Fault Attacks on Dual-Rail Encoded Systems

Jason Waddle and David Wagner
Department of Computer Science
University of California, Berkeley
jwaddle@google.com, daw@cs.berkeley.edu

Abstract

Fault induction attacks are a serious concern for designers of secure embedded systems. An ideal solution would be a generic circuit transformation that would produce circuits that are robust against fault induction attacks. We develop a framework for analyzing the security of systems against single fault attacks and apply it to a recent proposed method (dual-rail encoding) for generically securing circuits against single fault attacks. Ultimately, we find that the method does not hold up under our threat models: n-bit cryptographic keys can be extracted from the device with roughly n trials. We conclude that secure designs should incorporate explicit countermeasures to either directly address or attempt to invalidate our threat models.

1 Introduction

Securing embedded systems is exceedingly difficult due to the fact that potential adversaries have physical access. In this paper, we focus on fault induction attacks. Recently, dual-rail logic, a scheme previously used in asynchronous circuit designs, has been suggested as a countermeasure for fault induction attacks [2, 3]. In this article, we analyze, in a variety of threat models, the effectiveness of dual-rail encoding as a countermeasure against single fault induction attacks.

1.1 Summary of results

In Section 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 1 summarizes the effectiveness of our attacks in the various threat models.

Table 1. Summary of our attacks for extracting an n-bit key from a dual-rail encoded system.

Fault	Fault	Required Trials	
Type	Persistence	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

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.

1.2 Overview of sequel

We present the necessary background to understand our results in Section 2. In Section 3 and Section 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 conclude and suggest some directions for future research into securing embedded systems against fault induction attacks in Section 5.

Finally, to justify our threat models and help convince the reader of the practicality of fault induction attacks, Appendix A discusses some of the physics behind optical fault induction and gives an example with the CMOS digital logic family.

2 Background

2.1 Attacks

Securing embedded systems is complicated by the fact that an adversary may have unfettered physical access to the target system. As a result, several types of attacks that are impractical without physical access have emerged as primary considerations in the design of secure embedded systems.

2.1.1 Side-channel attacks

Side-channel attacks are typically *passive*: an adversary usually just observes the target system under normal operation. What the attacks exploit is the presence of a *side-channel*, an unintentional source of information about the internal operation of the target system.

Power analysis, the observation of the power consumption of a target system, has emerged as one of the the most effective, practical, and consequently well-studied type of side-channel attack. Other types of side-channels have been successfully exploited, as well; examples include electromagnetic emission analysis and timing analysis.

2.1.2 Fault attacks

Unlike side-channel attacks, fault attacks are fundamentally *active*: these attacks require the induction of some sort of fault during the operation of the target system.

Intuitively, it seems like it might be difficult to extract useful information from a target system by inducing faults. However, several fault attacks have proved surprisingly effective and practical, requiring relatively little technical sophistication of the attacker [8]. An instructive example is the fault attack on RSA decryption using the Chinese Remainder Theorem: an attacker can recover the target system's entire key by observing only one faulty computation [4]. Fault attacks have applied to other systems, including elliptic curve systems [6] and AES [7].

Skorobogatov and Anderson describe optical fault attacks [1], a precise and practical method for exercising fine-grained control over digital logic. This suggests that some very powerful threat models may be realistic, motivating the consideration of attacks that require much more finesse than the RSA-CRT attack.

We focus on these types of precision attacks against systems that employ dual-rail encoding as a defense against fault induction.

2.2 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 2 summarizes the encoding.

Table 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)

2.2.1 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 $(x_1, x_0) = (1, 0)$.

2.2.2 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 (they can be balanced and they detect single bit errors). For these reasons they have been suggested for use in secure embedded systems [2, 3].

2.2.3 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 each high-level logic gate is realized by a collection of OR gates and C-elements (explained below) that compute the appropriate logic function in dual-rail encoding. Figure 1 gives an example transformation of a logical OR gate into its dual-rail implementation.

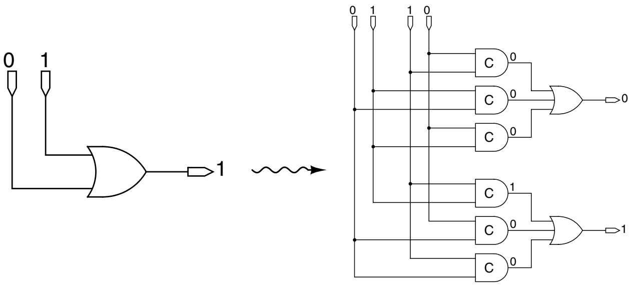

Figure 1. A logical OR gate and a dual-rail implementation.

The dual-rail implementation of a logical AND gate is similar: one is depicted in Figure 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 (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 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.

2.2.4 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.

2.2.5 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.

2.2.6 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 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].

2.2.7 Robustness against fault attacks

Dual-rail encoding is an instance of *m-of-n encoding*. In particular, "dual-rail logic" is another name for 1-of-2 encoding. 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.

On the other hand, an error indication may be all an attacker needs. As we will show, it is apparently important whether errors are *corrected* or just *detected*. If an attacker can observe whether carefully induced faults result in valid or erroneous computations, he can deduce some information on internal values (e.g., cryptographic key bits).

3 Models

In this section, we discuss both our models for the systems being analyzed and the classes of induced faults we will be considering.

3.1 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 2 along with the simple wire-swap for logical NOTs.

3.1.1 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.

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.

3.1.2 The secret

We assume that the systems we consider contain some sort of secret key that is used in computation. The motivating situation is where the secret is a key to a block cipher: the device takes the plaintext as input, performs encryption using the secret key, and outputs the resulting ciphertext. 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.

3.2 Threat models

We are concerned with the behavior of systems when subjected to single arbitrary faults. In practice, these faults are induced in the transistors at the physical level, but we model them as occurring in the OR and C-element gates of the dual-rail implementations of the logical AND and OR gates of the original circuit. In Appendix A, we discuss how faults induced in transistors motivate our threat models.

For our purposes here, we do not consider fault attacks on the bus to external RAM or to the instruction memory.

3.2.1 Effectiveness of attacks

The canonical task of the attacker is to extract a fixed n-bit secret from a device by repeatedly using the device to compute on known inputs and optionally inducing single faults during each computation. Our primary measure of the efficiency of an attack is the number of iterations required to deduce the target system's secret as a function of n.

3.2.2 Faults and assumptions

We think of an induced fault in a circuit as a possible deviation in the value at the output or input of a single OR or C-element gate in the dual-rail implementations of the AND or OR logic gates.[1]

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. We assume the adversary has complete knowledge of the layout and operation of the system (with the obvious exception of the value of the secret, however it is stored). These assumptions are admittedly strong, but recent results in fault induction indicate that at least the precision assumptions are not outlandish [1].

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.

3.2.3 Fault type

The type of fault indicates what kind of deviation the fault induces at its target. We consider three types: *set* and *reset* faults set the target value to 1 or 0, respectively, while a *flip* fault toggles the target value.

3.2.4 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, causing it to swing between the fault and natural states. We assume only that fault state does occur at least once.
- A *steady* fault forces the target to assume the fault state for the entire interval during which the attacker applies the steady fault.

[1] 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.

4 Attacks on dual-rail encoded systems

We describe how to efficiently extract secret bits in our various threat models.

In all cases, the attacker learns a particular 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. Inducing a fault in a well-chosen OR or C-element gate reveals something about the bit in question. The same approach is repeated for each of the n bits.

Figure 2 shows the encoding-level implementation of both an AND and an OR gate for dual-rail logic and indicates the sites relevant to our attacks.

4.1 Set and flip faults

We show how an attacker who can induce set or flip faults can recover the secret. The trick is to cause data-dependent activation of the C-elements intended for propagating alarm signals, ultimately revealing a logical input value in whether or not the circuit terminates in error. Due to the stateful, sticky nature of the C-elements, steady and transient flip- and set-faults can be employed in the same way to reveal the secret bit.

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

4.1.1 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 fault at the C-element input marked Sa in Figure 2 for that type of gate, while for a b input she would induce a fault at Sb.

Suppose the attacker is interested in the a input of a dual-rail logical AND gate (the b and OR gates are similar), and that both the a and b inputs contain valid data: each is in either a logical 0 (dual-rail encoded $(0, 1)$) or a logical 1 (dual-rail encoded $(1, 0)$) state.[2] In this case, the suggested attack is to set to 1 the Sa input to the C-element, as indicated in Figure 2.

Let us consider the resulting output of the dual-rail AND gate. There are two possibilities, depending on the logical value of a:

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

Table 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)	C-elt In	(r_1, r_0)	Result
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)	(1,0)	(1,0)	no error

  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, which propagates, and the computation finishes in an error state.

The cases are also summarized in Table 3. Notice that the computation halts with an error if and only if the logical value of a was originally 0, so the value of a (and bit i) is revealed. Repeating this attack once for each secret bit allows the attacker to learn the entire n-bit secret with n iterations.

4.1.2 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 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 makes

[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.

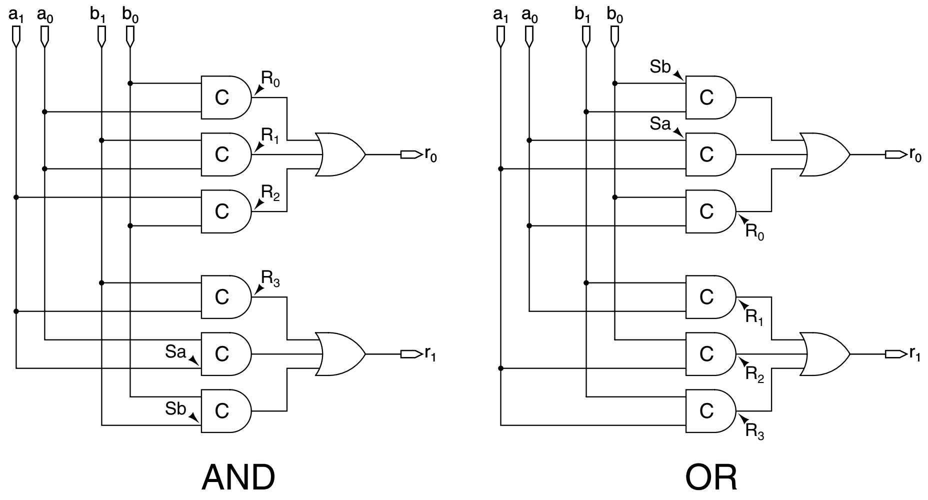

Figure 2. Dual-rail AND and OR gates. Set/Flip attack target sites are labeled Sa and Sb. Reset attack target sites are R_0 through R_3.

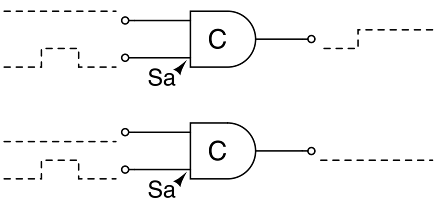

Figure 3. Depending on the value of the other input, a C-element may latch high from a transient signal at the Sa input.

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, and is depicted in Figure 3.

4.1.3 Steady flip model and power consumption

In general, the difficulty with a single steady flip is that it will always induce one of the two types of error states, quiet or alarm. Casual observation of the device will not reveal which error state resulted from the induced fault, but an attacker that also observes power consumption may well be able to distinguish quiet and alarm states, and thereby determine the target logic value.

4.1.4 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 with our assumptions, dual-rail logic is not secure against steady or transient set- and flip-faults.

4.2 Steady reset fault model

As we saw in Section 4.1, 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.

In the reset fault models, however, the induced faults can only change a 1 to a 0, and this presents 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 other inputs to 1 will never cause the output to go to 0. Likewise, clearing input bits of a gate 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

[3] This limitation is mostly an artifact of our chosen model. Physical

Table 4. Outputs from Figure 2, minterms, and implications.

Active Output	Active Minterm	Logical Implication
R_0	$a_0 b_0$	$a = b = 0$
R_1	$a_0 b_1$	$(a = 0) \wedge (b = 1)$
R_2	$a_1 b_0$	$(a = 1) \wedge (b = 0)$
R_3	$a_1 b_1$	$a = b = 1$

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, which is fed as an input to a dual-rail AND gate, as depicted in Figure 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 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.

Suppose the attacker guesses which of the the $R_0, \ldots, R_3$ outputs is 1 and then clears that output. If he guesses incorrectly, the fault will have no effect because that bit is already fixed at 0. If he guesses correctly, however, 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 clearing each of the R_i sites across several computations and observing whether each computation deadlocks or not, the attacker can learn the logical values carried on both inputs to any logical 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 effective 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. However, if the attacker attacks gates with two unknown inputs by randomly selecting the order of attack on the $R_0, \ldots, R_3$ sites, he can expect to run fewer tests. Let X_i stand for the event that the active site is known after i trials, and we compute:

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

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]

4.2.1 Reset fault model summary

We have shown 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 Conclusion and Recommendations

We have proposed some threat models and analyzed the efficacy of dual-rail logic as a countermeasure against a variety of types of induced faults. Since we have found efficient attacks against dual-rail encoded systems, we must conclude that dual-rail encoding is not a sufficient countermeasure against single fault attacks under the assumptions of our models.

5.1 Recommendations within our models

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 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

and implementation considerations suggest that the ability to induce reset faults implies the ability to induce set faults. Appendix A provides a more detailed discussion.

[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.

rather than divulging an incorrect output that might give the attacker enormous information [8]. Unfortunately, our attacks and others [9] successful exploit this error state 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 3, single fault errors in a logic value's representation could be *corrected* rather than just *detected*. Thus, single induced 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.

5.2 Recommendations beyond our models

Perhaps our threat models are too strong. Due to uncertainty about the target system and limitations of equipment, attackers may not be able to cause precise single bit errors. If we could develop models that accurately captured these limitations, we should be able to argue rigorously that approaches like dual-rail encoding *are* secure. These results would allow us to trust these designs as long as the attacker limitation assumptions seemed realistic with respect to the current state of technology.

References

[1] Sergei P. Skorobogatov and Ross Anderson, "Optical Fault Induction Attacks," *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," *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," *Microprocessors and Microsystems*, 27(9):421–430, 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," *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*, 14(2):101–119, 2001.

[9] Sung-Ming Yen, Marc Joye, "Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis," *IEEE Transactions on Computers*, 49(9):967–970, 2000.

A Physical motivation for attack models

The powerful yet surprisingly practical optical fault induction attacks of Skorobogatov and Anderson [1] provide the major motivation for our threat models. While previous work will convince the reader that fault induction 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 induction 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 induction 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.

A.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 potential difference between the the electrically-isolated metal[5] 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 gives commonly-used schematic representations of MOSFETs.

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

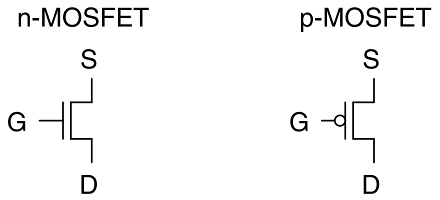

Figure 4. Common schematic representations of MOSFETs. G = Gate, S = Source, and D = Drain.

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.

A.1.1 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 nonconductive 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 *p-n junction* (a junction between p-type and n-type materials) only when the n-type material is negatively charged relative to the p-material. For example, a diode, which permits current flow in only one direction, is nothing more than a p-n junction.

A.1.2 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. Figure 5 depicts 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 5 to conduct, a positive charge is loaded onto the gate. Although the gate is electrically isolated from the bulk by a layer of nonconducting 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 as 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, with electrons being the majority charge carrier.

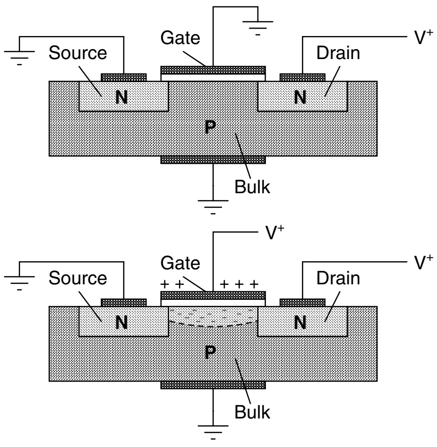

Figure 5. An n-channel enhancement-mode MOSFET in normal operation.

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.

A.2 Optically induced failure in n-channel MOSFETs

Loading the gate with a positive charge to induce an electric field is not the only way to create a channel of majority charge carrier electrons in the p-type material.

A.2.1 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

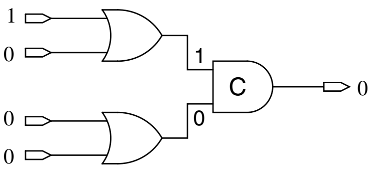

Figure 6. The logical schematic of our target circuit.

photon is determined by the amount of energy lost by the electron: the more energy lost, the smaller 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 desired wavelength of light.

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.

A.3 An optically induced fault in CMOS

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

Skorobogatov and Anderson have successfully used light to set the value of bits in SRAM memory (memory based on CMOS digital logic) [1]. For our example, however, we would like to use the same technique to influence the output of the circuit depicted in Figure 6.

In Figure 6, 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 6.

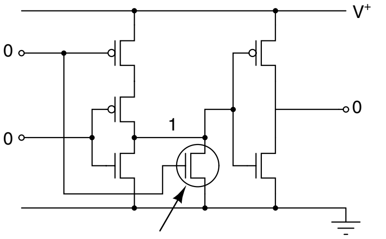

Figure 7. The CMOS schematic of the target OR gate. The n-channel MOSFET indicated by the arrow and circle is the target of our optical attack.

Figure 7 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 7, 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 may 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 6), ultimately causing the target circuit to output 1.

A.4 Practical considerations

In order to optically flip bits of SRAM, Skorobogatov and Anderson 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 [1]. 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.

Our example in the previous section demonstrates something close to our transient clear fault model, but it is not obvious whether the other fault models, the steady and flip varieties, are applicable to CMOS. On the other hand, a design that is secure in all of our fault models would be safe against single faults in a CMOS implementation.