X,Y], \tag{6.24}$$

where $\mathrm{Cov}[X,Y]$ is the *covariance* of X and Y ($\mathrm{Cov}[X,Y] \triangleq \mathrm{E}[XY] - \mathrm{E}[X]\,\mathrm{E}[Y]$). We would often like to add variances of random variables that are not independent; Equation (6.24) says we can do so if the random variables have 0 covariance.

Since the traces are independent and identically distributed,

$$\begin{aligned}
\mathrm{Var}[Z^b(t)] &= \sum_{i=0}^{n-1} \mathrm{Var}\left[\sum_{j=0}^{m-1} T_i^b(j) \cdot T_i^b(j+t)\right] \\
&= n\,\mathrm{Var}\left[\sum_{j=0}^{m-1} T_0^b(j) \cdot T_0^b(j+t)\right] \\
&= n\,\mathrm{Var}\left[dR_0^b([S_i^b(c_0-t) + S_0^b(c_1)] + (-1)^b[S_0^b(c_0) + S_0^b(c_1+t)])\right] \\
&+ n\,\mathrm{Var}\left[\sum_{j=0}^{m-1} S_0^b(j) \cdot S_0^b(j+t)\right]
\end{aligned} \tag{6.25}$$

where we were able to split the variance in the last line since the two terms have 0 covariance.

To calculate $\mathrm{Var}\left[dR_0^b([S_0^b(c_0-t) + S_0^b(c_1)] + (-1)^b[S_0^b(c_0) + S_0^b(c_1+t)])\right]$, note that its terms have 0 covariance. For example:

$$\begin{aligned}
\mathrm{Cov}[dR_0^bS_0^b(c_0-t), dR_0^bS_0^b(c_1)] &= \mathrm{E}[(dR_0^b)^2 S_0^b(c_0-t) \cdot S_0^b(c_1)] \\
&- \mathrm{E}[dR_0^bS_0^b(c_0-t)]\,\mathrm{E}[dR_0^bS_0^b(c_1)] \\
&= 0 - 0 = 0
\end{aligned} \tag{6.26}$$

since the expectation of a product involving an independent 0-mean random variable is 0. Furthermore, it is easy to check that each term has the same variance, and

$$\begin{aligned}
\mathrm{Var}[dR_0^bS_0^b(c_1)] &= \mathrm{E}\left[[dR_0^bS_0^b(c_1)]^2\right] - \mathrm{E}[dR_0^bS_0^b(c_1)]^2 \\
&= d^2\,\mathrm{E}\left[[S_0^b(c_1)]^2\right] - 0 \\
&= d^2,
\end{aligned} \tag{6.27}$$

for a total contribution of

$$\mathrm{Var}\left[dR_0^b([S_0^b(c_0-t) + S_0^b(c_1)] + (-1)^b[S_0^b(c_0) + S_0^b(c_1+t)])\right] = 4d^2. \tag{6.28}$$

The calculation of Var $\left[\sum_{j=0}^{m-1} S_0^b(j) \cdot S_0^b(j+t)\right]$ is similar since its terms also have covariance 0 and they all have the same variance. Thus,

$$\begin{aligned}\mathrm{Var}\left[\sum_{j=0}^{m-1} S_0^b(j) S_0^b(j+t)\right] &= m\,\mathrm{Var}[S_0^b(0) S_0^b(0+t)] \\ &= m\left(\mathrm{E}\left[[S_0^b(0)]^2[S_0^b(t)]^2\right] - \mathrm{E}[S_0^b(0)S_0^b(t)]^2\right) \\ &= m(1+0) = m. \end{aligned} \tag{6.29}$$

Finally, plugging Equations (6.28) and (6.29) into Equation (6.25), we get the result

$$\mathrm{Var}[Z^b(t)] = n(m + 4d^2) \tag{6.30}$$

and the corresponding standard deviation is $\sqrt{n(m+4d^2)}$.

As in Section 6.5.1, we expect n to be large and we say

$$Z^b(t) \sim \mathcal{N}\left(nd^2(-1)^b, \sqrt{n(4d^2+m)}\right). \tag{6.31}$$

Finally, we get the distribution of the difference:

$$Z^0(t) - Z^1(t) \sim \mathcal{N}\left(2nd^2, \sqrt{n(8d^2+2m)}\right). \tag{6.32}$$

Chapter 7

Related Work

7.1 Passive Attacks

Most of today's researchers examining the security of embedded systems address passive attacks. This is true whether or not they focus on employing rigorous, theoretical methods. Consequently, much of the theoretical progress has been made with models that capture passive attacks.

7.1.1 Towards Approaches to Counteract Power-Analysis Attacks

Chari, Jutla, Rao, and Rohatgi address passive aggregate attacks (e.g., power analysis) where the attacker is hindered by noisy information. [18] They choose a very strong attacker model, where almost all computations contribute to distinct elements of the power trace. That is, there is strong locality in the power measurements. They implicitly assume that the only simultaneous components of power consumption are bits in the same register. Assuming a register of size r contains one secret bit and $r-1$ random bits, they develop what is effectively a noisy probing channel. They use this model to analyze masking defenses; they show an exponential lower bound on the number of traces necessary to extract a secret bit in terms of the number of masking bits used.

7.1.2 Private Circuits: Securing Hardware against Probing Attacks

Ishai, Sahai, and Wagner address passive probing attacks by assuming a circuit model where the attacker can probe some finite number k of wires per computation. [15] They propose a generic transformation $T_k : C \mapsto C'$ where C is any circuit and C' is an equivalent circuit that uses random masking to preserve secrets even when attacker gets k wire-probes per computation. This results in an $n \to O(nt^2)$ blowup in the number of gates. More efficient transformations are also presented which provide different security guarantees.

7.1.3 Physically Observable Cryptography

Micali and Reyzin focus on passive attacks, which they call "Physical Observation" attacks, and aim for the most pessimistic model that still permits cryptography. [17] Their goal is to produce pseudorandom generators in *partially shielded* hardware. Their approach is very rigorous and follows analogously the standard definitions from complexity-based cryptography. They augment Turing Machines to include *leakage functions* and use this model to define pseudorandomness in the context of physical observation via leakage functions.

Rather than assuming limitations on the adverary's ability to exploit physical phenomena in observing the computation, they instead relate the adversary's effectiveness at attacking their constructions in terms of the ability to attack (using physical observation) their primitive *physically observable one-way functions*. This approach mirrors the proof of security by reduction approach used in classical theoretical cryptography. They construct, in this framework, *unpredictable* physically observable pseudorandom generators with arbitrary expansion and *indistinguishable* physically observable pseudorandom generators with 1-bit expansion.

7.2 Active Attacks

Active attacks are inherently difficult to address in a rigorous manner. Addressing active attacks coupled with passive observation seems even more daunting; nevertheless, there has been some respectable progress in that direction.

7.2.1 Algorithmic Tamper-Proof (ATP) Security

Gennaro, Lysyanskaya, Malkin, Micali, and Rabin separate devices into unreadable but tamperable components (e.g. RAM) and untamperable but readable components (e.g., circuitry). [16] They assume a powerful tampering adversary: he can repeatedly choose and apply a polynomially-computable transform $f()$ to the device's memory (without directly reading the memory) and then interact with the device. They hope to achieve *Algorithmic Tamper-Proof (ATP)* security, where even such a powerful tampering adversary can do no better than one who uses the device as a black-box. Necessary and sufficient conditions for achieving this security are:

1. a self-destruct capability
2. a small section of unreadable hardwired data (e.g., a public key).

Alternatively, they show how to achieve ATP security in devices without either of the above two augmentations if the class of tampering functions f available to the adversary is limited in a realistic manner (e.g., only random faults and targeted bit-flips).

Their models are flexible yet conservative enough to derive useful and convincing results.

Chapter 8

Conclusion

8.1 In Summary

8.1.1 The Benefits of Rigor

In Chapter 1, discussed the merits of what we call a *theoretical approach*. Essentially, we are referring to a principled approach that employs rigor at every step and is as explicit as possible about its assumptions. We posited that such an approach yields two primary benefits:

Comprehensibility and Comparability A well-defined model and rigorous application of logic provides results that can easily be evaluated and compared to other results. Much confusion can avoided and the synergistic effects of researchers building on each others' results are realized only when problems involving the ambiguity and incompatibility of assumptions and models are addressed. A principled, theoretical approach avoids these problems naturally.

Simplification and Refinement Carefully constructed models can abstract away inconvenient real-world details; analysis in these idealized models can provide insight into the nature of particular aspects of real-world phenomena that might be obscured when not viewed in isolation. It is true that all investigations work with at least an implicit model that incorporates some idealization. With a principled approach, however, the assumptions and abstractions used in creating the ideal model are *explicit*. Even when results derived from flawed and incomplete but otherwise *rigorously defined* models are inapplicable in the real world, they can be used as starting points for developing complete and practical results (i.e., by augmenting the model). When the assumptions underneath a result are not apparent, however, that result is not a suitable foundation on which to build new results.

8.1.2 Theory of Embedded Systems

The study of security in embedded systems is fairly new. We are only beginning to apply rigorous approaches. New models and techniques are being proposed, applied and evaluated; while some will certainly be dismissed, others will become the foundation on which future theoretical study of secure embedded computation will rest. This thesis represents some of the earliest thinking in this direction.

We have argued why we feel a theoretical approach would be useful and perhaps necessary for the advancement of security research in this area. Furthermore, we presented a high-level philosophy towards selecting models that can serve as a useful guide to those who wish to carry out a rigorous analysis. This approach has been demonstrated by the selection and application of a couple of models in deriving some interesting results about secure embedded systems.

In particular, with the thoughtful selection of models and application of rigorous techniques, we have:

- Described and analyzed our own proposed extensions to Differential Power Analysis for overcoming masking defenses (Chapter 6). In doing so, we helped resolve a discrepancy in assumptions that lead to some confusion. This confusion would have been avoided if a principled approach had been employed at the outset.

- Shown dual-rail encoding to be an inadequate countermeasure to targeted error induction attacks, highlighting the need to either address these attacks or qualify the security provided by this method (Chapter 5). This results suggests that either our models should be addressed (this could be expensive!) or more flexible models proposed, relative to which the dual-rail encoding techniques can be proved secure. In either case, a principled approach is required.

Ultimately, we feel that a theoretical approach will be integral in the advancement of research in secure computation in semi-private environments. The details of implementations and the practicality of various attack techniques will vary, but a well-developed theoretical foundation will adapt; results that are being derived in the next few years will likely remain relevant well into the future, continuing to provide insight and guide developments.

Bibliography

[1] Sergei P. Skorobogatov and Ross Anderson, "Optical Fault Induction Attacks," in *CHES 2002*, Springer-Verlag, pp. 2-12, 2002

[2] Simon Moore, Ross Anderson, Paul Cunningham, Robert Mullins, and George Taylor, "Improving Smart Card Security using Self-timed Circuits," in *ASYNC 2002*, IEEE, 2002

[3] Simon Moore, Ross Anderson, Robert Mullins, George Taylor, and Jacques Fournier, "Balanced Self-Checking Asynchronous Logic for Smart Card Applications," in *Microprocessors and Microsystems Journal*, 2003

[4] Marc Joye, Arjen K. Lenstra, and Jean-Jacques Quisquater, "Attacks on systems using Chinese remaindering," *Journal of Cryptology* 12(4):241-245, 1999

[5] Eli Biham, Adi Shamir, *Personal Communication.*

[6] Mathieu Ciet and Marc Joye, "Elliptic curve cryptosystems in the presence of permanent and transient faults," in *Designs Codes and Cryptography*, 2003

[7] Johannes Blömer, Jean-Pierre Seifert, "Fault Based Cryptanalysis of the Advanced Encryption Standard (AES)," *Financial Cryptography 2003*, pp. 162-181, 2003

[8] Dan Boneh, Richard A. DeMillo, Richard J. Lipton, "On the Importance of Eliminating Errors in Cryptographic Computations," *Journal of Cryptology*, Vol. 14, No. 2, pp. 101-119, 2001

[9] Sung-Ming Yen, Marc Joye, "Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis," *IEEE Transactions on Computers*, Vol. 49, No. 9, pp. 967-970, 2000

[10] Paul Kocher, Joshua Jaffe, and Benjamin Jun, "Differential Power Analysis," in proceedings of *Advances in Cryptology—CRYPTO '99*, Springer-Verlag, pp. 388-397, 1999.

[11] Thomas Messerges, "Using Second-Order Power Analysis to Attack DPA Resistant Software," *Lecture Notes in Computer Science*, 1965:238-251, 2001.

[12] Jean-Sébastien Coron and Louis Goubin, "On Boolean and Arithmetic Masking against Differential Power Analysis", in *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems*, Springer-Verlag, August 2000.

[13] Dan Boneh and David Brumley, "Remote Timing Attacks are Practical", in *Proceedings of the 12th USENIX Security Symposium*, August 2003.

[14] Thomas S. Messerges, "Securing the AES Finalists Against Power Analysis Attacks," in *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems*, Springer-Verlag, pp. 144-157, 1999.

[15] Yuval Ishai, Amit Sahai, and David Wagner, "Private Circuits: Securing Hardware against Probing Attacks," in *Proceedings of CRYPTO 2003.*, 2003

[16] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, Tal Rabin, "Algorithmic Tamper-Proof (ATP) Security: Theoretical Foundations for Security against Hardware Tampering," in *Theory of Cryptography Conference,* 2004.

[17] Silvio Micali, Leonid Reyzin, "Physically Observable Cryptography," in *Theory of Cryptography Conference,* 2004.

[18] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi, "Towards Approaches to Counteract Power-Analysis Attacks," in *The Proceedings of CRYPTO '99,* Springer Verlag, LNCS 1666.