UNIVERSITY OF CALIFORNIA SAN DIEGO

**Statistical Metrics of Hardware Security**

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Alric Althoff

Committee in charge:

Professor Ryan Kastner, Chair
Professor Farinaz Koshanfar
Professor Lawrence Saul
Professor Aaron Shalev
Professor Timothy Sherwood

2019

The dissertation of Alric Althoff is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California San Diego

2019

# DEDICATION

To my darling, Sarah, my companion.

# EPIGRAPH

*This is how he grows: defeat by greater and greater things.*

— Rainer Maria Rilke, Der Schauende

*Experience is something you don't get until just after you need it.*

— Steven Wright

*Don't worry about people stealing your ideas.*

*If your ideas are any good, you'll have to ram them down people's throats.*

— Howard Aiken

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

# ACKNOWLEDGEMENTS

It is not possible for me to list all of the people who've influenced me throughout my life. However, reader, know that if we have spoken for a suitable length of time then you belong on that unmakeable list. Any endeavor is a collage of souls. You on the list, welcome! Welcome to a new collage! Here we toast to a day when we can all trust one another.

And now for the formal part.

Chapter 2, in part, is a reprint of the material as it appears in the Proceedings of the 54th Annual Design Automation Conference (DAC), June 2017. Althoff, Alric; Kastner, Ryan, IEEE 2017. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in part, is currently being prepared for submission for publication of the material. Althoff, Alric; Blackstone, Jeremy; Kastner, Ryan. The dissertation author was the primary investigator and author of this material.

Chapter 4, in part, is a reprint of the material as it appears in the Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA), June 2018. Althoff, Alric; McMahan, Joseph; Vega, Luis; Davidson, Scott; Sherwood, Timothy; Taylor, Michael; Kastner, Ryan, ACM 2018. The dissertation author was the primary investigator and author of this paper.

## VITA

| | |
|---|---|
| 2013 | Bachelor of Science, Cognitive Science with Specialization in Computation<br>University of California, San Diego |
| 2013 | Bachelor of Science, Mathematics – Computer Science<br>University of California, San Diego |
| 2017 | Master of Science, Computer Science<br>University of California, San Diego |
| 2019 | Doctor of Philosophy, Computer Science<br>University of California, San Diego |

## PUBLICATIONS

"Holistic Power Side-Channel Leakage Assessment: Towards a Robust Multidimensional Metric", Alric Althoff, Jeremy Blackstone, Ryan Kastner, (in preparation)

"Synthesizable Higher-Order Functions in C++", Dustin Richmond, Alric Althoff, Ryan Kastner, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, September 2018

"Hiding Intermittent Information Leakage with Architectural Support for Blinking", Alric Althoff, Joseph McMahan, Luis Vega, Scott Davidson, Timothy Sherwood, Michael Taylor, Ryan Kastner, *International Symposium on Computer Architecture (ISCA)* , June 2018

"A Streaming Clustering Approach Using a Heterogeneous System for Big Data Analysis", Dajung Lee, Alric Althoff, Dustin Richmond, and Ryan Kastner, *International Conference on Computer-Aided Design (ICCAD)*, November 2017

"Quantitative Analysis of Timing Channel Security in Cryptographic Hardware Design", Baolei Mao, Wei Hu, Alric Althoff, Janarbek Matai, Yu Tai, Dejun Mu, Timothy Sherwood, and Ryan Kastner, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, November 2017

"An Architecture for Learning Stream Distributions with Application to RNG Testing", Alric Althoff and Ryan Kastner, *Design Automation Conference (DAC)*, June 2017

"Towards Property Driven Hardware Security", Wei Hu, Alric Althoff, Armaiti Ardeshiricham, and Ryan Kastner, *Microprocessor Test and Verification Conference (MTV)*, December 2016 – Invited Paper

"Spector: An OpenCL FPGA Benchmark Suite", Quentin Gautier, Alric Althoff, Pingfan Meng, and Ryan Kastner, *International Conference on Field-Programmable Technology (FPT)*, December 2016

"Adaptive Threshold Non-Pareto Elimination: Re-thinking Machine Learning for System Level Design Space Exploration on FPGAs", Pingfan Meng, Alric Althoff, Quentin Gautier, and Ryan Kastner, *Design Automation and Test in Europe (DATE)*, March 2016

"Composable, Parameterizable Templates for High Level Synthesis", Janarbek Matai, Dajung Lee, Alric Althoff, and Ryan Kastner, *Design Automation and Test in Europe (DATE)*, March 2016

"Quantifying Hardware Security Using Joint Information Flow Analysis", Ryan Kastner, Wei Hu, and Alric Althoff, *Design Automation and Test in Europe (DATE)*, March 2016 – Invited Paper

"Quantifying Timing-Based Information Flow in Cryptographic Hardware", Baolei Mao, Wei Hu, Alric Althoff, Janarbek Matai, Jason Oberg, Dejun Mu, Timothy Sherwood, and Ryan Kastner, *International Conference on Computer-Aided Design (ICCAD)*, November 2015

"A Scalable FPGA Architecture for Nonnegative Least Squares Problems", Alric Althoff and Ryan Kastner, *International Conference on Field Programmable Logic and Applications (FPL)*, September 2015

"A Brain–Computer Interface (BCI) for the Detection of Mine-Like Objects in Sidescan Sonar Imagery", Christopher Barngrover, Alric Althoff, Paul DeGuzman, and Ryan Kastner, *IEEE Journal of Oceanic Engineering*, March 2015

ABSTRACT OF THE DISSERTATION

**Statistical Metrics of Hardware Security**

by

Alric Althoff

Doctor of Philosophy in Computer Science

University of California San Diego, 2019

Professor Ryan Kastner, Chair

Hardware security is a fundamental and increasingly important contributor to the trustworthiness of our computing infrastructure. With the proliferation of embedded computers, attacks focused on exploiting hardware vulnerabilities are being discovered rapidly, and mitigation techniques are being proposed and deployed. But how do we determine the effectiveness of a mitigation against attacks that may not yet exist? Methods that answer this question are called *security metrics*. In situations where an attack relies on either disruption or interception of non-deterministic values, these security metrics are based on statistics.

Statistical metrics are based on assumptions about the capabilities of potential attackers and the nature of the measurements. Metrics with assumptions that are too strong can be a source of false confidence, because a favorable metric result does not necessarily indicate a secure device. In this dissertation, I formulate practical metrics in the context of power side-channel analysis and continuous testing of random number generators that do away with many of these limiting assumptions.

This dissertation has three major facets. In the first, I present a hardware system and statistic for continuous testing of random number pipelines which can be used irrespective of the underlying probability distribution and requires minimal a priori knowledge about the system to be monitored. The second, the Holistic Assessment Criterion, is a statistic that allows ranking of devices and algorithms with respect to the entire measurement window, and is sensitive to vulnerabilities in the underlying high-dimensional geometry of the measurement vectors. The third Computational Blinking, is an architectural mitigation strategy relying on a metric that ranks times during execution of an algorithm in order of vulnerability.

# Chapter 1

# Introduction

## 1.1 Motivation

Today, more than ever before, we store and communicate important information using computer systems. We entrust these systems with our personal safety and the integrity of our belongings and social identity. The emergence and commercial popularity of mobile and embedded technologies, such as mobile phones, home and office security and surveillance systems, and numerous "smart home" appliances all demonstrate this trust [2, 3, 4]. These objects that had previously been purely mechanical or electrically powered have been integrated with wireless communications and microprocessors. However, because of the spread of embedded devices into our physical environment, opportunities for exploitation of this trust have multiplied.

Hardware security is fundamental to the overall system security of all computers, and embedded devices in particular, given that the required access is more likely. Additionally, attacks relying on vulnerabilities in hardware primitives are being discovered rapidly—[5, 6, 7, 8, 9, 10] are just a few examples from the past year. Once discovered, these types of vulnerabilities are difficult to mitigate, often requiring hardware modifications that are expen-

sive and error prone to implement, test, and deploy. For example, an early microcode patch issued by the chip manufacturer Intel to mitigate the Spectre [5] variant two branch target injection attack caused system instability and potential data corruption [11].

There are several difficulties that we must grapple with when we acknowledge this fact. Principally, how do we know whether a mitigation technique or run-time vulnerability detection mechanism is effective *in general*? In other words, for a mitigation or detection strategy that has either been proposed or implemented, how effective will it be? Additionally, at what points is the mitigation strategy most/least effective? How does it compare with other strategies with respect to security? There are myriad schemes that attempt to answer these questions and measure the susceptibility of devices to these active or passive hardware attacks. In general these approaches to vulnerability quantification are called "security metrics".

Security metrics come in many forms, and differ with respect to the type of vulnerability they are designed to detect and how they quantify security. Some metrics are designed to give an indicator of vulnerability without any ranking in order of vulnerability/security implied, e.g. [12] returns a Boolean indicator of information isolation between hardware components, while [13] validates access control policies. Others provide numerical ranks with great/lesser values for systems that are more vulnerable to specific attack vectors. Often these either measure the size of the attack surface [14, 15], or the number of trials required on average for attack success [16, 17]. While some of these seek to formally guarantee a lack of vulnerability to specific forms of information leakage, others use statistics that allow estimates of security based on measurements during device operation. Metrics of this type are often used to judge resistance to power side-channel attacks [18, 19, 20]. While they may differ in method, any effective security metric tests for certain qualities correlated with the success of some, or many, forms of attack.

In this dissertation, it is the latter sort of metric that I will discuss: those that use measurements of a system, and provide a quantity that can be used to determine how vulnerable

a system is relative to either an absolute baseline or the results of other tests. Additionally, though not all statistical metrics use measurements taken at the hardware level, it is at this level that I will apply the statistics I introduce in this document. To be clear, this is not a *limitation* of these methods, inasmuch as a design choice due to the potential vulnerabilities of software[1].

The most straightforward statistical tests of this type are known attacks against the hardware device under test (DUT). Attack success clearly indicates vulnerability, and the difficulty, or *complexity*, in terms of number of measurements or attack trials required for a successful attack can be used to provide a numeric ranking. Unfortunately, the pace of development for new attacks traps us in a loop—hardware security engineers simply cannot keep up with the pace of attack discovery. Other metrics overcome this limitation by using statistics to indicate the existence of attackable properties. However, these metrics have various limitations, such as not allowing testing for variable interactions of numerous measurements, requiring a probability distribution of a random value to be known a priori, or ignoring the statistical assumptions of a particular test. I will discuss each of these in more detail later in this dissertation.

The important point to make at this time that that these limitations manifest in coverage gaps—their assumptions about the attacker's limitations are too strict to provide a strong statement about the security of the device. As an analogy, consider a house with two doors, front and back. Testing for an engaged lock on the front door is not sufficient to ensure that no one may enter the house without breaking any locks. To do that we must test the back door as well. While this analogy is very rough, it makes the point; where the two doors form the *attack surface*, the front-door-only metric has a *coverage gap* at the back door. Unfortunately, the attack surface of a computer system is generally much less obvious, and the coverage gaps of a metric are less clear. For example, a statistical metric may make strong implicit assump-

[1]It is also worth mentioning that I follow the convention that "hardware" is the physical, and generally static, portion of a computer system, while software is mutable.

tions about the capabilities of the attacker—such as *where* or *when* during the algorithmic pipeline an attack may be executed—that do not realistically reflect an attacker's limitations. Concretely, [7] shows that a correct implementation of AES-128 that uses the rotating S-Box masking (RSM) power side-channel mitigation scheme can be broken—the entire key can be recovered—using a *single power trace* after a profiling phase[2] with a total of 80,000 traces. The RSM approach has been shown to be secure under certain assumptions [21]. So what went wrong? The problem is that while the technique protects against attacks against pairs of points in time during execution, the attacker has no such concerns or limitations. It is just as easy to attack a device using a large number of points as it is to attack a single time point. Note that there are often limitations to total coverage of the attack surface, but we can make practical metrics that take more of the attack surface into account than the prior state of the art, such as multiple, or even *all*, points in time. A metric that can measure the entire temporal attack surface will detect that AES-128-RSM is vulnerable to attack, and we introduce such a metric in Chapter 3.

Creating metrics without these coverage loopholes in is not trivial. For run-time tests, we must design new hardware modules. For pre-deployment tests, I argue that we should take a different perspective on statistical testing, and develop a theoretically sound basis for the metrics themselves. This theoretical basis must address the underlying mathematical issues, though it does not, and in some cases cannot, identify a specific attack vector. This is a natural result of the attack-independent qualities of sufficiently general metrics. I take the perspective that the "patch and pray" approach, in which vulnerabilities are fixed after a working attack is reported, while in some cases necessary, is not sufficient, and I develop these statistical metrics with this in mind.

The underlying philosophy that I take in this dissertation *is not* that we should create metrics that are suitable for discovering vulnerabilities to known attacks, but rather that we

[2]Note too that this profiling phase only needs to be done once for an entire class of similar hardware devices and this profile information can be easily transferred between attackers.

should develop metrics that cover a larger attack surface. Ideally, these would test for attack vectors that have not yet been discovered, whether or not the specific details of these attack vectors are revealed by the metric. Overall, the thesis of this dissertation is that **it is possible to close vulnerability detection loopholes using statistical metrics with few assumptions for power side-channel attacks, and random number generation pipelines.** To demonstrate, I develop several metrics with coverage improvements for both of these application domains, with a focus on removing assumptions about attack types. I also discuss mitigation and detection approaches that enable making use of these metrics after deployment.

## 1.2 Contributions

Overall, this work contributes to the field of hardware security metrics. The philosophical message of this work is that trustworthy security metrics should *cover the threat surface* as much as practically possible. We also argue that they must be attack-independent and statistically valid within known and respected limits.

To this end, and to demonstrate our philosophy in practice, this thesis introduces three metrics along with practical test methodologies and tools:

- **A metric for testing algorithms relying on random number pipelines at run-time.** To make this metric as useful as possible, we include several lightweight hardware implementations that have the additional benefit of being *trainable*—this they can be used to test random variables from *any* distribution, either known a priori, or unknown and only determined during run-time. Other randomness testing hardware makes assumptions about the probability distribution of the random values, and so has a coverage gap in that they cannot be applied to later stages of an algorithmic pipeline. Any flaw, or exploit, in later stages will go unnoticed by these less flexible test regimens.

- **A metric for testing vulnerability to power side-channel attacks of algorithms at**

**the device level.** We have taken steps to ensure this metric follows our philosophy by making no assumptions about the *type or location in time* of the vulnerability that it is are sensitive to. This also requires us to develop significant statistical and geometric intuition regarding what vulnerability to side-channel attacks looks like in a holistic sense. Existing metrics do not have this level of holistic coverage of all time points and their complementary interactions.

- **A metric for ranking the times during an algorithm's execution in order of vulnerability to power side-channel attacks.** This technique also takes the complementary interactions of differing time points into account. We also develop an associated programmable, hardware/software, architectural mitigation strategy for power side-channel attacks that can be applied to general purpose processors. Additionally, the metric, tools, and methods we introduce can be used to help security engineers find vulnerabilities when applied to algorithms prior to deployment.

## 1.3 Outline of this Dissertation

The overarching theme of this dissertation is statistical security metrics with increased detection coverage over the state of the art. Chapter 2 discusses testing of random number generation pipelines and introduces novel and very lightweight hardware and an associated statistical test for correlation and bias. Chapter 3 introduces a security metric that allows us to rank devices and algorithms holistically with respect to power side-channel leakage. Chapter 4 concludes with our work on computational blinking: an architectural mitigation strategy for power side-channels, that includes additional insights and tests of a new security metric that ranks leaky intervals over time.

Now, we discuss the motivation for each of these in more detail.

### 1.3.1 Quality Testing the Outputs of Random Number Pipelines

Cryptographic algorithms often depend on random initialization vectors or single-use nonces and are thus dependent on randomness—sequences of bits that are completely unpredictable—to maintain confidentiality of the secret keys. Additionally, many emerging applications in statistical computing and simulation also depend on a trustworthy source of random bits to function properly. Due to the critical nature of random number generator (RNG), U.S. government standards [22, 23, 24] are in place requiring continuous run-time health checks that, at least, ensure that subsequent blocks of generated bits are not equal to one another and, more recently, that test for a certain amount of entropy. These standards also mandate that health checks be in place to test startup integrity of the source of random bits. Many works develop tests tailored to these scenarios [1, 25, 26, 27, 28, 29], while other recent work in RNG quality testing focuses on implementing previous well-regarded tests [30] for *bit-wise*, Bernoulli $p = 1/2$, correctness and statistical independence between generated bits.

In Chapter 2, we present a modified algorithm, lightweight hardware architecture, and a hypothesis test that generalizes this work such that it can be used when testing for bias in random values drawn from any distribution. As a bonus, our test is also sensitive to dependence in terms of sample order. Our testing regimen is particularly suited to use-cases where many stages of post-processing are involved prior to final use of the random samples.

In terms of cryptographic techniques, the quality of the random or pseudorandom number generation routine is of utmost importance. Over the past several years many security flaws have been at least in part caused by bias in either the method used to generate initial random samples, or simple coding errors in the interpretation of random values. For example, as discussed in [31], a fencepost error created by using $\leq$ instead of $<$ significantly compromised the security of the Cryptocat chat application, and such errors are easily made. While for off-line error checking it is often feasible to use a technique such as an Anderson-Darling test [32] along with time-lagged cross-correlation, a straightforward hardware implementation

would add considerable area to designs that may already be resource-starved. The approach we introduce in Chapter 2 addresses these concerns, and can help mitigate attacks that rely on manipulating the environmental or algorithmic vulnerability surface involved in random and pseudorandom number generation.

The dangers of attacks or errors involving random number generation routines are not limited to cryptographic scenarios, and neither is our testing hardware. Consider a hypothetical situation where one or several investing institutions use Monte Carlo simulation routines—which often require random samples from specific non-uniform distributions—in financial models. It is a well known result from the theory of dynamical systems that small periodic perturbations at well-placed intervals can cause a system previously thought to be stable to diverge dramatically [33]. If a hardware/software flaw or attacker were to introduce a subtle bias in the random number generation routine at the right times, the model could be made to behave either erratically. Even more nefarious attacks can manipulate the algorithm in precisely predictable ways that still have the appearance of validity. Testing for bias as I propose—as late as practically possible in the algorithmic pipeline—will mitigate such issues.

### 1.3.2 Holistic Side-Channel Leakage Assessment

Providing secrecy is challenging when an attacker has physical access to the device. In cases where a hardware device is in the possession of the attacker, they likely have the ability to measure the dynamic power usage of the chip, which frequently leaks useful information about any secret bits used during computation.

Attacks on cryptographic hardware using so-called *side-channel information* are often fast and simple to execute, and these side-channel attacks (SCAs) are well-studied across both industry and academia. Conversely, mitigation strategies are expensive and time consuming to implement since they rely on specialized hardware design techniques or complex algorithmic modifications [34, 35]. Unfortunately, even after the implementation of a mitigation scheme,

it is difficult to assess its effectiveness and correctness. This has motivated NIST and ISO/IEC to solicit recommendations for side-channel leakage assessment (SCLA) metrics and propose standards such as ISO/IEC 17825:2016 [36, 37].

A straightforward method of determining the vulnerability level of a device is to perform side-channel analysis on each sample of measured power traces across differing keys with a fixed number of traces. In fact, this is exactly the approach taken by the ISO/IEC 17825:2016 standard for Security Assurance Levels 3 and 4 [36][3]. However, exploitable leakage may exist across more than a single sample and time points may be complementary. As a result, such unidimensional tests are not robust indicators of vulnerability. As such, these tests will in general give a false sense of confidence about the security of devices against multi-target, or higher-order, attacks that using multiple points in time [38, 39, 40, 7]. A robust SCLA metric should therefore consider all time points, that is to say, it should consider power traces *holistically*.

In Chapter 3, we discuss the limitations of current tests in more detail. We then list characteristics of a robust and holistic SCLA metric. Then, we develop an SCLA metric that addresses each of these factors. Our metric, the Holistic Assessment Criterion (HAC), is nonlinear, holistic, and assumption-free.

### 1.3.3 Measuring and Mitigating Power Side-Channel Information Leakage Over Time with Computational Blinking

During a side-channel attack one of the largest advantages that an attacker has is that they are free to collect an effectively unlimited number of traces in order to overcome noise. This means that attackers have essentially noise-free windows into the underlying changes in dynamic power use in order to infer activity at specific times. Luckily, not all times during algorithm execution are equally vulnerable to attack.

[3]Description freely available at [37]

To take advantage of this non-uniformity, in Chapter 4 I propose a mitigation strategy called *computational blinking*. This architectural approach provides controlled dynamic power consumption isolation through a mechanism to electrically disconnect general purpose computation from the rest of the system. This isolates all aspects of the computation from the power, ground, and other pins of the chip, and the rest of the on-chip functionality. While disconnected—during a blink—the attacker cannot learn anything of value from the computation. Thus repeated measurements provide no additional information about those disconnected regions of execution. In the same way that architectures have been extended with mechanisms to enable the software management of explicit and now implicit flows of information through special security modes [41, 42, 43, 44], this technique places the power side-channel under software control. This opens a spectrum of trade-offs. At one end of this spectrum execution is fully subject to power analysis, and at the other the entire secret operation is performed only in a series of disconnected states. This approach gives security engineers the ability to both make decisions within this trade-off space, and adapt to changes in the underlying algorithms.

However, the reality of such an architecture is that it can only manage to operate in a disconnected state for very short time windows. Unlike traditional security mode architectures, where complete and long-running computations can take place if necessary, within the blinking paradigm we must be judicious about what is covered by this mode. An important contribution of our work in Chapter 4 is the technique by which we take a set of traces and analyze them to uncover the points in time where the majority of information leakage is occurring.

# Chapter 2

# Random Number Quality Testing

## 2.1 Introduction

Computing systems are formed by multiple interacting layers of hardware and software with ascending levels of abstraction. In order for higher layers to be trustworthy, they must rely on components that are themselves secure. At the lowest levels of this hierarchy of abstraction we have security critical components, roots of trust, that are integral to the entire hierarchy. One of these roots of trust in modern processors is the hardware-based random bit generator (RBG) or *random number generator* (RNG). In this chapter, because we are focusing on post-processed values, we will use "RNG" to refer to both deterministic and true random number and bit generators—our work applies to either scenario.

In this chapter, I discuss a method that propagates the trustworthiness of the RNG, along with other sources of unpredictability closer to the point of use. In addition to a quality test integral to the RNG, it is important to have a quality testing system in place near the point of final use for mission critical applications, or systems that might be relied upon for high-impact decision making. Such systems could be, for example, physical simulations or prediction systems relying on generative models. We also devise a novel algorithm for dy-

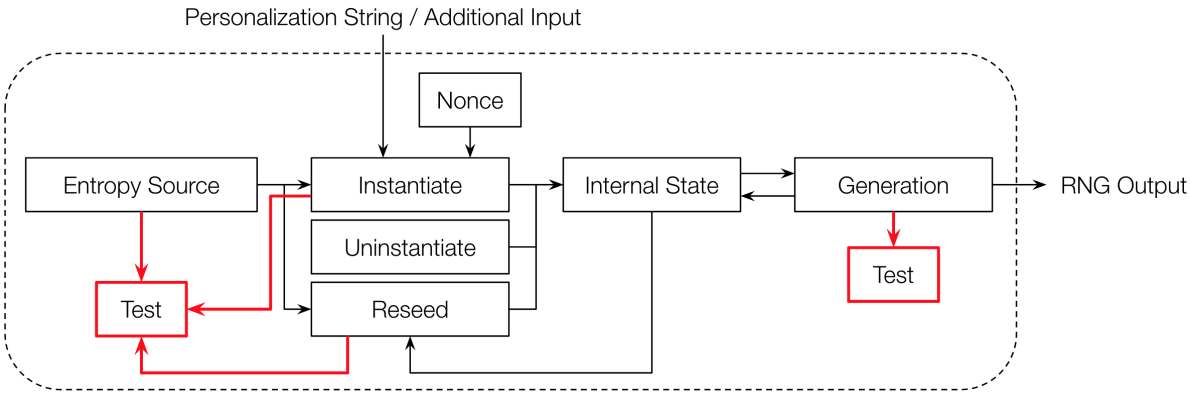


Figure 2.1: An example deterministic random number generation pipeline with health tests highlighted. If a health test fails, user intervention is required, or the device is replaced.

namic spacing of cumulative distribution function (CDF) estimation points that is designed to obtain higher precision in the tails of completely unknown distributions. In the end, we implement these in an efficient hardware architecture for quantile and CDF estimation that requires constant storage independent of the length of the stream of random values. We then combine these with a few statistical tests, thereby providing a flexible technique and architecture for discovering bias in random number generation pipelines regardless of source, expected distribution, and true vs. pseudorandomness.

### 2.1.1 The Structure of a Random Number Generator

In order to understand some of the available attack options against RNGs, we will describe the basic structure of a modern RNG. While there are many ways to build a random number generator there are a few standard components:

- **Entropy source(s).** The core component of any RNG is a source of uncertainty, or *entropy* [23]. This uncertainty can be derived from any unpredictable behavior that can be monitored by the system. Feasible sources include thermal variation, purpose-built oscillators, the amount of time taken between key-presses, mouse movement vectors, system times of RAM access, or a combination of on-board sensor outputs [24]. The

entropy source commonly mixes many such sources to help increase unpredictability.

- **A harvesting mechanism.** Most entropy sources do not output a uniformly random sequence of bits, and so a mechanism needs to be in place to transform the native format. Usually this is a sequence of XORs. XOR is useful as a sort of "statistical decoupler"; if any of the inputs to XOR are random, the output will be as well.

- **Generation/conditioning.** In order to increase throughput and make the distribution of bits as uniform (zero or one with equal probability) as possible, the high-entropy bits are provided to a conditioning, or generation, phase as seeds which "prime" this post-processing mechanism. It is quite common to use a cryptographic algorithm for this purpose, such as AES or triple-DES [45], in a recursive mode—combining its previous output with new seed inputs in a loop—for this step.

In addition to these functional components, the role of the system RNG as a root of trust necessitates that we know if it is behaving correctly, i.e. in a sufficiently uncertain manner. It is therefore required by the FIPS 140 standard to have health tests (also called built-in self tests (BISTs), or quality tests) in place at many stages of the RNG pipeline. For an example random number generator block diagram structured in accordance with the FIPS 140-2 [22] Annex C specification [23, 24], see Figure 2.1. Unfortunately, it is uncommon to implement RNG health tests beyond the RNG itself, even though many algorithms that consume these random numbers require several stages of processing after these values have left the RNG.

### 2.1.2 Biasing Attacks on Random Number Generators

Attacks on RNGs fall into a number of categories, with a feasibility decided by the nature of the RNG. If the seed of an RNG pipeline is static (i.e. a secret key) or deterministic, the system output is pseudorandom. Of the several ways that such a system can be compro-

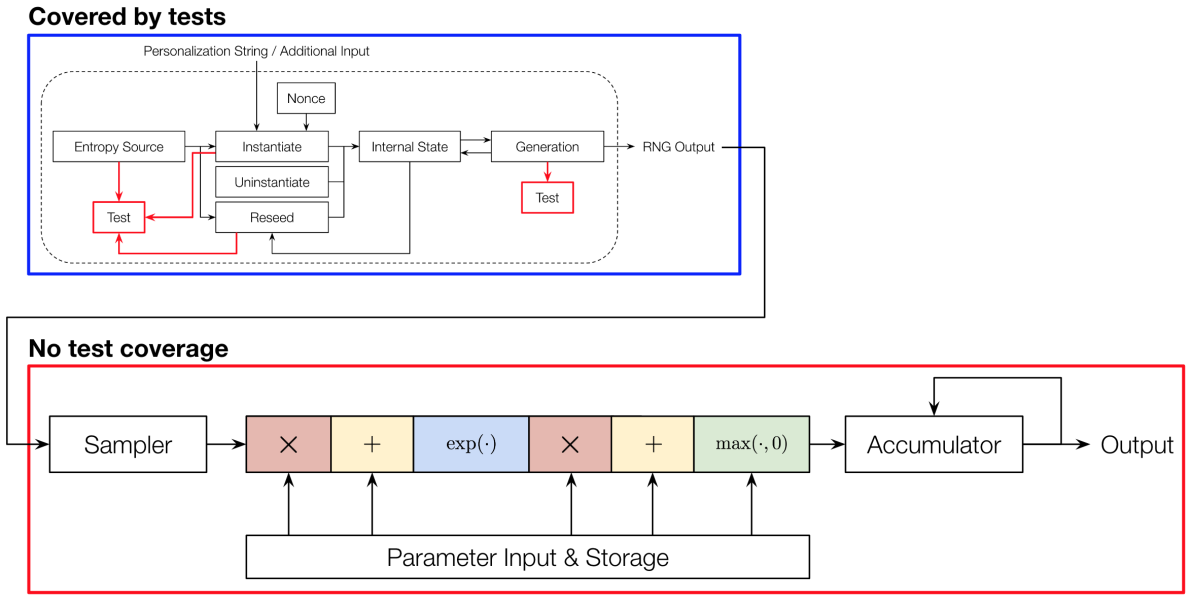


Figure 2.2: An example testing coverage gap. A Monte Carlo simulation of Black-Scholes European option pricing that has a large untested attack surface after the system RNG. This untested region may be exploited to change the output of the simulation in a way that is beneficial to an attacker.

mised, an attackers acquisition of this seed (using, for example, power side-channel analysis) is obviously the most damaging—this will allow the attacker to predict the sequence of bits completely [45]. Unfortunately, there may not be any measurable indication of whether an attacker has obtained the seed. This issue can be mitigated by seeding with unpredictable information. However, even in this case, there are attacks on the underlying sources of unpredictability that can reduce the entropy of their outputs significantly. That is to say, they can introduce *bias* which may increase the likelihood that a stream of random samples will contain values that an attacker expects, making the system more vulnerable.

Many sources of unpredictability are possible, for example, optical or radioactive sources [46, 47]. However, to ensure that the opportunity for observation by potential attackers is minimized it is best that these sources be located on-chip. Common sources of on-chip entropy are chaotic analog/digital systems and electronic noise [48]. Biasing attacks on these mechanisms can be accomplished by injection of electromagnetic interference, and in [49, 50]

this is demonstrated for ring oscillator-based entropy sources. The potential for a successful attack of this type is exacerbated by fact that while health tests are designed to detect these cases of bias due to either attack or other failure in the RNG, they are not always used, either due to provable security and resistance [51, 52] or customized RNG implementations which bypass the health testing system.

Even if the RNG implementation is otherwise secure, it is entirely possible for these attacks, among other errors, to exist later in the RNG pipeline. As an example application, assume that a trusted uniform and independent bit-generating random number generator outputs samples for use in a Monte Carlo simulation of a Black-Scholes pricing simulator for European options (see Figure 2.2). Testing the output of the RNG remains useful in this case, but the error and/or attack surface also includes the implementation of the initial sampling algorithm for converting bits into Gaussian random floating point numbers along with all other post-processing stages. Without a testing framework in place immediately prior to the consumer of the random numbers such an error could go undetected for a long time. In this example application, an attacker could alter the input parameters or operations anywhere in the unprotected region to manipulate the output of the pricing simulator, and thus potentially gain significant financial advantages over time through subtle manipulation of the simulation. This example is hypothetical, but we believe it demonstrates the importance of continuous testing for high-value targets.

### 2.1.3 Related Algorithms

The streaming component of our testing algorithms and hardware is a stochastic approximation technique for computing the $k$th smallest element in a list following a known, or learned, distribution. The statistical test component checks that the $k$th smallest element, also called the $k$th order statistic, is sufficiently close the empirical value. While the concept is simple, the algorithms are not obvious for doing this *without* storing and sorting the entire stream

of random values, nor are the statistical tests. While there is quite a bit of previous work on this problem in the literature, other algorithms focus on estimation of a stream's order statistics that have actually been observed in the stream. Our work is based on an algorithm that converges if the stream has a stationary probability distribution with independent elements, but does not necessarily return elements that have actually been observed.

While a naïve algorithm for determining a quantile is $O(N \log N)$—sort ascending and pick the $k$th element—for the large number $N$ of elements found in many databases, or when $N \to \infty$ as in a stream of data which is later summarized and discarded, such an algorithm is impossible to apply. Even substantially less intuitive approaches such as *Quickselect* [53] require $O(N)$ space. To address this issue, many algorithms have been developed for quantile approximations, (see [54, 55, 56] for recent examples,) that require a small fraction of this space. While these algorithms may have a very efficient software implementation, an efficient hardware architecture would require implementing and maintaining the update algorithms and complex data structures that make these approaches possible, and hence require substantial overhead. In this work we develop an architecture—and introduce several extensions—for the algorithm presented in [57] and [58] that only require constant space and time for both storage and updates. As we've mentioned, this algorithm is not guaranteed to be as precise as those in the work mentioned above, and only achieves minimal error when the data stream elements are processed in time independent order. An algorithm without this requirement would return an estimate within $\epsilon$ of the true quantile function even when given a sorted stream—which is very nearly the worst case for our algorithms. In many situations this is a serious drawback, but in several important applications this can be quite advantageous, and we exploit this property to enable testing for bias, and correlation in random streams, described in Section 2.2.4.

Hardware for testing RNGs related to our work here has been developed in many recent publications [26, 1, 28, 29, 27, 25]. The method of [29] implements several tests recommended by the National Institute of Standards and Technology (NIST) [30] using dynamic reconfig-

uration due to the large hardware requirements of all 15 implementations of these tests. In [27] the authors implement versions of two of the NIST tests and optimize them by identifying common operations between tests and approximating the statistical thresholds used. The work of [1]—which is closest to our own—extends this by efficiently implementing eight of the 15 tests recommended by NIST, and approximating the thresholds. The work of [28] and [25] address this issue and provide extension to true—and in [25], non-ideal—random sources, but these works focus on RNGs where the output can be only one of two values, and are potentially insensitive to programming errors or post-processing-based attacks. In [26] the authors of [25] address environmental attacks on true RNGs—those RNGs extracting randomness from their physical environment—and use empirical tests to determine the behavior of the statistical features they extract. Several of the papers mentioned above present work designed to detect problems at the output of the RNG in accordance with the health test paradigm discussed in the relevant NIST recommendation [59]. Our contribution addresses a gap in the taxonomy: nonparametric testing of random variables from any distribution at the point of use.

## 2.2 Background and Methodology

### 2.2.1 Quantiles

An empirical quantile of some random variable $X \sim f_X$, where $f_X$ is a probability density, is the value at a location in a sorted list $L$ of unique entries drawn from $f_X$, where "location" is stated in terms of a fraction of the length of the array. That is to say, if $Q$ is the quantile function of a probability density $f_X$, and $N$ is the number of values drawn from $f_X$, sorted, and placed in $L$, then $Q(\alpha) = L[\lceil N \cdot \alpha \rceil]$ is the empirical $\alpha$-quantile of $f_X$. This implies that the median equals the $\alpha = 0.5$ quantile, and quartiles are the $\alpha = [0.25, 0.5, 0.75]$ quantiles.

**Algorithm 1:** A Simple Quantile Learning Algorithm

**Input:** $\vec{\alpha}$, $\hat{Q}_0$, $\lambda$
**Result:** $\hat{Q}_t \approx Q$
$t \leftarrow 0$
**while** $x_t$ **exists**
  **for every** $j \in [n]$
    $\hat{Q}_t(\alpha_j) \leftarrow \hat{Q}_{t-1}(\alpha_j) - \lambda \text{sgn}_\alpha(\hat{Q}_{t-1}(\alpha_j) - x_t)$
  **end for**
  $t \leftarrow t + 1$
**end while**

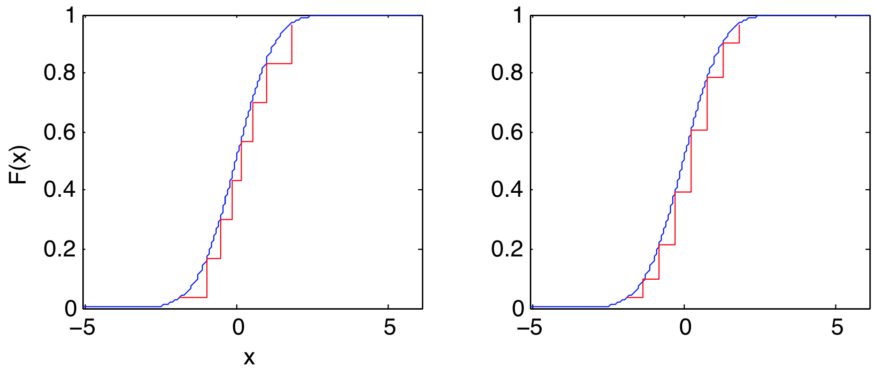


Figure 2.3: Adaptive versus non-adaptive quantile spacing. Results from Algorithms 1 (left, non-adaptive) and 2. The blue curved line is the ideal true CDF. The points of the red steps are the learned result for $n = 8$, $\lambda = 10^{-2}$, and $\zeta = 10^{-4}$. Algorithm 1 misses detail in the tail regions, while Algorithm 2 starting from a equally spaced $\alpha$ vector adapts to represent the tails more densely.

Put somewhat more formally, $Q$ is the inverse of the CDF $F_X$ of $f_X$. Assuming we treat a stream element $x_t$, taken at time $t$ from the stream, as a sample from $f_X$, the CDF of that data stream is $F(z) = \Pr(x_t \leq z)$, and so the quantile function can be written

$$Q(\alpha) = F_X^{-1}(\alpha) = \inf \{x \in \text{supp}(F_X) : \alpha \leq F_X(x)\} \tag{2.1}$$

### 2.2.2 Stochastic Approximation of Quantiles

For each stream element $x_t$ we use the update equation defined in [58] for our approximation. This is

$$\hat{Q}_t(\alpha) \leftarrow \hat{Q}_{t-1}(\alpha) - \lambda \text{sgn}_\alpha(\hat{Q}_{t-1}(\alpha) - x_t) \tag{2.2}$$

where

$$\text{sgn}_\alpha(z) = \begin{cases} -\alpha & \text{if } z < 0 \\ 1-\alpha & \text{if } z \geq 0 \end{cases} \tag{2.3}$$

Given a family of $\alpha_j$ values at which to approximate the quantile function of a univariate stream we can run this update equation in parallel for each $\alpha_j$.

*Theorem:* If samples $x_t$ at $t$ are independently drawn from $\hat{Q}_t(\alpha)$ in Algorithm 1 converges to within $\lambda$ of the $\alpha$-quantile of the data stream generating $x_t$.

*Proof (informal):* We consider only $\alpha = 0.5$ because the reasoning for all $\alpha \in [0, 1]$ flows naturally from this case. Let $x_t$ be a stream element available at time $t$. Recall that by assumption $x_t$ and $x_{t-1}$ are independent in time for all $t$, and so each have an equal chance of being above or below $Q(0.5)$ (the median). Assume $\hat{Q}_t(0.5)$ has converged. Then half of the time $x_t$ cancels out $x_{t-1}$ because $\text{sgn}_{0.5}(x_t) \in \{-0.5, 0.5\}$, and so $\hat{Q}_t(0.5)$ stays where it began, at $Q(0.5)$, in expectation.

Next, assume $\hat{Q}_t(0.5)$ has not converged. If $\hat{Q}_t(0.5)$ is below the median, then there is a probability greater than $0.5$ that $x_t \geq \hat{Q}_t(0.5)$ and so $\hat{Q}_t(0.5)$ will, in expectation, increase by $0.5\lambda$ at each time $t$ with probability equal to $1 - F_X(\hat{Q}_t(0.5))$. This will continue until convergence, i.e. until $x_t \geq \hat{Q}_t(0.5)$ with probability $0.5$, which occurs when $\hat{Q}_t(0.5)$ is within $\lambda$ of the median of the observed stream elements as desired. The logic naturally extends to the case where $\hat{Q}_t(0.5)$ is above the median, and when considering any particular $\alpha$ the intuition also remains the same. □

### 2.2.3 Learning Equal Spacing for $\hat{Q}(\alpha)$

Algorithm 1 is useful when we desire $\hat{Q}(\alpha)$ for a fixed set of $\alpha$ values that we choose a priori. Now we will introduce a modification that allows us to learn $\alpha$ under the constraint that for $j \in [n]$ the $\hat{Q}(\alpha_j)$ values be equally spaced through the range of the distribution, with the exception that both $\alpha_1$ and $\alpha_n$ remain fixed at their a priori values. These recovered $\alpha$ values under this constraint form the CDF computed at equally spaced points over the domain of $f_X$. This implies that the $\alpha$ values that we learn have relatively greater density in areas of low probability, which leads to more accurate tail estimates.

We accomplish this adaptation by adding the second finite difference of all $\hat{Q}_t(\alpha)$, denoted $\Delta^2[\hat{Q}]$ in what follows, to the set of $\alpha$s at each recursive step after attenuation to a small value in the range of the CDF. That is to say

$$\alpha_j \leftarrow \alpha_j + \zeta\Delta^2[\hat{Q}] \tag{2.4}$$

where $\zeta$ is a suitable step size. We advise practitioners to take care when selecting $\zeta$. A heuristic is to set $\zeta = (C \cdot n)^{-1}$, for some large constant $C$. This is because (a) all CDFs are sharply bounded in $[0, 1]$, and (b) we are simultaneously learning $\hat{Q}_t(\alpha)$, and too great a change in $\alpha$ can destabilize other algorithmic components. Note that if $\zeta$ is not sufficiently small then this algorithm can fail, and oscillations in the results indicate that a smaller step size is required. For a visual comparison between the results of Algorithms 1 and 2 see Figure 2.3.

*Theorem:* $\hat{Q}_t(\alpha)$ in Algorithm 2 converges to equally spaced points over between $\hat{Q}_t(\alpha_1)$ and $\hat{Q}_t(\alpha_n)$, while these two converge to those $\alpha$-quantiles of the data stream generating $x_t$.

*Proof Sketch:* This is easy to see when considering that adding $\zeta\Delta^2[\hat{Q}_t]$, for small $\zeta$, forces the second finite difference toward zero, $\zeta\Delta^2[\hat{Q}_t]$ toward a constant, and thus $\hat{Q}_t$

**Algorithm 2:** A Distribution Learning Algorithm

**Input:** $\vec{\alpha}$, $\hat{Q}_0$, $\lambda$, $\zeta$
**Result:** $\vec{\alpha} \approx F_X$
$t \leftarrow 0$
**while** $x_t$ **exists**
  **for every** $1 < j < n$
    $\hat{Q}_t(\alpha_j) \leftarrow \hat{Q}_{t-1}(\alpha_j) - \lambda \text{sgn}_\alpha(\hat{Q}_{t-1}(\alpha_j) - x_t)$
    $\alpha_j \leftarrow \alpha_j + \zeta \Delta^2[\hat{Q}_t]$
  **end for**
  $t \leftarrow t + 1$
**end while**

towards linearity. If $\hat{Q}_t(\alpha)$ are equally spaced, then $\alpha$ are equal to points on the CDF $F_X$, where $Q(\alpha) = F_X^{-1}(\alpha)$, and $\hat{Q}_t(\alpha) \approx Q(\alpha)$.

### 2.2.4 Simultaneous RNG Monitoring and Quantile Learning

To determine whether our RNG under test is biased, we first derive an upper bound $\hat{p}$ on the probability that the stream is unbiased, as

$$z_j = \alpha_j - \frac{|\hat{Q}_t(\alpha_j) - Q(\alpha_j)|}{\lambda t} \tag{2.5}$$

$$\hat{p} = \min_j \left\{ 2n \left( \left( \frac{\alpha_j}{z_j} \right)^{z_j} \left( \frac{1-\alpha_j}{1-z_j} \right)^{1-z_j} \right)^t \right\} \tag{2.6}$$

where $t$ is the number of values tested so far. We derive $\hat{p}$ from the Chernoff-Hoeffding bound [60] on sums of independent Bernoulli random variables. To see this, note that the Chernoff-Hoeffding bound is

$$\Pr\left( t^{-1} \sum_{i=1}^{t} v_i \leq \mu - \epsilon \right) \leq \left( \left( \frac{\mu}{\mu - \epsilon} \right)^{\mu-\epsilon} \left( \frac{1-\mu}{1-(\mu-\epsilon)} \right)^{1-(\mu-\epsilon)} \right)^t \tag{2.7}$$

and if we substitute $z_j = \mu - \epsilon$, where $\mu = \alpha_j$ and $\epsilon = |\hat{Q}_t(\alpha_j) - Q(\alpha_j)|/(\lambda t)$ this gives us the one-tailed, $\mu - \epsilon$, result. However, the bound is symmetric, and so we can take the union bound for the two-tailed test. Since the probability that $|\hat{Q}_t(\alpha_j) - Q(\alpha_j)| \geq c$ is twice that of the deviation without the absolute value, and we are taking $n$ different tests, we multiply by $2n$, making 2.6 a conservative bound on the true probability.

If the input stream is arriving in an independently and identically distributed manner with true $\alpha$-quantile equal to $Q(\alpha)$, then $\hat{p}$ from Eqn. (2.6) will be on the order of $2n$. If the input stream is biased, then $\hat{p}$ will be very small. So for a fixed false rejection probability $\theta$

$$\text{Reject} = \hat{p} \leq \theta \tag{2.8}$$

To simplify this computation, a user can choose the update interval for $\hat{p}$ to be as long as they wish. For the example in Figure 2.7 $\hat{p}$ is computed once per thousand observations. This has little effect other than to make the time until detection last until the end of an interval—recall that $\hat{Q}(\alpha)$ is updated continuously—and allows us to take a leisurely approach to updating $\hat{p}$ using whatever method is most practical. In practice, a user can numerically compute the values of $|\hat{Q}_t(\alpha_j) - Q(\alpha_j)|$ for which Equation (2.6) equals their chosen $\theta$. For example, minimizing (where $z_j$ is defined as in Equation (2.5)),

$$J(z_j) = \left( 2n \left[ \left( \frac{\alpha_j}{z_j} \right)^{z_j} \left( \frac{1-\alpha_j}{1-z_j} \right)^{1-z_j} \right]^t - \theta \right)^2 \tag{2.9}$$

In practice, we avoid numerical issues by computing the positive real root of

$$\begin{aligned} L(\delta_j) &= \log(2n) - \log(\theta) \\ &+ \left( t\alpha - \frac{\delta_j}{\lambda} \right) \log \left( \frac{\alpha}{\alpha - \frac{\delta_j}{t\lambda}} \right) + \left( t - t\alpha + \frac{\delta_j}{\lambda} \right) \log \left( \frac{1-\alpha}{1-\alpha+\frac{\delta_j}{t\lambda}} \right) \end{aligned} \tag{2.10}$$

with respect to $\delta_j = |\hat{Q}(\alpha_j) - Q(\alpha_j)|$. Then $\exp(L(\delta_j))$ will yield a threshold for the absolute

difference instead and simplify computation at each interval. The values of $\hat{Q}(\alpha_j)$ can be reset after a reasonable number of intervals to avoid storing many of these thresholds for different $t$.

Because the Chernoff bound assumes independence, and the movements of $\hat{Q}(\alpha)$ are dependent on the underlying CDF, Equation (2.6) is imprecise. However, we have found that if $\lambda$ is small enough such that a $\lambda$-step of $\hat{Q}(\alpha)$ does not appreciably change the probability of $\hat{Q}(\alpha)$ increasing or decreasing, then Equation. (2.6) holds quite well. "Small enough" will depend on the probability distribution under test, but in our experiments, it is not terribly sensitive, as long as it is small.

We can see in Figure 2.8 that the statistic closely follows the well-known Kolmogorov-Smirnov (KS) and Anderson-Darling (AD) test statistics with varying bias for several distributions. Note also that neither the KS or AD test has a trivial extension to a streaming environment. Additionally, our test is sensitive to autocorrelation—an issue that is not generally possible to detect with either a KS or AD test.

### 2.2.5 Sensitive RNG Monitoring without Quantile Learning

As we noted in Section 2.2.4, the computed $\hat{p}$-value has a dependence on $\lambda$, and will be approximate due to the new probability $1 - F_X(\hat{Q}_{t+1}(\alpha))$ of increase versus decrease of the updated $\hat{Q}_t(\alpha)$. However, if we are not concerned with the quantile-learning component of the RNG test, we may make a straightforward modification that is more sensitive to bias and does not depend on $\lambda$.

Our correction requires that, instead of comparing with $\hat{Q}_t(\alpha)$ at each iteration, we compare with $Q(\alpha)$—the true quantile of the distribution we are testing for. This requires a single additional register to store the true $Q(\alpha)$ value. This also requires the removal of the connection to the $\hat{Q}_t(\alpha)$ register, shown as the dashed and marked connection in Figure 2.4.

## 2.3 Implementation

### 2.3.1 A $\hat{Q}(\alpha)$ Functional Unit

A $\hat{Q}(\alpha)$ unit with $n = 1$ leads to the hardware architecture depicted in the block diagram of Figure 2.4. All $\hat{Q}(\alpha)$ hardware units are independent except for the read-only value $x$. On account of this, increasing $n$, and hence the number of $\alpha$ values, creates a group of nearly identical $\hat{Q}(\alpha)$ units. We implemented our designs using Xilinx Vivado High Level Synthesis (HLS) targeting a Zynq-7020 SoC.

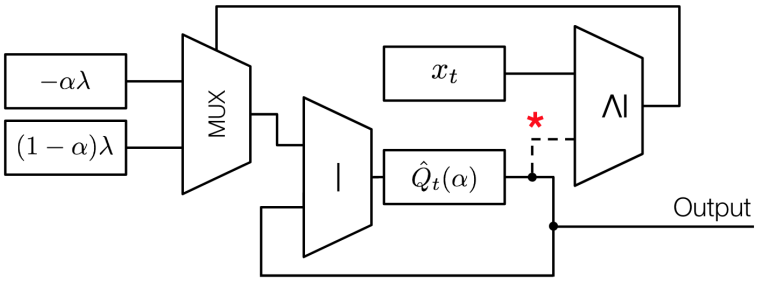


Figure 2.4: A hardware estimator for a single $\hat{Q}(\alpha)$. The dashed wire marked with the asterisk can be moved to a register containing $Q(\alpha)$ (not shown) for more sensitive RNG testing, but this disables the learning feature of the algorithm.

Referring to the HLS code for this operation, shown in Figure 2.5, we initialize $(1-\alpha)\lambda$ and $-\alpha\lambda$—in Figure 2.4 these are arrays *pos* and *neg* respectively—only once before use, and these values remain constant throughout. In practice this loop may be unrolled manually or using the HLS option #pragma HLS UNROLL to minimize impact on latency. Also note that the ternary statement is necessary to ensure that the multiplexer is generated as desired. FPGA area and performance estimates are listed in Table 2.1.

```
for (int j = 0; j < n; ++j)
  Q_hat[j] -= (Q_hat[j] > x) ? pos[j] : neg[j];
```

Figure 2.5: HLS Code for Algorithm 1. See discussion in section 2.3.1.

Table 2.1: Area and Performance Estimates for a Single $\hat{Q}(\alpha)$ Unit. Latency (Lat) is in cycles, Throughput (Tp) in MHz

| | Lat | Tp | FF | LUT | DSP |
|---|---|---|---|---|---|
| 8-bit int | 2 | 164 | 33 | 67 | 0 |
| 32-bit int | 2 | 164 | 129 | 267 | 0 |
| 32-bit float | 7 | 143 | 400 | 867 | 2 |

Users should make sure to select matching data types for both $\hat{Q}(\alpha)$ and the arrays *pos* and *neg* as shown in Figure 2.5. If these data types do not match, the user (or tool) will have to insert a (relatively) expensive conversion to the highest precision type, which is often floating point, for the subtraction. This is almost never beneficial or necessary. The most common case, where $\hat{Q}(\alpha)$ is integer and $\alpha\lambda$ is not, can be easily solved by rounding $\alpha\lambda$ and interpolating to find something near the true value. If $\alpha\lambda$ loses an unacceptable amount of precision by doing this, it is very likely that the domain of the distribution of interest is small enough to require a fixed or floating point version. Now is a good time to mention again that the resulting $\hat{Q}(\alpha)$ is an *estimate* that can be as much as $\lambda$ away from $Q(\alpha)$ at any time when the input is completely independent in time, which would be the ideal case.

While Algorithm 1 can be implemented efficiently in hardware with relative ease, Algorithm 2 is somewhat more challenging. Our adaptations are described in the next section and result in Algorithm 3.

### 2.3.2 A Hardware Friendly Approximation

Algorithm 3 is an approximation of Algorithm 2. To make Algorithm 3 hardware amenable, we eliminate multiplications completely, and replace the very small $\zeta\Delta^2[\hat{Q}_t]$ values with small constants. Algorithm 3 updates $\lambda\alpha$ at every time step instead of $\alpha$, and so in in order to obtain the $\alpha$ values as viable points on the CDF in question, $\alpha$ must be divided by $\lambda$. If a group of $\alpha$ values are needed at every time step, then this division might be prohibitive, but if it is possible for us to choose $\lambda$ to be a power of two, then these concerns are obviated for the

**Algorithm 3:** An Approximation of Algorithm 2

**Input:** $\vec{\alpha}$, $\hat{Q}_0$, $\lambda$, $\zeta$
**Result:** $\vec{\alpha} \approx F_X$
$t \leftarrow 0$
**while** $x_t$ **exists**
  **for** $1 < j < n$
    $b \leftarrow \begin{cases} \lambda & \text{if } \hat{Q}_{t-1}(\alpha_j) \geq x_t \\ 0 & \text{otherwise} \end{cases}$
    $g \leftarrow \begin{cases} \zeta & \text{if } \Delta^2[\hat{Q}_t](\alpha_j) \geq 0 \\ -\zeta & \text{otherwise} \end{cases}$
    $\hat{Q}_t(\alpha_j) \leftarrow \hat{Q}_{t-1}(\alpha_j) - (b - [\lambda\alpha]_j)$
    $[\lambda\alpha]_j \leftarrow [\lambda\alpha]_j + g$
  **end for**
  $t \leftarrow t + 1$
**end while**

most part. Replacing $\zeta\Delta^2[\hat{Q}_t]$ with $\zeta$ has the asymptotic effect of adding an error of at most $\zeta$ to the output.

The hardware block diagram in Figure 2.6 shows a three-point CDF estimator. $\alpha$ of the two extreme end points are fixed, and so we reuse the $\hat{Q}(\alpha)$ functional units as shown in Figure 2.4 for them. Like the $\hat{Q}(\alpha)$ units, the area of this architecture scales as $O(n)$, as all elements shown in the figure must be duplicated for each additional $\alpha$, minus one subtract unit and the end-point $\hat{Q}(\alpha)$ units. For area and performance results for a an implementation when $n = 3$ see Table 2.3

Table 2.2: Comparison with Previous Work [1]

| | This work* | [1] all 8 tests |
|---|---|---|
| FF | 774 | Sum of 8 = 519 |
| LUT | 402 | Sum of 8 = 934 |
| Throughput (MHz) | 164 | Min of 8 = 132 |

* The architecture shown in Figure 2.4 with $n = 6$ replications.

We compare our implementation to the closest prior work [1]—an implementation of approximations of eight tests from the NIST Statistical Test Suite—in Table 2.2. We have

made an effort to make the comparison fair by working under the assumption that all eight of these tests are implemented simultaneously on the same FPGA, and (as stated in [1]) are not sharing resources. So we compare to the sum of the area consumption of the eight tests from [1]. The throughput of the test supporting the slowest clock will dictate the rate at which other tests, and the RNG under test, can be run. Thus we take the minimal throughput from [1] for our comparison. For our work, we select a 32-bit implementation of $\hat{Q}(\alpha)$ with $n = 6$ replications with differing $\alpha$s. This is not the most hardware efficient implementation we have proposed, but it is the most fair comparison.

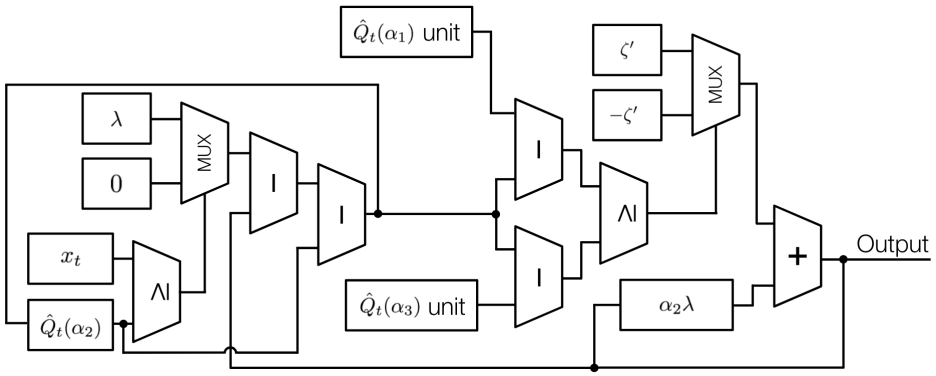


Figure 2.6: A hardware architecture for CDF estimation at three points.

Table 2.3: Area and Performance for a 3-point CDF Estimator. Latency (Lat) is in cycles, Throughput (Tp) in MHz

| | Lat | Tp | FF | LUT | DSP |
|---|---|---|---|---|---|
| 32-bit float | 25 | 117 | 1,030 | 2,107 | 4 |

## 2.4 Experimental Results

We compare three of the PRNGs mentioned in [28] in our experiment (see Table 2.4). LFSR and BigMod are known to be biased. LFSR over-represents small values, while BigMod has a slightly "lumpy" histogram. Note that we have verified that neither of these passes the NIST Statistical Test Suite (STS), (0/15 for LFSR, 1/15 for BigMod), while Twister passes

Table 2.4: PRNGs Scored in Figure 2.7.

| PRNG | Implementation |
|---|---|
| Twister | Mersenne Twister [61] generating 31 bit integers |
| LFSR | $x \leftarrow x^{31} + x^{28} + 1$ |
| BigMod | $x \leftarrow 1583458089 \cdot x \mod (2^{31} - 1)$ |

14/15 of these tests with test parameters set to the defaults. For our STS tests we used 550 groups of 16,000 bits each, making $8.8 \times 10^6$ bits in all.

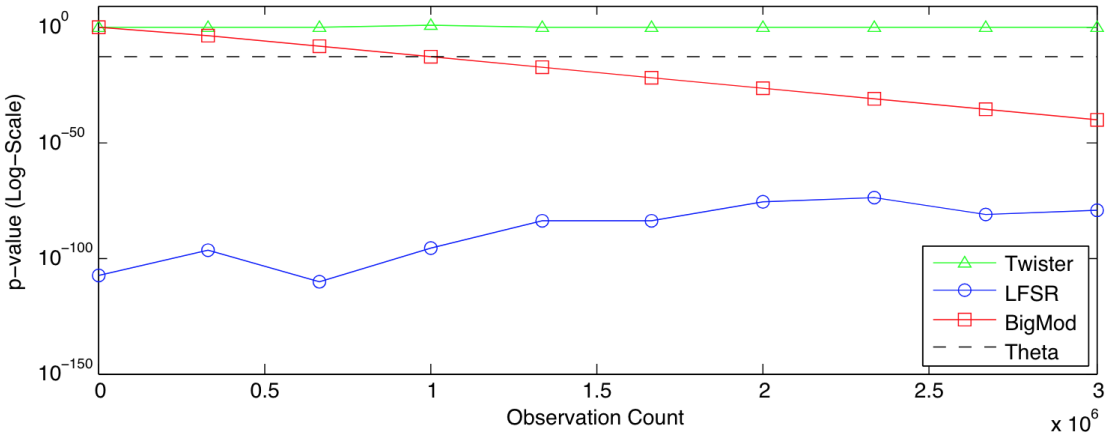


Figure 2.7: A comparison of $\hat{p}$-values over a sequence of three million observations from the PRNGs in Table 2.4, with $n = 6$. $\theta$ (dashed line) is a constant threshold computed for a false positive rate equal to $2^{-40}$. See section 2.4.1 for discussion.

### 2.4.1 Inversion Sampling Bias Test

For our example, we test the three PRNGs listed in Table 2.4. Our null hypothesis is that these PRNGs produce a uniformly random sequence of integers in the range $[1, 2^{31} - 1]$. Most importantly, we do not test the outputs $x$ of the PRNGs directly, instead, we post-process each, to make $x'$, using inversion sampling according to the following:

$$x' = \hat{F}_X^{-1}(x/(2^{31} - 1)) \cdot (2^{31} - 1) \tag{2.11}$$

where $\hat{F}_X^{-1}$ is an fine grained approximation to the quantile function of a standard normal distribution and we set $Q(\alpha)$—our null model—accordingly. $\hat{F}_X^{-1}$ is approximate in that we bin the $2^{31}-1$ unique values from the PRNGs into $22,734$ bins equally spaced over $[-10^{10}, 10^{10}]$.

$\hat{p}$-values computed at intervals of $10^3$ for $n = 6$ during three million observations are shown in Figure 2.7. See equation (2.6) for the details of $p$-value computation.

Using the rather relaxed $\theta = 2^{-40}$, we reject the null hypothesis the first time the test statistic crosses the reject threshold $\theta$. We can see from Figure 2.7 that even a significantly larger $\theta$ would not reject Twister, but would reject the others considerably sooner.

## 2.5 Conclusion

In this work we have developed an algorithm for learning values, $\hat{Q}(\alpha)$, and probabilities, $\alpha$, of an unknown CDF, and included modifications to ease hardware implementation. We demonstrated the usefulness of the technique for uncovering bias in sources of randomness when considering the entire random number generation pipeline as opposed to only the RNG or PRNG. We have also shown that our work compares favorably in terms of both area consumption and throughput with previous less flexible approaches to run-time bias detection.

Chapter 2, in part, is a reprint of the material as it appears in the Proceedings of the 54th Annual Design Automation Conference (DAC), June 2017. Althoff, Alric; Kastner, Ryan, IEEE 2017. The dissertation author was the primary investigator and author of this paper.

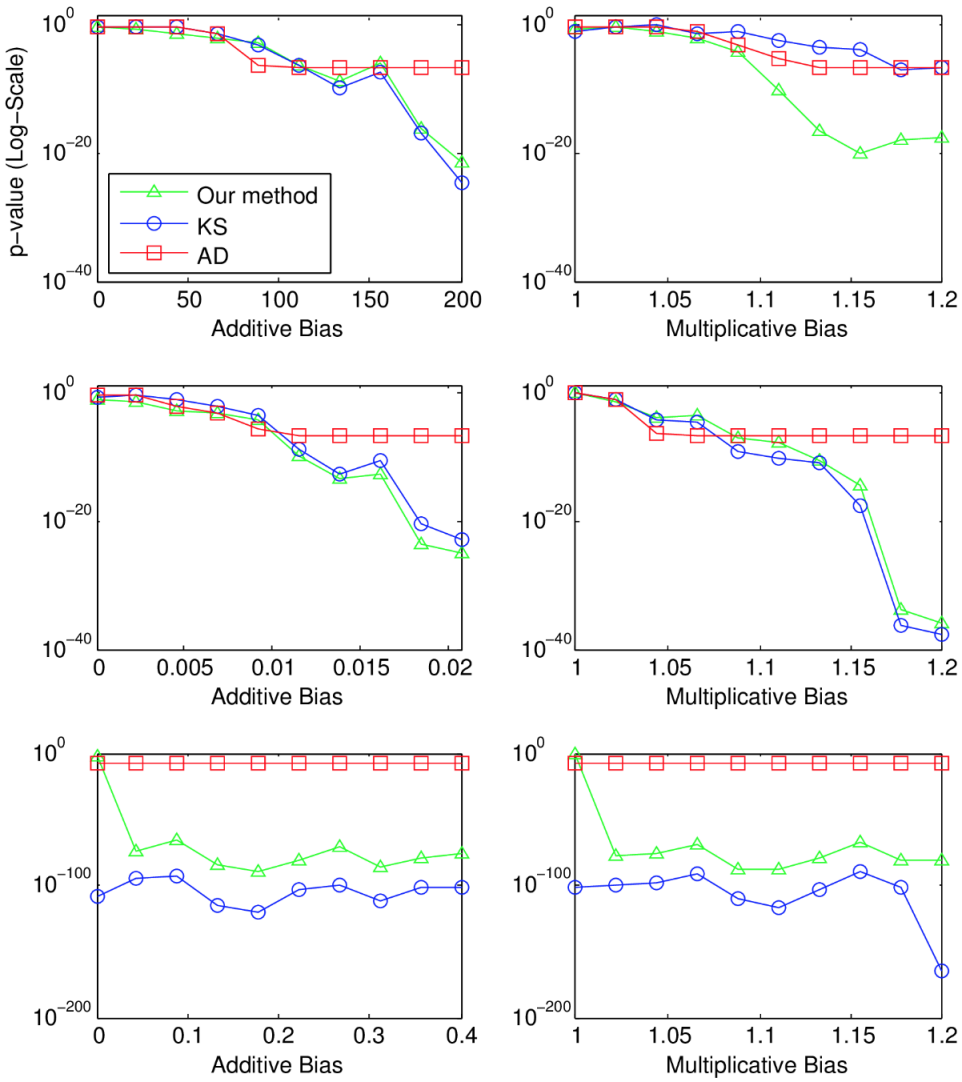


Figure 2.8: A comparison of $p$-values from Anderson-Darling (AD) and Kolmogorov-Smirnov (KS) tests against those of our statistic (see Eqn. (2.6)). Each row are results given a different initial probability distribution, points are $p$-values at varying degrees of bias. The top row is from a normal distribution with $\mu = 0$ and $\sigma = 1000$, second is a beta distribution with parameters $(a, b) = (2, 10)$, and the bottom row is a Poisson distribution with parameter $4$. Each point is a $p$-value computed given a sequence of three thousand observations with $n = 6$ and $\lambda = 10^{-4}$. Note that the AD test for $p$-values $< 6 \times 10^{-9}$ are not computed and set to $6 \times 10^{-9}$ due to the excessive computation necessary to determine smaller $p$-values.

# Chapter 3

# Holistic Side-Channel Leakage Assessment

## 3.1 Introduction

Later on we will dive into a discussion on power side-channels specifically, but first we will sketch the ideas surrounding side-channels in general, define some terminology, and sketch a few important mathematical concepts. A *side-channel*, in the sense that we mean it, is an unintended communication channel for information. In the case of computing devices, this information could be about processor state, or the data involved in the computation.

If a side-channel exists, then we say that it *leaks* information, or *is leaky*. For this reason, many publications discussing side-channels use the term *leakage function*, which we will write, using $\cdot$ (the centered dot) to stand in for the function's parameters,

$$\mathbf{y} = L(\cdot) \tag{3.1}$$

when referring to mathematical descriptions of the channel. Outputs from the leakage function are the observations that an analyst or attacker will make.

### 3.1.1 When do side-channels exist?

A necessary condition for the existence of a side-channel is a resource that is both altered by computations on confidential data, and observable by the attacker. This resource could be time, electrical energy, or a more abstract resource provided by the system, such as memory.

For example, assume some process "A" running on a computer system depletes the entropy pool of an internal random number generator. Additionally assume that it may deplete the pool to a point where the RNG stops providing values at the same rate. In this case, some other process "B" that can also observe this entropy pool can determine information about the state of process A. In this case the entropy pool is the resource, and the observable outputs of the leakage function are the lag times detected by process B.

There are certainly many ways to prevent this scenario, and many of them involve ensuring that the output of the leakage function observable by process B is no longer changing based on the RNG use of process A. Sometimes this is not possible, or at least more difficult than is warranted by the value of the data about process A. This brings us to another intuition that appears in the context of mitigation strategies: If the recovery of information about process A is sufficiently difficult to dissuade any attacker from expending the resources needed to recover it, then the side-channel is also effectively mitigated. In fact, we can rephrase this discussion in the context of information transmission channels in general, and use Shannon's definition of channel capacity from [62] to make more precise statements.

### 3.1.2 Relevant Information Theory

Next we will summarize a few information theoretic results. These will help build intuition for later discussions where we will talk about side-channels more formally in order to develop metrics. Let $X$ be a random variable taking values with probability $p(X = x)$ for discrete values $x \in \Omega_X$. Next, recall that the number of symbols, using an alphabet

of $b$ symbols, required to distinctly represent some cardinal value $x$ is $\log_b x$. So in order to represent the probability of $x$ with sufficient precision, we require at least $\log_b p^{-1}(x) = -\log_b p(x)$ symbols from the alphabet. Elsewhere in this thesis we assume that $b = 2$, units are bits, and omit the base of the logarithm in the notation.

Following the intuition that unexpected/unlikely values are more surprising and thus contain more information, $-\log p(x)$ is a measure of the information content of an event/value $x$. This leads us directly to the conclusion that the expected value of information per symbol, or *entropy* [62], is

$$H(X) = -\sum_x p(x) \log p(x) \tag{3.2}$$

The *rate* of transmission of information about the state of $X$ through a channel is then

$$R = H(X) - \delta \tag{3.3}$$

where $\delta$ can be thought of as "missing information" due to channel effects.

We can reframe $\delta$ as the residual uncertainty about $X$ when the channel output $Y$ is known. This is exactly the conditional entropy

$$\delta = H(X|Y) = -\sum_{x,y} p(x,y) \log \frac{p(y)}{p(x,y)} \tag{3.4}$$

where $p(x, y)$ is the joint probability of $(x, y)$, making the transmission rate

$$\begin{aligned} R &= H(X) - H(X|Y) \\ &= -\sum_{x} p(x) \log p(x) + \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(y)} \\ &= \sum_{x,y} p(x, y) \log p(x, y) - \sum_{x,y} p(x, y) \log p(y) - \sum_{x,y} p(x, y) \log p(x) \\ &= \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \\ &= I(X; Y) \end{aligned} \tag{3.5}$$

where $I(X; Y)$ is the mutual information between $X$ and $Y$. The maximum possible rate of transmission through a channel—the *channel capacity*—is then $C = \max_X I(X; Y)$.

At this point, things work like we intuitively think they should. For example, if $x$ is an $n$-bit value then $\max(1, n/C)$ is the average number of measurements that are required read $x$ from the channel. That is to say, if a noiseless $n$-bit channel is saturated then we need one $n$-bit measurement on average to be certain about the value of $x$. This intuition extends to channels of other capacities as well. Let's say that our channel capacity is $n/10$-bits, then we require ten measurements on average to recover any $n$ bits.

Now we have constructed some useful tools for talking about side-channel analysis. If we allow *channel* to be synonymous with a leakage function having outputs $Y$, i.e. $Y = L(X)$, then in the context of side-channel analysis, a mitigation strategy works by reducing the magnitude of $C$ with respect to confidential information $X$—ideally, to zero[1]. From the definition of $C$ we can tell that $C$ is zero when $p(X, Y) = p(X)p(Y)$, which defines statistical independence. This means that if $p(X)$ and $p(Y)$ are statistically independent, then $C = 0$ and a side-channel does not exist for measurements $Y$ about data $X$, because infinitely many meansurements would be required to learn $X$ from $Y$.

[1] We'll take the convention that $n/0 = \infty$.

## 3.2 Power Side-Channel Analysis

While side-channels come in many flavors, for the rest of this dissertation we will focus on the power side-channel. There are many specific varieties of power side-channel analysis that can successfully recover data, typically cryptographic secrets, used or generated by a processor. However, the methods require only minor modification to intercept other forms of data, such as trade secrets or health records. The techniques are also generally non-invasive, requiring only that the device function as it usually would in a standard deployment environment.

Differential Power Analysis (DPA) is one of the earliest side channel attacks, and is still in use due to its simplicity and effectiveness. Its notoriety is such that in commercial and industrial circles "DPA" is often used to refer to the entire class of power side-channel attacks. It can be performed with low cost equipment in a small amount of time [63].

DPA attack success is conditioned on the fact that the energy consumed by the computational device will be differ depending upon the values of input data. The changes in power draw during the execution time interval reveal information about the input, intermediate, and output data, regardless of whether that input data is confidential. By taking measurements, or *traces*, of these power fluctuations, an attacker may make educated guesses about the data used and/or produced during the execution of the algorithm, for example, "is the least significant bit of the result of a computation '0'or '1'?" This can happen, and is in fact easier in some respects, when the instructions executed are exactly the same across all traces, as is common for cryptographic algorithms. DPA and its more modern derivatives, statistically verify whether or not a guess about the data[2] is correct, which can quickly lead to determining the secret information. Typically an attacker only needs to know a few details about the computation being performed, for example, which cryptographic algorithm in running.

[2]In practice, these guesses are about small portions of a larger piece information. For example, a single byte, or even a single bit, at a time can be recovered, and a large number of bytes can be recovered this way with a little patience and/or automation.

A successful attack is possible even with a significant amount of noise. However, additional noise increases the required amount of execution trace data because the channel capacity is decreased based on the signal-to-noise ratio. As an example, a DPA attack on a particular AES software implementation running on a microcontroller requires approximately 200 traces to determine the entire key, while one hardware (ASIC) implementation is more difficult, requiring approximately 6500 traces [64].

The effectiveness of DPA is due to the large reduction in search space required to recover the desired information, for example, to $16 \times 2^8$ from the $2^{128}$ required for a brute force attack on a 128-bit secret key. This is also true for Correlated Power Analysis (CPA) and Template Attacks (TA) [65, 39]. These methods improve upon DPA by using models that are consistently correlated with the measurements and/or constructing these models using measurements from the physical device itself. Where DPA relies on trace differences, CPA an TA have an advantage in execution time and affirmative key recovery (low false positive rate). For CPA, this advantage comes through use of a simple but powerful leakage model, which we discuss further in Section 4.5.1. The leakage modeling approach only requires that the attacker know what algorithm is being run, and then they can correlate the power model outputs for specific sub-key guesses with the actual measurements to determine the correct key. In TA, this leakage model is (optionally [40]) eliminated. Instead, an attacker has access to the device of the type they intend to attack ahead of time. They can collect (at least) several thousand traces and construct a probabilistic model of measurements for each sub-key hypothesis [39]. Then an attacker will use these models to quickly recover the correct key, occasionally with as little as one power trace and often with less than one hundred [66][3].

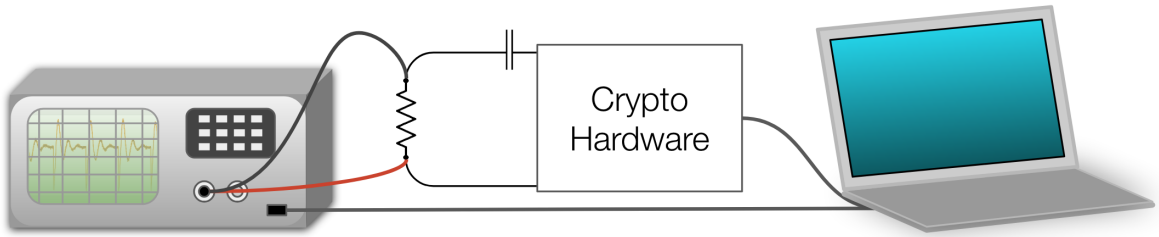


Figure 3.1: A simplified diagram of a power side-channel attack measurement setup.

**Threat Model**

In the model we assume in this thesis, an individual conducting power-side channel analysis will have physical access to the hardware device, along with knowledge of the cryptographic algorithm. Though it is not necessary, it is common to assume that the attacker will also have complete information about the hardware. We will assume this "white box" approach, though it is seldom necessary, in which there are no unknowns with respect to either the algorithm or the hardware device.

The attacker must also have some form of measuring equipment, commonly an oscilloscope, to collect measurements correlated with power-use of the processor over time, and a computer on which to perform analysis of these collected measurements. These measurements are seldom of actual power consumption (in watts) instead they are often of voltage or current alone, because they are measurable and power-correlated quantities. Notice that this implies that a "power attack" is not particularly sensitive to the actual measurement source, and there will often be large amounts of noise from the operation of other devices mixed in with the quantities of interest.

The attacker (or security analyst) will then collect samples during many, typically on the order of thousands, of runs of the algorithm on the device of interest, prior to proceeding to the analysis stage. Figure 3.1 depicts a typical setup.

[3]CPA and TA attacks have themselves spawned large families of attack strategies that differ almost entirely in the statistics they rely on. However, the underlying principles of these newer attacks are not essentially different. In general, an attacker will favor a simple and well-known approach over a more complex or demanding one.

### 3.2.1 Leakage Modeling

A critical component to many modern attacks is the *leakage model*.

$$\hat{\mathbf{y}} = \hat{L}(\cdot) \tag{3.6}$$

Many modeling approaches have been proposed [67, 68, 69, 70] beginning from device-level physical modeling via SPICE, to approximations that are far less time consuming. Of these approximations, the *Hamming distance leakage model* is by far the most common because it has been shown repeatedly to be robust and quick to compute. The Hamming distance model is

$$\hat{\mathbf{y}} = \hat{L}_{\omega}(a, b) = \omega(a \oplus b) \tag{3.7}$$

where $\omega(x)$ is the Hamming weight of $x$, which is the number of set bits in $x$ when represented in base-2, $\oplus$ indicates exclusive-OR (XOR), and $a$ is the prior state and $b$ is the current state, also in base-2, of the current operation's output. The central assumption of this model is that the energy required to change the state of a single bit in a memory element is significantly more than that required to maintain its prior state.

Notice that we do not say *power* model—this is in part because power consumption is only weakly correlated with many leakage models and we would like to avoid confusion. Another somewhat subtle point is that power side-channel attackers are really not particularly concerned with power at all! *Leakage* of information is what matters, and, as we discuss in Section 3.1 and 3.1.2, detectable information leaking through side-channels is based on *changes* in a common resource—the channel's transmission medium. It is thus only necessary that $\hat{L}$ and $L$ are somehow related in a known way, and perhaps only often enough to enable attacks to succeed in a reasonable amount of time.

More precisely, it is necessary for $I(\hat{L}(\cdot); L(\cdot)) > 0$ where $I(X; Y)$ is the mutual information defined in Equation 3.5, which means that they cannot be statistically independent.

This is not sufficient for practical attacks, and a *good* leakage model will correlate directly with the true leakage function, or at least $I(\hat{L}(\cdot); L(\cdot)) \gg 0$, and thus reduce the number of measurements required for the analyst to draw definitive conclusions.

### 3.2.2 Correlated Power Analysis

Fundamentally, power side-channel analysis (PSCA) is a classification problem: The attacker collects data during a cryptographic operation, and attempts to determine the class, corresponding to the current secret key, of the collected measurements. All power side-channel attacks aim to either specifically recover the key, or narrow the number of key candidates down to a value that permits tractable brute force search of those keys that are most likely. The difference between attack types lies in the particular technique used to classify, or discriminate, between differing underlying data. Though DPA [71], is the most widely discussed attack method, it has been superseded by other techniques and is almost never used in practice, and the principles and statistics used by Correlated Power Analysis (CPA) [65], underlie most present-day attack techniques.

To conduct a by-the-book CPA attack, an attacker will acquire a vector of measurements

$$\mathbf{y} = L(t, s, \mathbf{m}) \tag{3.8}$$

where $L(t, s, \mathbf{m})$ is the leakage function from a specific time point—or point-of-interest (POI)—$t$, with a fixed and unknown secret $s$, and a vector of messages $\mathbf{m}$. There are attacks for cases where $\mathbf{m}$ is either known or unknown. If $\mathbf{m}$ is unknown, then we may assume the attacker has access to the results of the algorithm. In the case of a cryptographic algorithm, the attacker would have access to the encrypted ciphertexts and perhaps know something about $\mathbf{m}$ as well, so this assumption is not terribly permissive. The attacker also computes a *leakage model*

$$\hat{\mathbf{y}} = \hat{L}(t, \mathbf{s}_{sub}, \mathbf{m}) \tag{3.9}$$

(see Section 3.2.1) using knowledge of the algorithm under analysis. Note the use of $\mathbf{s}_{sub}$ to represent a vector of *sub-key hypotheses*—each sub-key is usually 8-bits long (a byte) but can be larger or smaller depending on the particular implementation of the algorithm under analysis.

With $\mathbf{y}$ and $\hat{\mathbf{y}}$ in hand, the attacker applies a decision function to determine which sub-key hypothesis from the leakage model best matches the measurements. The decision function or *discriminator* used by the CPA attack is the linear (Pearson's) correlation $r$ [72], computed between length-$n$ vectors $\mathbf{y}$ and $\hat{\mathbf{y}}$ as

$$r(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum_{i=1}^{n} \hat{y}_i y_i - n\bar{\hat{\mathbf{y}}}\bar{\mathbf{y}}}{(n-1)s_{\hat{\mathbf{y}}} s_{\mathbf{y}}} \tag{3.10}$$

where $\bar{\mathbf{x}}$ and $s_{\mathbf{x}}$ are the arithmetic mean and standard deviation of the vector $\mathbf{x}$.

To perform the attack, the attacker scans through all the sub-keys hypotheses in their hypothesis set and selects the sub-key having the largest correlation $r$ with their leakage model. Doing this for each sub-key yields the entire cryptographic key.

### 3.2.3 Template Attack

Template attacks are a significantly more powerful class of attack for two reasons, the first is that the discriminator statistic relies on an empirical model of device power consumption (for this reason they are fall into the class of *profile-based* or simply *profiled* attacks), and the second is that a TA easily captures leakage from multiple points of interest simultaneously. In contrast to the CPA or DPA class of attack, the attacker will have access to a device of similar make/model ahead of time, and collect a set of *profiling traces*

$$\mathbf{y}_p = L\big(\mathbf{t}_p, \mathbf{s}_{sub_p}, \mathbf{m}\big) \tag{3.11}$$

where we use the subscript $p$ to indicate "profile". For each sub-key hypothesis they will collect a new set of $\mathbf{y}_p$ measurements. For example, an AES-128 key (of 128 bits) could be divided into 16 single-byte sub-keys. If an attacker chooses each single-byte sub-key as a hypothesis class, then they will have $2^8 \times 16 = 4096$ data sets to collect, each of which could contain upwards of one thousand data points. Despite the seemingly large number of data points in such a profile data set, the process can be largely automated, and only needs to be performed once for a device class.

From these profile data, the attacker will form, for each sub-key hypothesis, a discriminator function, or "template", that is in the ideal case maximized when the sub-key of a measurement set has the value corresponding to the profile key used when collecting the data to form the template. Various statistics can be applied, but a the classical variant is the *Gaussian template* computed for $\mathbf{y}_p$ of length-$n$ as,

$$f_{sub}(\mathbf{y}) = \frac{1}{(2\pi)^n |\mathbf{\Sigma}_p|} \exp\left(-(\mathbf{y} - \mu_p)\mathbf{\Sigma}_p^{-1}(\mathbf{y} - \mu_p)\right) \tag{3.12}$$

for which $\mathbf{\Sigma}_p \in \mathbb{R}^{n \times n}$ and $\mu_p \in \mathbb{R}^n$ are the covariance matrix and mean vector respectively of measurements $\mathbf{y}_p$, and "sub" indicates that this $f$ is maximized for a particular sub-key hypothesis. This means that a TA against AES-128 using a single-byte sub-key hypothesis, an attacker will need to form $4096$ templates, and while smaller hypothesis classes can be used, 4096 is not a prohibitively large template set.

Templates in hand, an attacker proceed with the attack—collecting power measurements from the target device. They will select the sub-key that maximizes that sub-key's respective template ($f_{sub}$ in the Gaussian template example). Doing this for all sub-keys yields the entire key. Occasionally, this can be done using a single power trace, when it cannot, many attack traces can be combined by taking their average before applying the template.

## 3.3 Measuring Device Security

A metric for *whole device* certification should provide a numeric score so that hardware vendors and end users can rank devices with fine granularity and make informed decisions. This means we prefer methods giving comparable numeric values to a "pass/fail" qualitative indicator of vulnerability. Additionally, a practical metric would have modest data requirements and be quick to execute so that tests, and testing labs, can be less expensive to run. In line with this requirement, a metric should make *valid* confidence intervals available so that a quick estimate can be differentiated from a thorough analysis, thus allowing security grades to be qualitatively compared.

Mean Traces to Disclosure (MTD) [34] is one commonly used metric. This metric is fundamentally tied to the employed attack method(s). Attacks are improving rapidly and becoming less expensive to execute. This means that a robust SCLA metric should not be tailored to detect vulnerabilities based on a specific attack type—or even a group of attack types. Instead, as pointed out in [73], a metric should indicate whether information about secret data is leaking though the tested side-channel for a particular device under test (DUT) in a way not linked to a particular power model or type of attack.

The Test Vector Leakage Assessment (TVLA) from Cryptography Research Inc. [74] makes strides in this direction by removing the requirement for a model of power consumption. However, it is univariate, i.e., it only considers one sample at a time. Multivariate, or "higher-order," modes of the TVLA exist, but these violate the assumptions of the underlying statistic and provide uncertain conclusions [75]. While many attack techniques are univariate (DPA, CPA, template attacks), very powerful multivariate attacks have been developed [39, 76, 38] and we expect this trend to continue. A metric that detects vulnerability to these multi-target attacks must be *holistic* in the sense that it should consider all time points *and their interdependencies*. Additionally, TVLA assumes measurements are Gaussian and independent. Making assumptions about the underlying distribution of the power trace data may

be statistically invalid when these assumptions don't hold. Thus, an ideal metric would avoid such assumptions.

In this chapter, begin by we describing and experimentally demonstrating the limitations of existing single-dimensional and higher-order side channel leakage assessment (SCLA) metrics for power SCA. We then provide a framework for robust multidimensional metrics and introduce a new robust and model-free holistic SCLA metric within this framework. We then evaluate our metric on power traces from different AES implementations, and show that it correlates with state of the art test and attack results.

This chapter is organized as follows, Section 3.4 defines the power SCA threat model and implications for SCLAs, Section 3.5 discusses current SCLAs, Section 3.6 gives theoretical, synthetic, and real-life examples of situations that holistic SCLAs should be sensitive to. In Section 3.7, we introduce our framework for an ideal holistic SCLA, and in Section 3.8 we develop an algorithm within this framework. In Section 3.9 we verify our metric experimentally, and we conclude in Section 4.6.

## 3.4 Threat Model for Power SCA

We assume an attacker can cause security critical programs to run with arbitrary inputs and collect detailed measurements $\mathcal{M}$ of device power use from a specific device under test (DUT). For example, the attacker could write a program which calls either a library function, or cryptographic hardware, to encrypt their chosen data while gathering voltage measurements over time with a connected oscilloscope. We additionally assume that the attacker can synchronize these measurements so that all times $\mathbf{t} = [t_1, t_2, \ldots, t_n]$ during computation are aligned across runs, and that she can attack any or all of these points of interest (POI). She also knows precisely when the start of the computation occurs, e.g., by using a simple power analysis [63]. These are common assumptions across power analysis attacks.

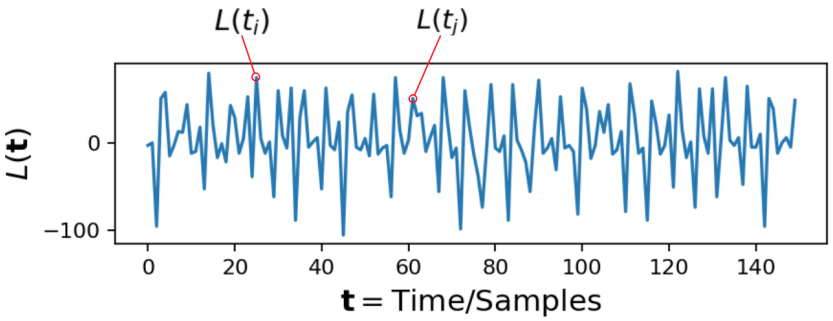


Figure 3.2: An example power trace from DPA Contest v4.2 [77]. $L(t_i)$ and $L(t_j)$ are leakage at different time indices. Univariate SCLAs (and attacks) only use leakage at one time index, multivariate SCLAs (and multi-target attacks) use more than one, while a holistic test (our approach) uses any subset of the trace, and can use the full trace. Practical attacks often use at least several hundred or thousand traces.

With respect to SCLA, we assume that the equipment used to record the power traces, or "leakage", $L(\mathbf{t}) \in \mathcal{M}$ during a leakage assessment is at least as good as that available to an attacker, i.e., with comparable or greater sampling rate, bit depth, and noise characteristics. We note that while relatively low-cost equipment is often used for power analysis attacks [63], a leakage assessment should endeavour to equal or supersede the attacker's capabilities. We also assume the assessor has sampled traces in a representative manner. Our sampling method is discussed in Section 3.9. Figure 3.2 shows an example power trace that would be collected for attack.

Our technique focuses on leakage assessment for power SCA. Other side-channels (timing, acoustic, RF, etc.) are outside of our threat model, although this technique is likely to be much more broadly useful.

## 3.5 Related Work

The goal of any empirical assessment of leakage is to determine if it is statistically possible to correctly classify disjoint groups of traces, $D_0 \subset \mathcal{M}$ and $D_1 \subset \mathcal{M}$, corresponding to different secret keys. Many techniques [20, 78, 19, 79] exist to detect information leakage,

but all of these approaches are either univariate (testing a single time points one-by-one), require explicit selection of a small subset of time points, have onerous data requirements, or offer indefinite conclusions. We will discuss these points in Section 3.6.

The focus of recent work in univariate SCLAs [80, 78], with multivariate extensions [81], is the TVLA [74]. It is an attractive alternative to MTD because it doesn't emphasize a particular attack or power model. In this section, describe it and discuss its benefits and drawbacks. While variations on the TVLA test have been proposed, (other univariate hypothesis tests in particular,) these share many or all of the shortcomings of the TVLA, making it a good example for discussion.

### 3.5.1 Test Vector Leakage Assessment (TVLA)

A typical TVLA [74] for use with the Advanced Encryption Standard (AES) begins with the collection of many power traces from the device under test for either fixed or random plaintexts[4] divided into two different groups, $D_0$ and $D_1$, described in [74]. Then the assessor will perform a two-tailed Welch's $t$-test on particular points of interest (POIs) in time to determine if there is a statistically significant difference between the traces in $D_0$ and those in $D_1$ at those POIs. If $\bar{x}_i$ is the sample mean of traces at a single POI, $s_x$ is a sample standard deviation, and $N_x$ is the number of traces taken in $D_x$, the $t$-statistic is

$$t = \frac{\bar{x}_0 - \bar{x}_1}{\sqrt{s_0^2/N_0 + s_1^2/N_1}} \tag{3.13}$$

Assuming that the data are independently drawn from Gaussian distributions with unknown variance, we may use $t$ to determine the probability that these two Gaussian distributions have equal means $\mu_1$ and $\mu_2$. This probability is computed under the Student's $t$ distribution parameterized by the degrees of freedom d.f.. The formula commonly used to

[4] It may seem counterintuitive that keys do not always differ between groups, see Section 3.9 for details.

approximate d.f. is well-known [81], and available in many statistical software packages, so we will not reproduce it here.

The TVLA assumes that the DUT is secure at a particular time index if the $t$-test fails for to reject the null hypothesis a confidence level $\alpha$ equivalent to $99.999\%$. The assessor is free to check any time index in the sampled power traces where secret data may be used, with algorithm-specific recommendations such as during S-Box access in the first or last round of AES.

It is also used to validate higher order statistical moments, i.e., variance, skewness, kurtosis, etc. However, estimation of these higher order moments is very sensitive to noise, time-consuming, and rather numerically unstable. Our approach negates this issue by avoiding higher moment-based statistics entirely. They are not necessary in non-parametric holistic testing, and as we will show in Section 3.6, may even lead to incorrect conclusions about security of the DUT.

### 3.5.2 Multivariate Combining

An extension to multiple variables is explored for attacks in [82] and used in the TVLA assessment regimen of [74, 81]. The method combines leakage from multiple time indices via a combination function and then performs a univariate assessment as before with a somewhat higher threshold. The optimal combiner analyzed in [82] computes the centered product of the samples at the POI, that is

$$L(t') = \prod_{t \in \text{POI}} L(t) - \bar{x}(t) \tag{3.14}$$

such that the resulting collection of $L(t')$ act as pseudo-traces for the collection of times in the set of POI. The corresponding test then uses moments computed over this pseudo-trace set.

In order to combine time indices we must first identify the indices we wish to test. This

requires another stage of analysis resembling what in machine learning and statistics is called *feature selection*. POI determination for SCLA has a critical difference: where feature selection *minimizes* the size of the set of indices that we need to use, it discards redundant features, in the context of power SCA an attacker would find redundant indices equally exploitable. Therefore, we must consider time indices that are redundant to be equally leaky. Intuitively, this means that our identification of POI should also allow us to label redundant time indices, and we are unlikely to know this if we choose POI a priori for a previously un-analyzed piece of hardware. So multivariate assessments that do not operate holistically, i.e. on every sample in the trace, should include a technique for selecting these POI. The examples in Section 3.6 suggest that a POI selection algorithm for a multivariate SCLA should also be multivariate, and this is a complex problem in itself.

This being said, our method of holistic testing does not require identifying POI at all. Because holistic testing checks all POI at once, we never need to solve the problem of identifying time indices to test. However, that doesn't mean that this is a limitation of holistic testing; a set of POI could certainly be tested using a holistic test regime, but we believe it is important to create security metrics that require *minimal* a priori assumptions, even assumptions about what set of times during execution an attacker may select for use.

As an example of how such assumptions may harm attack mitigation efforts, consider S-Box masking [35] for AES. Masking has been designed to protect the most common attack POI in AES, however, as we can see in the results of DPA Contest v4.2 [77], which invites researchers to attack AES with a masked S-Box implementation, a successful attack can be performed by relocating the attack target to a different POI. One attack shown in the contest results does just that, moving their POI to ShiftRows and recovering the entire key.

## 3.6 Sensitivity Requirements for Robust SCLA

In this section we will identify what types of information leakage an SCLA should be able to detect, and what statistical pitfalls an SCLA should avoid. First, we demonstrate that any SCLA that is *robust* to attacker ingenuity or future algorithmic and hardware developments must consider multivariate leakage *jointly*.

### 3.6.1 Motivating Examples

Multivariate leakage exists. This is evidenced by the increased effectiveness of multivariate attacks such as [39, 76, 38]. What is less clear is that this type of leakage is also *quite likely*. In this section we will show some examples of multivariate leakage, and explain why two (or more) points can be considerably more powerful when considered together.

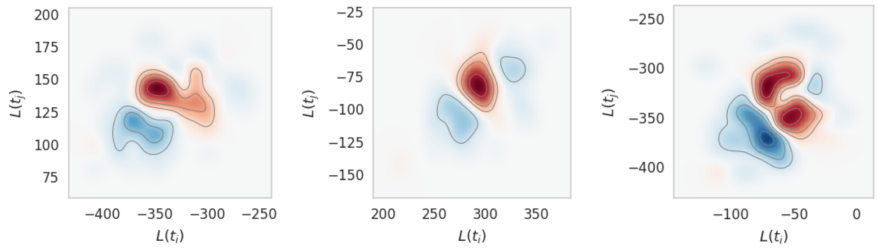


Figure 3.3: Three examples of nonlinear bivariate leakage in real measurements. $L(t_i)$ and $L(t_j)$ of (a), (b), and (c) are from power traces where times $t_i$ and $t_j$ are rated secure by TVLA and the multivariate extension, but a nonlinear SVM classifies the traces into correct key distributions with over 70% accuracy. Red corresponds to the distribution of key 1 and blue to key 2. Such examples are not uncommon.

Figure 3.3 shows distributions from power traces from an FPGA AES core. Figure 3.3(a), 3.3(b), and 3.3(c) are bivariate kernel densities of voltage measurements taken from different pairs of time points $(i, j)$. Colors correspond to different keys. Each of these bivariate distributions is rated secure[5] by TVLA and also by the multivariate combining method of

[5] $4.5\sigma$ implies a $p$-value $< 10^{-5}$, 3.3(a), 3.3(b), and 3.3(c) have $t$-test $p$-values $> 0.001$ for both TVLA and TVLA with multivariate combining.

[74, 81], but a nonlinear SVM classifies them each with a cross-validated accuracy greater than 70%. So, if we attack one of either time point $t_i$ or $t_j$, with power traces $L(t_i)$ or $L(t_j)$, or even if we consider them at the same time, but independently, we would have a harder time attacking them than if we consider them jointly.

In statistics and machine learning this type of interaction is called *variable complementarity* and, as we can see, certainly exists in trace samples at different points in time. Additionally, it is possible for complementarity to exist and yet be statistically impossible to detect using any moment-based statistics, or even those based off of statistical independence between keys and traces!

This may seem surprising, but consider the following example: for Boolean $x$ and $y$, $x$ XOR $y = z$, but knowledge of $x$ alone does not reveal any information about $z$ ($x$ and $z$ are statistically independent), and neither does knowledge of $y$, however, knowing both completely determines $z$.

Replace $z$ by the secret key, and $x$ and $y$ by power trace measurements, and we can see that this example implies that it is theoretically possible to have a power measurement $L(t_i)$ which under the *best* univariate metric at *all statistical moments* will indicate that there is little to no information in the traces about the secret key, but by considering additional samples at some other time $t_j$ all key bits may be recovered. This example also applies to univariate tests of statistical independence and mutual information. This implies that neither of these criteria are sufficient to detect all forms of clearly exploitable multivariate leakage.

Figure 3.4 shows a synthetic example that is close to what we might see in practice. Assume that the orange and blue points are leakage measurements $L(t_0)$ and $L(t_1)$ at two time indices $t_0$ and $t_1$. Further, assume that the orange and blue points are measurements taken during a cryptographic operation where only the key differs. The univariate $t$-test on $L(t_1)$ marginal distributions between indicates that they differ significantly, with a $p$-value of effectively zero that the distributions have means that are equal. A Bayes-optimal classification

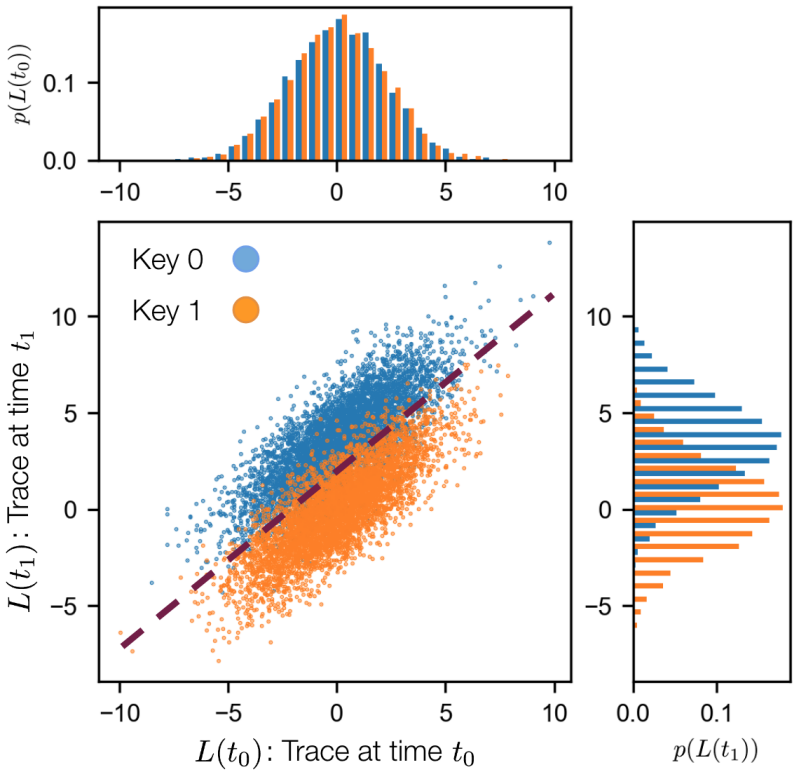


Figure 3.4: A synthetic example where a univariate metric would underestimate vulnerability. Considering the $L(t_0)$ and $L(t_1)$ axes jointly is more powerful than either one alone. We may not have known that $L(t_0)$ was relevant without testing all pairs of samples. See Section 3.6 for details.

using only $L(t_1)$ data (right histogram) has a success rate of 78%. A similar $t$-test on time index $L(t_0)$ gives a $p$-value of 0.07, which passes—i.e. is not flagged as "leaky"—due to the TVLA detection threshold of $p < 10^{-5}$ recommended in [74]. However, if $L(t_0)$ and $L(t_1)$ are used together, the Bayes-optimal linear decision boundary has a classification accuracy of 91%. This implies that a multi-target or higher order attack using power trace data from two time indices with a joint probability distribution resembling that of Figure 3.4 would be considerably faster and have a higher success rate than if the $L(t_1)$ data were used alone. It is possible for this to happen in higher dimensions than just two, e.g., nine time points could seem individually useless, but when combined with a tenth, it could increase the attackability of the DUT by a significant amount. This is a problem with using strict thresholds for security

evaluations in general.

Our security metric (Section 3.8) successfully detects information leakage in these scenarios.

### 3.6.2 Common SCLA Issues

#### Multiple Comparisons

Many tests use a threshold $\alpha$ for the $p$-value that is not changed regardless of the number of tests conducted. While this leads to an issue that applies to any hypothesis test conducted multiple times, we will use the TVLA as an example.

To see the problem, consider that the TVLA requires us to conduct a separate $t$-test for each POI, and each of these POI have probability $10^{-5}$ of exceeding the threshold *if the null hypothesis is true*, i.e. the means of the presumptive underlying Gaussian distributions are the same. However, if we test each sample in the entire trace, then for traces more than $10^5$ samples long we should expect for the test to reject the null hypothesis at least once by chance alone. For this reason an uncorrected test procedure is vulnerable to false "insecure" ratings when multiple POIs are tested.

Recently researchers have made mathematically sound efforts to address this. For example, [78] applies a goodness of fit test to the $p$-values from the $t$-test at each POI in the trace, and rejects the null hypothesis of DUT security if this $p$-value distribution is non-uniform. Unfortunately, even though a few recent works have begun mentioning this issue e.g. [19], it remains unimplemented in the analysis. We stress that if correction is not used during multiple testing, the $p$-values should not be considered correct; they no longer represent the probability of a false reject.

**Assumption Violations when Testing Hypotheses**

The multivariate combining technique in Section 3.5.2 is an assumption violation for the $t$-test: Even if the samples for all POIs are Gaussian and independent the product distribution of these random variables is non-Gaussian. As written, Eqn. (3.14) does not correct the distribution of $L(t')$ such that the $p$-value returned by a $t$-test is the probability of a false rejection of the null hypothesis; because $t$-statistics derived from these data will not have a Student's $t$ distribution, derived $p$-values will be heuristics. This a particular concern when using these "$p$-values" as a *comparable* indicator of certainty in the result.

Consider too that Eqn. 3.14 was analyzed in [82] in the context of second order CPA, where a combination function is part of the attack method. It is optimal in the sense that it is the best in terms of a Hamming weight power model when using Pearson's correlation to distinguish between differing keys when a combining function is required. This does not imply optimality in general.

**Poorly Defined Test Criteria**

If the SCLA is sensitive to criteria that are too weak to indicate security, then we run the risk of falsely labeling a device secure when it is not. The opposite can be damaging as well: A test that is sensitive to factors *beyond* DUT security can cause practitioners to waste time and effort attempting to fix issues that are artifacts of the SCLA method but unrelated to security of the DUT.

To make this point more clear, consider a bivariate Gaussian distribution with random variables $(X, Y)$ where $X$ and $Y$ taken alone, (i.e. their marginals,) have equal means and standard deviations. If we are given samples of $X$ and $Y$ and asked to discern the most likely $Y$ given $X$, we might inspect their correlation to determine a likely region for $Y$. However, if we replace $Y$ with plaintext inputs, and $X$ with power traces, it seems odd that we should be able to determine the *key* from the relationship of $X$ and $Y$.

Yet this is exactly the protocol applied by many tests [74]. The reasoning for this fixed-vs.-random input (plaintext) protocol is that certain attacks (e.g. CPA) require knowledge of a set of plaintexts or ciphertexts, and predictable variations in the traces due to these values enables the attack. However, in the general case, dependence on the plaintexts does not matter (e.g. [39] represents a family of attacks that do not require knowledge of plaintexts), and we do not want holistic testing to be tied to certain attack techniques.

This example implies that it is entirely possible to to develop an *effective* SCA countermeasure that alters statistical dependence between traces and plaintexts arbitrarily. SCLAs relying on tests of these quantities would inappropriately label a design as insecure after the mitigation is implemented. The formal definitions in Section 3.7.1 and metric framework we supply in this paper do not have this limitation.

While the work of [75] mentions, with specific reference to the $t$-test, several of the issues that we have pointed out, e.g. there is an inappropriate risk of false positives and negatives, and notes the multiple comparison issue, in the end, many of their points and assumptions are specific to AES S-Box masking. Side-channel attacks, hardware architectures, and new algorithms are being developed rapidly, and so we would like to address the overall problem. Therefore, in this work we are not developing a framework or an algorithm that focuses on any implementation specifically. Instead, we propose that metrics address the overarching problem: measuring the complexity of the hypothesis class necessary to handle particular massively multi-class classification problems using feature vectors with very high dimension.

## 3.7 A Framework for Holistic SCLA

In this section we will introduce a framework for holistic—considering the entire trace at once—and robust SCLA. The main components of this framework are a null hypothesis which assumes vulnerability given a compact but well-founded definition of exploitable leak-

age, non-parametric confidence intervals, and a minimum of implicit assumptions influencing the resulting scores.

### 3.7.1 A Baseline Vulnerability Formalism

We do not assume that a device is secure prior to having tested it, and this is rarely, if ever, a reasonable assumption for a DUT pulled off a shelf at random. Thus, a holistic SCLA should begin in the state of "insecure/unmitigated" with respect to an unknown DUT and work to collected evidence to the contrary. This is a philosophical, rather than mathematical, departure from other SCLAs. The null hypothesis of other tests is that the device is secure. For example, the null hypothesis, $H_0$, of the $t$-test is that the means of the of the two groups are identical, which would imply that the DUT is secure.

The major reason for this perspective is that we believe we should choose a null hypothesis that "imitates life". Without descending too far into the arguments of Fisher and Pearson from nearly a century ago [83], we also note the logical fallacy that data that does not contradict a hypothesis "proves" it. For a binary alternative, a rejection of one possibility implies acceptance of the other, but this is a mathematical abstraction in general. In the real-world case there are many reasons that we may be lead to reject a particular hypothesis. Experiments cannot be blind to a reality that dictates vulnerability as a baseline.

In order to formalize our intuition of "vulnerability" as key distinguishability given measurements, we need to define the type of leakage we are testing for:

**Definition 1** (Exploitable Leakage)**.** *If there exists for some key distributions $\mathcal{K}_0$ and $\mathcal{K}_1$, and measurements $X_0 \sim \mathcal{K}_0$ and $X_1 \sim \mathcal{K}_1$ where $(X_0, X_1) \stackrel{d}{\neq} (X_1, X_0)$ then the pair of key distributions $\mathcal{K}_0$ and $\mathcal{K}_1$ have exploitable leakage $> 0$ where $\stackrel{d}{=}$ denotes identical probability distribution.*

In other words, devices leak secret information if the random variables underlying the power traces are not *exchangeable* with respect to different secrets.

**Definition 2** (Exchangeable Random Variables [84]). *If a tuple of random variables*

$$(X_0, \ldots, X_n) \stackrel{d}{=} (X_{\pi(0)}, \ldots, X_{\pi(n)}) \tag{3.15}$$

*for arbitrary permutations $\pi$ then the random variables $X_i$, $1 \leq i \leq n$, are exchangeable.*

This means that for $n$ exchangeable random variables, all $n!$ permutations have the same joint distribution. With respect to power traces and secret keys, this means that we cannot organize traces into bins corresponding to their most likely key with an accuracy better than chance. Exchangeability is a strictly stronger property than identical distribution, (i.e. all exchangeable random variables are i.d., but not the reverse,) and so detecting this property also indicates when measurements are classifiable by most likely distribution It is also strictly weaker than independence, which we have already shown by counterexample is too strong.

This logic allows us to state that the baseline state, or null hypothesis $H_0$, of a holistic SCLA should be that traces are not exchangeable over secrets. This stems directly from the definition; if it is usually possible to differentiate power traces measured when using different secret keys, then the DUT is vulnerable to power attacks.

### 3.7.2 Nonparametric Confidence Intervals

Statistically derived values always have an associated uncertainty. This can either be due to noise or non-representative sampling. Any security metric should be accompanied by a confidence interval such that—for a fixed probability $\alpha$—the true value of the statistic lie within the range. This will allow us to say something about the range in which the true value of the statistic may lie.

In addition, an ideal confidence interval would not be make assumptions about measurements, models, or the relationship between measurements and latent variables unless this is justified. Specifically, when deriving confidence bounds we should not assume that mea-

surement noise is Gaussian, or even that it is "noise" that cannot be perfectly deconvolved using known processor state.

### 3.7.3 A Lack of a Priori Assumptions

$t$-tests assume that the data are Gaussian and that samples are independent—things that are very unlikely to be true for power traces in general [85]. In the case of a holistic SCLA, we would like our test to be valid across all known—and ideally, unknown—hardware types. This prohibits us from using tools that rely on untested assumptions about the nature of the measurements. For this reason, we believe that a holistic SCLA metric should be as assumption-free (nonparametric) as practically possible in order to be comparable across devices and robust to changing technologies and attack vectors.

While it is common practice in statistics to take advantage of assumptions that have *empirical support* given the sample in question, there is a danger of false inference when imposing these assumptions on situations where they do not apply. We are not making a case against assumptions in general, but we do feel that we should have high *empirical* confidence that they are true for specific data before relying on them.

## 3.8 Our Proposed SCLA Metric

Now that we have identified the criteria for a holistic SCLA, we are in position to derive a metric in accordance with these principles.

### 3.8.1 A Holistic Assessment Criterion

Consider two sets, $D_0$ and $D_1$, consisting of $N$ traces each, where each trace is $M$ samples long. Our algorithm is a Holistic Assessment Criterion (HAC) that is motivated by the fact that if the sampled traces in $D_0$ and $D_1$ are identically distributed then the $k^{th}$ nearest

**Algorithm 4:** Holistic Assessment Criterion (HAC)

**Data:** Matrices $\mathbf{L}_0, \mathbf{L}_1 \in \mathbb{R}^{N \times M}$ of $N$ measurements, each $M$ samples long. $\mathbf{L}_0$ contains traces from $D_0$, and $\mathbf{L}_1$ from $D_1$, and $\alpha$, specifying the confidence region, such that $\mathbf{z}$ will be within $[\mathbf{c}_l, \mathbf{c}_u]$ with confidence $1 - \alpha$.

**Result:** A sorted vector of proportions $\mathbf{z}$, with upper and lower confidence intervals $\mathbf{c}_u$ and $\mathbf{c}_l$, and the HAC slope (average slope of $\mathbf{z}$), for which larger values indicate more vulnerability of the DUT.

// Concatenate the trace matrices in both orders; 01 and 10
$\mathbf{J}_{01} \leftarrow \mathbf{L}_0 \| \mathbf{L}_1$
$\mathbf{J}_{10} \leftarrow \mathbf{L}_1 \| \mathbf{L}_0$
// Pairwise distances between rows of $\mathbf{J}_{01}$ other rows in $\mathbf{J}_{01}$
$\mathbf{T}_{\text{self}} \leftarrow d(\mathbf{J}_{01}, \mathbf{J}_{01})$
// Pairwise distances between rows of $\mathbf{J}_{01}$ and those of $\mathbf{J}_{10}$
$\mathbf{T}_{\text{other}} \leftarrow d(\mathbf{J}_{01}, \mathbf{J}_{10})$
// Concatenate the pairwise distance matrices
$\mathbf{C} \leftarrow \mathbf{T}_{\text{self}} \| \mathbf{T}_{\text{other}}$
// Form the set-membership matrix $\mathbf{Q}$
**forall** $i \in [N]$ **do**
    $\mathbf{s} \leftarrow \text{sortAscending}(\mathbf{C}_{i,:})$
    **forall** $k \in [2N]$ **do**
        **if** $s_k$ *is from* $\boldsymbol{T}_{self}$ **then**
            $\mathbf{Q}_{i,k} \leftarrow 0$
        **end**
        **else if** $s_k$ *is from* $\boldsymbol{T}_{other}$ **then**
            $\mathbf{Q}_{i,k} \leftarrow 1$
        **end**
    **end**
**end**
// Sum up the columns of $\mathbf{Q}$
**forall** $k \in [2N]$ **do**
    $\mathbf{z}_k \leftarrow \frac{1}{N} \sum_{i=1}^{N} \mathbf{Q}_{i,k}$
**end**
// See Fig. 3.6 for plots of sorted $\mathbf{z}$ and confidence intervals
$\mathbf{z} \leftarrow \text{sortAscending}(\mathbf{z})$
// By Hoeffding's inequality [60]
$\epsilon \leftarrow \sqrt{\log(2/\alpha)} / \sqrt{(2N)}$
$[\mathbf{c}_l, \mathbf{c}_u] \leftarrow [\mathbf{z} - \epsilon, \mathbf{z} + \epsilon]$
// HAC slopes are reported in Table 3.1
HAC slope $\leftarrow N/(N-1) \sum_{i=2}^{N} \mathbf{z}_i - \mathbf{z}_{i-1}$

neighbor of any trace picked at random from $D_0$ (resp. $D_1$) will have an equal probability of being from $D_0$ as $D_1$. To formalize the intuition,

**Theorem 1.** *Let $A$ and $B$ be composed of samples from the random variables $X$ and $Y$ respectively. If $X$ and $Y$ are identically distributed, then the nearest neighbor $y \in A \cup B$ of any $x \in A$ is a member of $A$ or $B$ with equal probability.*

*Proof Sketch.* Identically distributed random variables $X$ and $Y$, with samples $x$ and $y$, obey $\mathbb{P}(x \in C) = \mathbb{P}(y \in C)$ for all subsets $C$ of the sample space. This implies that, for some open region of the sample space, the probability of a sample landing in that region is equal for $X$ and $Y$. Further assume that this region is a $\delta$-thin band at distance $\rho$ from another sample, that is, $(\rho - \delta, \rho + \delta)$, where we let $\delta \to 0$. Earlier statements imply that if $X$ and $Y$ have identical distributions, then a sample of either $X$ or $Y$ will have equal probability of being in this band. Letting $\rho$ be the distance of a nearest neighbor completes the proof. Notice that this is true regardless of the distance function we use, so long as it is a well-defined distance metric on all $C$. □

This means that if sets $A$ and $B$ with cardinality $n$ are identically distributed, then a Boolean indicator of membership in $B$ for the list of $1 \leq k^{th} \leq 2n - 1$ nearest neighbors to all points in $A \cup B$ would have a Bernoulli$(p = 1/2)$ distribution, and so the sum of these indicators would have a binomial$(2n - 1, p = 1/2)$ distribution.

This yields a test that indicates whether two distributions are equal. Moreover, it does not make any assumptions about the distributions and is fast and robust to data with high dimension $M$.

With this technique in place, the only thing left to do to verify exchangeability directly from Def. 2 is to consider the traces from $D_0$ and $D_1$ *jointly*, that is, with the traces appended end-to-end, then swap them, and check to see if the joint traces are identically distributed.

To restate this in precise terms, first, we will "stack" all the traces as row vectors from $D_x$ into two $N \times M$ matrices $\mathbf{L}_x$, where $x \in \{0, 1\}$. Then, we will concatenate these matrices,

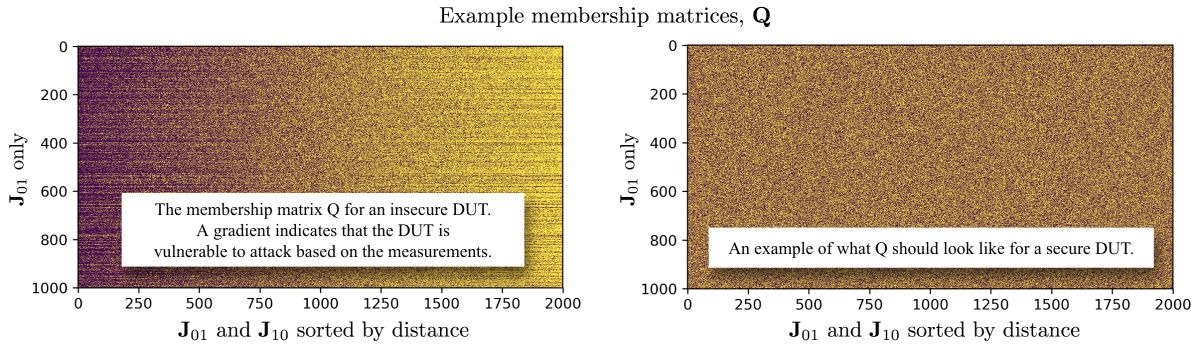


Figure 3.5: Example membership matrices **Q** for HAC evaluation. Column $k$ of **Q** shows the group membership, $D_0$ or $D_1$, of the $k^{th}$ nearest neighbors to the trace of row $i$. For secure devices and algorithms, entries of **Q** should be independently $0$ (purple/dark) and $1$ (yellow/-light) with probability $1/2$. If there a gradient, or vertical "stripes" visible in **Q**, then the DUT is insecure. This is because **Q** would not have a Bernoulli$(N, p = 1/2)^{N\times 2N}$ distribution. In general, though visual inspection can serve as an early warning sign, a statistical test is required. Steps for construction of **Q** are discussed in Section 3.8 and Algorithm 4.

to form two $N \times 2M$ "joint trace" matrices $\mathbf{J}_{01} = \mathbf{L}_0 \| \mathbf{L}_1$, and $\mathbf{J}_{10} = \mathbf{L}_1 \| \mathbf{L}_0$, where $\|$ in this case signifies matrix concatenation. Note that if $\mathbf{J}_{01} \stackrel{d}{=} \mathbf{J}_{10}$, then the sets $D_0$ and $D_1$ are exchangeable. To this end, we can make use of Thm. 1.

Let $d(\mathbf{A}, \mathbf{B})$ be the matrix of pairwise distances between each row of **A** and all rows of **B**. So $\mathbf{T} = d(\mathbf{A}, \mathbf{B})$ implies $\mathbf{T}_{i,j}$ is the distance between row $i$ of **A**, (which we write $\mathbf{A}_{i,:}$) and row $j$ of **B**, ( $\mathbf{B}_{j,:}$). In our algorithms and experiments we use the Euclidean distance $d(x, y) = \|x - y\|_2$, but this is, as we can see in the proof of Theorem 1, one of many possible statistically equivalent choices. We use this distance function to compute the joint pairwise distance matrices,

$$\mathbf{T}_{\text{self}} = d(\mathbf{J}_{01}, \mathbf{J}_{01}) \tag{3.16}$$

$$\mathbf{T}_{\text{other}} = d(\mathbf{J}_{01}, \mathbf{J}_{10}) \tag{3.17}$$

$$\mathbf{C} = \mathbf{T}_{\text{self}} \| \mathbf{T}_{\text{other}} \tag{3.18}$$

We then identify the nearest neighbors in order of distance and mark the membership of these

neighbors with respect to one of the joint matrices. That is, we sort each row of the concatenated distance matrix $\mathbf{C}$ in ascending order and form a new Boolean matrix $\mathbf{Q}$, letting $\mathbf{Q}_{i,k}$ be the entry of $\mathbf{Q}$ at the $i^{th}$ row and $k^{th}$ column, $\mathbf{Q}_{i,k} = 0$ if the $k^{th}$ nearest neighbor of row $i$ of $\mathbf{J}_{01}$ is some row of $\mathbf{J}_{01}$, and $\mathbf{Q}_{i,k} = 1$ if it is from $\mathbf{J}_{10}$. See Figure 3.5 for two examples of $\mathbf{Q}$ matrices, one from a vulnerable DUT, and an what $\mathbf{Q}$ should look like for an ideally secure DUT.

This is a direct test of Definition 2, and Theorem 1: We now have joint variables in two permutations ($\mathbf{J}_{01}$ and $\mathbf{J}_{10}$), if they are identically distributed, then the samples are exchangeable. More specifically, we may infer that the DUT is insecure, given that the samples are representative, if the proportion, $\mathbf{z} = N^{-1} \sum_{i=1}^{N} \mathbf{Q}_{i,:}$ when sorted, has an average (arithmetic mean) slope greater than that predicted by the quantile function of a binomial distribution with parameters $N =$ Number of traces in $D_0$, $p = 1/2$. By symmetry of distance, it is not necessary to form the "full" distance matrix. As we will demonstrate in Section 3.9, this slope is an indicator of overall attack difficulty. We describe our technique step-by-step in Algorithm 4.

Using this technique of examining the proportion of nearest neighbors we are able to detect exchangeability, up to limits imposed by noise, numerical issues, and the number of data points. While statistically drawn conclusions cannot constitute mathematical proof, we can be certain enough for comfort. To this end we propose the confidence intervals in Section 3.8.1.

### Confidence Intervals on z

We certify our result with a probabilistic bound on $\mathbf{z}$ for the reasons discussed in Section 3.7.2. Hoeffding's inequality [60] implies that a two-sided bound on the deviation of an empirical proportion $\hat{p}$ from the true proportion $p$ decreases with the number of measurements $N$ according to the rule,

$$\mathbb{P}(|\hat{p} - p| > \epsilon) \leq 2 \exp(-2N\epsilon^2) \tag{3.19}$$

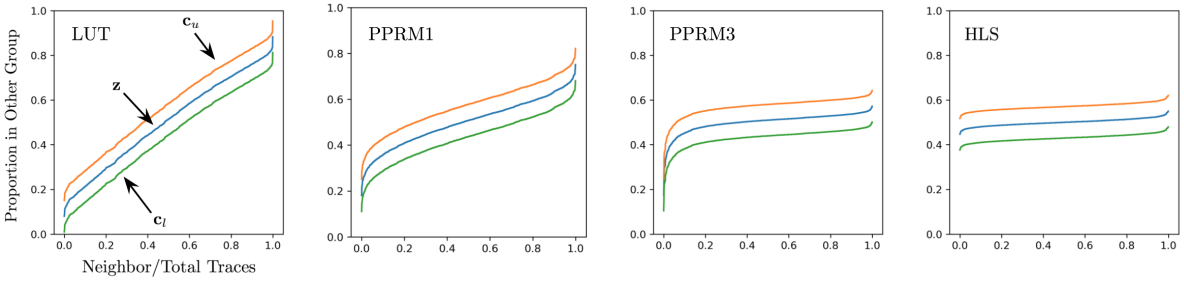


Figure 3.6: HAC plots of the four AES architectures tested in Section 3.9. The horizontal axis of each plot is the fraction of the total number of columns in $\mathbf{Q}$ ($x$-axis 0.0 representing the first neighbors, and 1.0 representing the farthest neighbors), and the vertical axis is the proportion of joint traces from $\mathbf{L}_0\|\mathbf{L}_1$ whose $k^{th}$ neighbors are in $\mathbf{L}_1\|\mathbf{L}_0$ instead of $\mathbf{L}_0\|\mathbf{L}_1$. If the blue/center line ($\mathbf{z}$ from Algorithm 4) has a greater average slope, then the groups of traces in $D_0$ and $D_1$ are less exchangeable, and the DUT is less secure. The orange/upper and green/lower lines are the $\alpha = 10^{-4}$ confidence interval $[\mathbf{c}_l, \mathbf{c}_u]$ about the center line. That is, we may be sure that the true proportion lies within the bounded region with probability $1 - 10^{-4}$.

this means that we can solve for a region where $p$ will be with probability $1 - \alpha$ given $\hat{p}$ by setting

$$\epsilon = \sqrt{\log(2/\alpha)/(2N)} \tag{3.20}$$

and so

$$[\mathbf{c}_l, \mathbf{c}_u] = [\mathbf{z} - \epsilon, \mathbf{z} + \epsilon] \tag{3.21}$$

is a valid confidence interval for each $p$ given the empirical proportions $\hat{p}$ in $\mathbf{z}$. $\mathbf{c}_u$ and $\mathbf{c}_l$ are shown for our experiments in the upper and lower lines of the Fig. 3.6 plots.

### Interpretation of HAC Results

First things first: For a DUT to be secure, the slope of $\mathbf{z}$ should be small, and confidence intervals should be narrow. Confidence interval width depends on measurement count.

We anticipate that the reader will question our approach. Why not use $p$-values from a test of $\mathbf{z}$ against the binomial distribution? We believe that a single $p$-value is too reductive. In our HAC, we leverage the fact that if the probability of set-membership is $1/2$, then the vari-

ance of the proportion will narrow for $N$ measurements at a rate of $1/(4N)$. This implies that the slope of the non-decreasing sequence **z**, the empirical quantile function of the proportion of other-set membership, should shrink at a rate of $1/(4N)$ as well, but will never quite reach zero, which is the "perfect score" for a secure DUT. We take the view that it is very likely *impossible to empirically prove* that a design is perfectly secure, however, we can have a high confidence in such a result by observation, and our approach supports this view.

Additionally, as we collect more data, the confidence intervals around **z** will narrow, and so we can use these to say when "enough is enough," either with regards to our confidence that a design is secure, or that it is insecure to a certain degree. This enables us to make informed statements about the HAC ranking between designs, which is our central goal in this chapter.

Also, our approach avoids the problems of multiple comparison, unverified model assumptions, inappropriate metric criteria, and hypothesis tests which only give only qualitative pass/fail indicators without a degree of precision. This is true *even though it does not reveal the most vulnerable points of interest* (POI). Though it is possible to apply HAC to sub-sections of the traces and conduct a search for highly vulnerable regions.

## 3.9 Experiments

To validate our metric we compare correlated power analysis (CPA) [65] attack results on four different FPGA architectures for AES-128 encryption. Figure 3.6 shows HAC results **z**, $\mathbf{c}_l$, and $\mathbf{c}_u$ from these designs (further details are in the caption). Table 3.1 gives a comparison of the HAC slope (the mean slope of **z** in Fig. 3.6) with CPA trace complexity (MTD) and TVLA $\max(-\log(p))$ values, which are standards in the literature.

These four FPGA designs differ mainly in their method of obtaining S-Box values during the SubBytes step.

Table 3.1: Comparison of CPA mean trace count (MTD) for a successful attack, $t$-statistic (TVLA) $\max(-\log p\text{-values})$, and HAC slope (Alg. 4) values across AES implementations.

| Impl. | CPA MTD | TVLA | HAC† |
|---|---|---|---|
| LUT | 4K | 96.5 | 0.80 |
| PPRM1 | 30K | 30.7 | 0.57 |
| PPRM3 | 38K | 32.3 | 0.40 |
| HLS* | >100K | 11.7 | 0.10 |

* Design remains unbroken even after several attack types.

† This work. Value is the mean slope of the HAC plots in Fig. 3.6.

**LUT** stores the S-Box values in on-chip Block RAM. Each value is retrieved as-needed from memory. Retrieving key-dependent values from RAM is a well-known source of information leakage.

**PPRM1** implements positive polarity Reed-Muller AND-XOR logic to compute the S-Box values at run time.

**PPRM3** the same principles as PPRM1, but using three AND-XOR stages. Because this was designed for low-power systems, we should expect the signal-to-noise ratio (SNR) to be smaller (i.e. more noise) due to power efficiency of S-Box computations.

**HLS** was implemented in hardware via high-level synthesis from C++. It is pipelined and loops are all unrolled. S-Box values are stored in registers near the SubBytes functional units and accessed concurrently. It has very low SNR, and pipelining adds significantly to the noise.

### Trace Collection Protocol

We acquired voltage-drop traces using a National Instruments PXIe-5186 oscilloscope from a SAKURA-G evaluation board [86], at a sampling rate of 1 GHz for all FPGA designs. The minimum trace length is over 8k samples, and the max is 32k samples. Lengths vary to ensure that the entire AES computation is captured. Triggering is controlled by an independent controller FPGA on the SAKURA-G board.

HAC scores are computed via Algorithm 4 run on 2k traces, 1k in each of the sets $D_0$ and $D_1$. The set $D_0$ corresponds to the "fixed-key" and $D_1$ to the "random-key" collected according to the fixed-vs.-random key regimen of the data collection protocol released in a report [18] by RAMBUS.

The protocol defined in [18] specifies that one set of data are gathered with a fixed key, and one with a pseudorandom key, with pseudorandom plaintexts generated for each set. The reason that we do not use the the fixed-vs.-random input versions is because these do not necessarily reveal *key-dependent* information leakage. As demonstrated in Table 3.1, our method is predictive of attack complexity when using the fixed-vs.-random key protocol.

As we mention in Section 3.6.2, tests using groups with differing inputs (plaintexts) instead of keys do so under the assumption that you cannot conduct an attack unless plaintexts significantly alter the trace. However, consider that Gaussian template attacks [39] require *no* knowledge of the plaintexts used in the attack. For the purposes of constructing Gaussian templates, inputs are considered "noise" and generated pseudorandomly.

## 3.10 Conclusion

In this work we demonstrate the value of holistic SCLA testing. We additionally lay out criteria for an ideal holistic test and develop a nonlinear, nonparametric statistical metric, HAC, within this framework. To evaluate our approach, we test our technique on traces from four AES implementations and show successful vulnerability detection in accordance with both $t$-test and CPA attack results. As a bonus, our technique can be applied quite generally to assess the difficulty of massively multi-class high dimensional classification problems, and is novel to the best our knowledge.

It is our hope that this work will inspire further research, and lead to certification efforts examining the benefits of incorporating holistic testing in standardized SCLA procedures.

Chapter 3, in part, is currently being prepared for submission for publication of the material. Althoff, Alric; Blackstone, Jeremy; Kastner, Ryan. The dissertation author was the primary investigator and author of this material.

# Chapter 4

# Identifying and Mitigating Temporally Nonuniform Information Leakage

## 4.1 Introduction

Power analysis attacks are surprisingly effective, and because power is a required resource for all computer systems developing countermeasures is difficult. It may seem counter-intuitive that small series of bit transitions performed by an algorithm running on a microprocessor could be reliably detected from its power consumption without unfathomably precise equipment, yet they absolutely are. Such a feat is possible, at a high level, through two observations: 1) the attacker need only find differences that are correlated with data that they are looking for, and 2) the attacker can force the system to use the data they are looking for while the rest of the system activity generates "noise". Thus system activity is either correlated with the data we are looking for (in which case that activity is useful to the attacker), or uncorrelated with the data, in which case averaging across many runs will remove the noise. This is an oversimplification of power analysis attacks, but gives some intuition as to why it is so hard to develop robust countermeasures.

During a power attack an attacker will typically target a time point that both leaks information and from which it is easy to derive bits of the true key. While many algorithms have well-known easy attack points, the amount of information leakage is the dominant factor in attack success. However, not every point in time in a trace has the same amount of useful information. In analyzing real power traces, we have found that the amount of useful information varies considerably as a function of time. While there are many different prior approaches that attempt to mask an entire trace, hide such a trace under noise, or simply be resilient in the face of leaks which we discuss in Section 4.2, in this chapter we will present a first attempt to exploit the measured non-uniformity of leakage over time using a *programmable* electronic countermeasure.

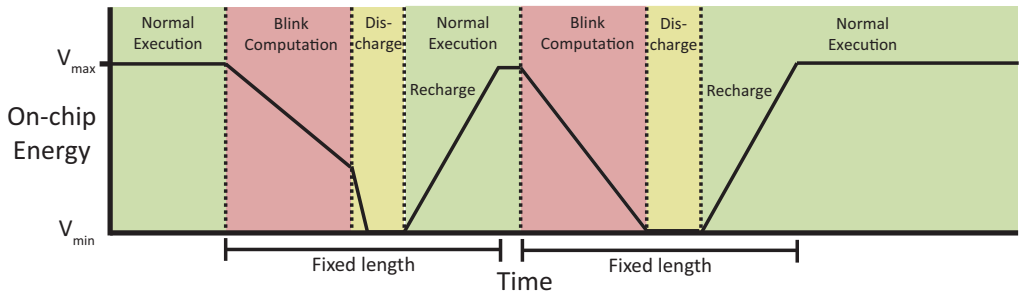


Figure 4.1: A depiction of a computational blink. When a blink computation is invoked, three phases occur serially: 1) The blink computation begins and lasts as long as the maximum time needed for the worst case execution time of the code using the allotted energy. 2) A fixed "discharge" time occurs, regardless of the on-chip energy source having been drained or not. During this time, the energy source is emptied. 3) a fixed "recharge" time occurs and the on-chip energy source is filled back to its original capacity. The length of time for the blink computation, the discharge, and the recharge is fixed to insure that the computation does not leak any information.

We begin with a discussion of power traces, the threats they introduce, and how the time-varying nature of their information leakage can be exploited. Section 4.3 introduces a rigorous method by which we can accurately quantify that time-varying information leakage and find the most useful points to target for removal. To edit those high points out, in Section 4.4 we introduce the hardware details of our approach which dynamically connects and

disconnects to power according to a static (but software controlled) schedule. We evaluate our approach across a variety of metrics and cryptographic algorithms in Section 4.5 and relate our contributions to prior work in Section 4.2 before summarizing in Section 4.6.

## 4.2 Related Mitigation Strategies and Metrics

The overriding goal of power analysis countermeasures is to make the energy consumption independent of the secret bits. The vast majority of proposed countermeasures attempt to reduce the power trace signal and/or increase the noise in the system, i.e., they attempt to lower the SNR. An ideal system would have a SNR of zero, which would make the system obey Equation 3.15 and 4.1, but this is difficult to achieve in practice. There are many mitigation techniques that we broadly classify into masking, hiding, and electronic countermeasures. Most of these works are complementary to ours, e.g., you could use masking and hiding techniques in addition to computational blinking. The most closely related work is electronic countermeasures. Next, we summarize related work in these categories.

### Masking

Masking-based approaches conceal intermediate values by the secret key though an XOR with random values—a "mask" that is assumed to be unknown to an attacker—that changes each time the algorithm is executed. Pioneered in several works [87, 88, 21], these approaches are the most commonly implemented. This is because they are purely algorithmic, software-based, solutions and so they require no additional specialized hardware modifications to run on general purpose processors. They also come equipped with strong proofs of security against first order attacks (i.e. attacks using only a single point in time) [87, 35], and with increased computational cost, attacks of arbitrary order [88].

There are a few serious downsides to masking. The first is that because masking mod-

ifies the cryptographic algorithm it cannot be directly applied to existing *hardware* implementations; these would require a redesign that is costly or unfeasible in many cases. Additionally, and more seriously, there is a logical consequence to to this algorithmic solution: the mask is, essentially, a single-use "nonce" that also affects power consumption when operations are carried out using it. Popular attacks on masked implementations first determine the mask through power analysis, and then proceed normally with the power attack, or use multivariate attacks [7]. Higher order masking approaches have longer mask lengths, and so prove to be more challenging, but the unfortunate result is that masking only makes an attack more frustrating, perhaps significantly so, but in the end, does not offer complete mitigation. As an example, the DPAv4.2 traces come from a first-order masked AES implementation running on a microcontroller. This protection has not proved particularly effective, and many successful attacks exist [77].

**Hiding**

Defenses in this category include randomly inserting dummy operations or cycles [89], shuffling the operations of the algorithm, changing the behavior of the clock and power signal [90], using multiple clock domains, and redundant logic styles to equalize transition probabilities [91, 92, 93]. While all of these techniques may cause the attacker to work harder, they do not eliminate the threat. Hiding defenses only moderately increases the number of measurements to disclosure (MTD) [94, 95]. The signal still exists and formal lower bounds on the number of traces can be determined [96, 64].

Other hiding approaches work at the logic/standard cell level to ensure that all computations take the same amount of energy regardless of the input data. For example, Tiri et al. [34] propose a number of different dual rail logic styles that equalize the current of rising and falling transitions by ensuring that every logical '0' to '1' transition has an equivalent '1' to '0' transition (and vice versa). They present the design of an AES core in 18 um technology.

This methodology increases the complexity of the secret key extraction to 20,000 runs. Yet, it requires three times more area, four times more power, and four times increase in execution time over a similar "unprotected" AES core. Additionally, it requires a significant change to the design flow. Unfortunately, this approach very sensitive to process variation and place and route differences that can cause very small, but exploitable timing differences [97]. In the end, this means that while these and related approaches are ineffective at completely removing leakage from critical areas.

**Electronic Countermeasures**

These approaches aim to modify the power supply during the computation. For example, adding a filter between the internal power supply and the output pin reduces the bandwidth of the signal making it more difficult to attack. However, this was shown to be ineffective and in fact made the attack easier by further differentiating the time at which the tested hypothesis occurs [98]. Another approach is to insert active equalization circuitry to ensure that the power supply signal stays constant. Ratanpal et al. [99] describe a suppression circuit that reduces low frequency current fluctuations and a low-pass filter to remove high frequency variations. Corsonello et al. [100] propose a charge-pump circuit using on-chip capacitors. Muresan et al. [101, 102] propose a current flattening technique which maintains a constant current draw from the power supply. Active equalization is difficult in practice as computation can rapidly draw current requiring fast compensation to mitigate any visible power spikes. Shamir [103] proposed the use of two switched capacitors, which go through a cycle of charging from the power supply and then providing energy to the device. Each capacitor provides power to the device for approximately tens of microseconds requiring a capacitor on the order of microFarads. Tokunaga and Blaauw [104] extend this approach by using a switched capacitor current equalizer block to isolate an AES encryption core from the power supply line and switching at a rate 5 times faster than the clock. They implement the modified AES core using

a 130 nm design node running at 100 MHz at 1.2V. This core had a 25% area overhead, a 33% power overhead and $2\times$ decrease in performance. This normalization technique is effective for the AES core where activity factors are highly stable at 50%. This approach is quite effective—ten million power trace measurements showed limited correlation between that of the correct and incorrect keys, far less than the unprotected core showed with 10,000 power traces. Regardless, even if the specifics of the protection circuitry are changed, the ability to apply those countermeasures intermittently and under control of software opens up new possibilities for optimization and trade-offs as we have discussed.

**Side channel leakage metrics**

These aim to quantify the vulnerability of the hardware against an attack. Side-channel vulnerability factor [105] provides a metric to quantify the difficulty of exploiting a side-channel by correlating between ground truth patterns and attacker observed patterns. While they state that their techniques are more broadly applicable, they only perform evaluation on cache timing channels. CC-Hunter [106] detects timing channels by dynamic tracking conflict patterns, e.g., bus locks and functional unit contention. SAVAT [107] is a fine-grained side channel leakage metric that considers differences instruction variations due to branching or cache misses. However, all of these metrics focus on timing channels while our metric (Section 4.3.2) focuses on power consumption. Many other power side-channel security metrics exist for selecting vulnerable points, but these are univariate or require time points to be chosen a priori. We will discuss this in more detail in Section 4.3.2.

### 4.2.1 Threat Model

We assume an attacker can cause security critical programs to run with arbitrary inputs and collect detailed power traces. For example, the attacker could write a program that would call a library to encrypt the data of their choosing while gathering current measurements over

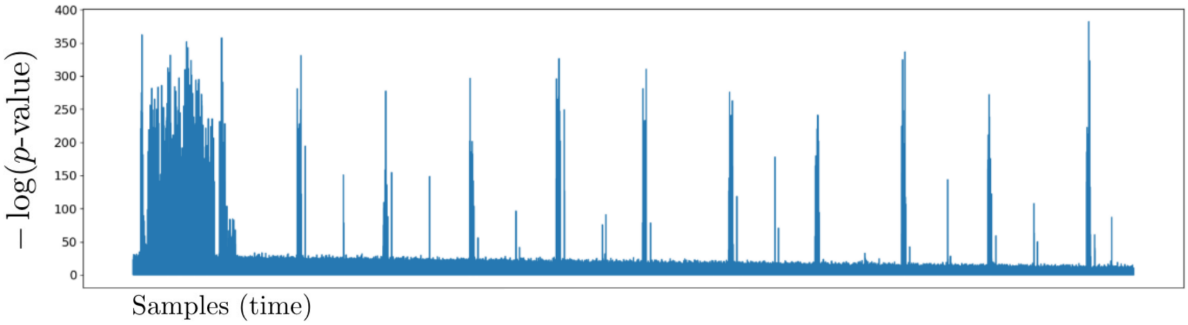


Figure 4.2: Vulnerability of AES over time (extracted from power traces). The $x$-axis is a unit of time and the $y$-axis is a measure of the leakage. Specifically the $y$-axis is the $-\log(p\text{-values})$ from a TVLA analysis of power traces. Larger values indicate that a location is more likely to leak information due to power variations.

time. We assume that the attacker has the ability to synchronize the power supply signal with the computation, i.e., she knows precisely when the start of the computation occurs (perhaps using simple power analysis [71]), and can synchronize multiple traces of an encryption as is assumed in most power analysis attacks. We make no assumptions on the equipment used to record the power traces, i.e., the signal acquisition equipment could have arbitrarily high sampling rate and resolution and/or the attacker could have a detailed power simulator. We note that relatively low-cost equipment is often used for power analysis attacks [63], but our mitigation strategy is effective regardless of how the power traces are collected. Our techniques focus on mitigating power analysis. Attacks and other side channels (timing, acoustic, RF, etc.) are outside of our threat model, although the analysis technique we present may be more broadly useful. In essence, we assume that the attacker can obtain power traces that have similar or better quality than those in the DPA contest [108].

### 4.2.2 Leakage Non-Uniformity

Figure 4.2 shows an example of leakage over time using real power traces by running AES on an AVR microcontroller. Note that we are *not* showing the power trace directly, but rather the degree of leakage as measured by a $t$-test according to the Test Vector Leakage

Assessment (TVLA) by Cryptography Research Inc. [109]. Larger $-\log(p\text{-values})$ from the $t$-test indicate samples in time with larger average power difference for differing data, i.e. samples that leak more information from the algorithm via the univariate mean.

The $t$-test determines the probability that the means of two Gaussian distributions are equal, and it assumes that the data is drawn independently from a Gaussian distribution. If the means are not equal, then this indicates that there could be key-dependent leakage at this location in time. The plot shows the $-\log(p\text{-values})$ of the $t$-statistic that the means are equal. High values indicate a greater confidence that there is a difference between the means, and therefore indicates a greater leakage of key information. Values of $p < 0.00001 \implies -\log(p) > 11.51$ are vulnerable according to the TVLA-recommended threshold. Note that this threshold is not adjusted for the length of the traces, and so it is a heuristic rather than the true probability of a false rejection of the null hypothesis.

It is clear from the plot that leakage over time varies radically. If we can protect those points in time that leak the most information, then we can increase our security without employing more costly protection mechanisms that cover the entire execution of the crytographic algorithm. There is one important thing to make clear—the $t$-test is only *one of myriad possible schemes* to test for secret information leakage from traces [20, 109, 110, 19, 111, 73]. The questions remaining for Section 4.3 are why should we make another method, and how can we make sure our approach is at least as powerful as *any* such scheme. However, before we get into that we will discuss what we are going to do with that information at a high level.

### 4.2.3 Blinking as a Method to Exploit Non-Uniformity

To take advantage of the observation of leakage non-uniformity as a countermeasure to power SCA, we introduce *computational blinking*—a technique that intermittently removes the power consumption for parts of an algorithm from the observation of the attacker. We draw inspiration from a "blink", i.e., the rapid closing of an eyelid. The average person blinks

15-20 times per minute [112] for a duration between 100-400 ms [113]. Thus, we spend between 2.5-13.3% of our waking time with our eyes closed due to blinking. These spontaneous blinks occur at natural breakpoints when our visual attention is least needed, e.g., during a pause when listening to a speaker [114] or at a scene change in a video [115]. Computational blinking aims to perform a similar activity: to blink during times that the algorithm leaks the most critical information, yet do this in a manner that minimizes the impact on performance.

During a computational blink, or period of power disconnection, the computation should not draw power from an external (and thus measurable) energy source. Instead, the computation uses a source of on-chip energy, e.g., an on-chip bank of capacitors. To mitigate the energy storage overhead of this scheme, blinks would ideally be very short. Because of this, blinks must be applied with great care if there is any hope of effectively hiding the sensitive operations from an attacker. For example, if we were to blink randomly, the attacker would be able to, in effect, remove the blink just as they could for any other uncorrelated noise; by collecting more traces.

Any blinking approach must bring together three elements: an analysis to find the points in the trace where one should blink, a method of implementing disconnection in the hardware, and an extension to the system that allows for such information to be passed from the software to the hardware as a blink schedule. This general framework enables programmers and system designers to perform a computational blink and mask intermediate energy usage over a fixed amount of time, either eliminating or greatly reducing the information leakage. While there are many details to such a system that we discuss in depth later in the paper, we will begin with a high level description of a blink.

Figure 4.1 shows the high-level idea of the execution of two computational blinks for a small security core built into a larger system on chip (SoC). At the beginning of execution the security core is connected to the same shared power system as the rest of the design. As a software-determined static schedule forces the system into a blink, the power is discon-

nected and the security core starts to draw down the energy in the capacitors until it reaches $V_{min}$—the minimum tolerated operating voltage specified for the security core. This first blink computation draws down some, but not all, of the energy from the on-chip energy source. The computation is followed by a discharge time that dissipates the stored energy to a known, fixed, minimum level. The energy in the capacitors is then built back up during normal execution. The second blink computation in the example happens to use all of the energy from the on-chip energy source. However, the discharge time must still be incurred to avoid adding a new timing channel.

In the end, a blink presents an abstraction under which a fixed energy and time budget is allocated and where the computation will take no more and *no less* than the amount to which it has been allocated. The energy is emptied to a fixed level—otherwise the amount of time and energy for the recharge will vary and this could leak information. The schedule is determined before execution and not secret-data dependent, i.e., being able to detect the schedule does not provide any additional information. The major advantage of this combination is that a *complete* lack of variance in the signal measured at the output means zero bits of Shannon entropy and thus, no leakage of information. However, before this idea can be fully exploited we need have a new understanding of exactly what it means to partially remove information from a trace this way.

## 4.3 Quantifying and Exploiting Time-Varying Information Leakage

For blinking to be feasible, we need to know how to determine which regions of the trace are most useful to an attacker, and how secure a system would be if we could remove them. Ideally the answer to such a question is not tied to a specific attack, but rather says something more fundamental about the nature of the amount of information remaining in the

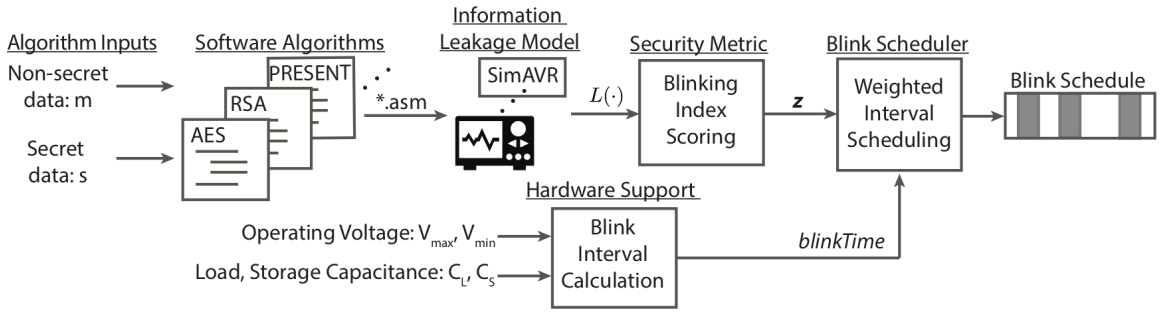


Figure 4.3: An overview of the process to determine the best blink schedule for an algorithm. The algorithm inputs are labeled as secret and non-secret and the algorithm is analyzed to determine its leakage $L(\cdot)$ either by collecting power traces or using a model. This is given to a security metric (Section 4.3.2) that outputs a score $\mathbf{z}$ indicating the best places to blink, i.e., the times with the most leakage. A blink scheduler (Section 4.3.3) takes this data and information from the hardware (Section 4.4) that determines the blink time(s) and outputs an optimal blink schedule.

trace that an attacker can learn. While one is very unlikely to get to the point that side-channel attacks are *provably impossible* in practice, we can capture this notion by developing a criteria that covers learning-based attacks that have yet to be realized. In this section we outline our blinking approach as summarized in Figure 4.3. This requires a sound information leakage model (Section 4.3.1), a security metric to measure the "leakiest" moments in time (Section 4.3.2), and a blink scheduling algorithm (Section 4.3.3).

### 4.3.1 Formalizing Leakage and Security Criteria

At a high level, our main assumption is that if a system is secure against power SCA then it is impossible for an attacker to *differentiate* between *different secrets* given sets of measurements. One way to do this is to ensure that the measurements with differing secrets are always equal. Another is to ensure that they are random noise that is completely unassociated with the secret. If they are equal, then looking at one power trace is equivalent to looking at any other, if they are random, this is also (statistically) true, and there is a spectrum of scenarios mixing equality and randomness where security is maintained. This implies

that measurements could be noisy functions of one another and still maintain the security of underlying secrets. Metrics that only test *univariate* statistical independence between traces and secrets, such as [19] without multivariate combining, will not be powerful enough for blink scheduling in general, because single points in time may be independent of the secret when taken alone and yet, as we show in the XOR example in Section 4.3.2, when combined may yield the secret with little effort. This is one of the reasons why we need to develop a mathematically sound security criterion and algorithms in this work.

First, we develop some notation that we will use to formalize our intuition. Measurements are a function $L(\cdot)$ of processor behavior. In power SCA, $L(\cdot)$ would be measurements of voltage drop recorded with an oscilloscope. In general, $L(\cdot)$ could be physical measurements, or simulations from a model that are correlated with measurements. The power side-channel analysis community refers to such measurements as *leakage*, and we will do the same. We take the convention that $L(\mathbf{t}, \cdot)$ is a vector of leakage values at sample times $\mathbf{t}$. Under our threat model we run processes using data $m$ that are assumed to be known to attackers, and data $s$ that is secret—for example, $m$ could be a plaintext message and $s$ a secret key in a cryptographic algorithm—so our leakage function also includes those latent variables, and becomes $L(\mathbf{t}, m, s)$. Per our convention, $L(\mathbf{t}_i, m, s)$ is the single value measured—or *leaked*—at time $i$, while $L(\mathbf{t}, m, s)$ is a vector of leakage values (a power trace), at all times $\mathbf{t}$ for a fixed non-secret and secret. Likewise $L(\mathbf{t}, \mathbf{m}, \mathbf{s})$ is an order three tensor—a multidimensional array—of leakage values for multiple times, multiple non-secrets, and multiple secrets

Intuitively, a system would be invulnerable to a leakage-based analysis if secrets have no effect on the measurements at all. Such as system is "unlearnable" to the attacker as they can observe no difference in traces when secrets differ. Likewise, we might incorrectly assume that if the traces are random and have the same distribution for any pair of underlying secrets, then it is also impossible for an attacker to succeed. However, this is not enough to guarantee security. For intuition as to why this is so, consider that a list in random order and a sorted

copy of that list are both identically distributed in terms of their values, and so their histograms would be equal, yet the *ordering* of the information allows us to reliably tell them apart.

For this reason, it must be necessary for security that

$$L(\mathbf{t}, \mathbf{m}, \mathbf{s}) \stackrel{d}{=} L(\mathbf{t}, \mathbf{m}, \mathbf{Ps}), \forall \mathbf{P} \tag{4.1}$$

where $\mathbf{P}$ is a permutation matrix, and $\mathbf{s}$ and $\mathbf{m}$ are vectors of all messages and all secrets respectively, and $\stackrel{d}{=}$ indicates equality in probability distribution. This criteria is known in statistics as *exchangeability* [84], and implies that the joint distribution leakage is unaffected by reordering secrets.

Eqn. 4.1 is a very general security criteria, and useful even if the space of secrets is small enough to make a brute-force search easy. If a system with leakage $L(\mathbf{t}, \mathbf{m}, \mathbf{s})$ obeys Eqn. 4.1, then we cannot do better than a random guess at the value of $s$ for the leakage function $L$ for a given device and measurement setup. Unfortunately, verifying Eqn. 4.1 for all permutations requires $O(n!)$ multivariate hypothesis tests! We must clearly tackle this problem approximately, and so we take the Monte Carlo approach explained in Section 4.3.2.

### 4.3.2 Measuring the Leakiest Regions of a Trace

Now we turn our mathematical intuition into a security metric and blink window selection algorithm. Our statements here are without regard for implementation. In Sections 4.4 and 4.5 we will consider practical constraints along with their security and performance ramifications.

In order to make Equation 4.1 hold for more, and ideally, all, subsets of times in the trace, we would like to blink at times that contribute the most to making traces "unexchangable" with respect to different keys. If Equation 4.1 does not hold, then we should be able to build a model of groups of power traces with differing keys, and use this model to clas-

sify traces with an accuracy consistently better than chance. We can now see that identifying the leaky time intervals corresponds closely to the data analysis problem of *feature selection*. The goal of feature selection is to identify a set of properties, or indices of data vectors, that consistently contribute as much information as possible about a datum's associated class. If we consider classes to correspond to secrets, a good feature selection approach will determine which set of indices $\mathcal{B}$ allow us to differentiate among secrets, then we can blink them out, along with all redundant features.

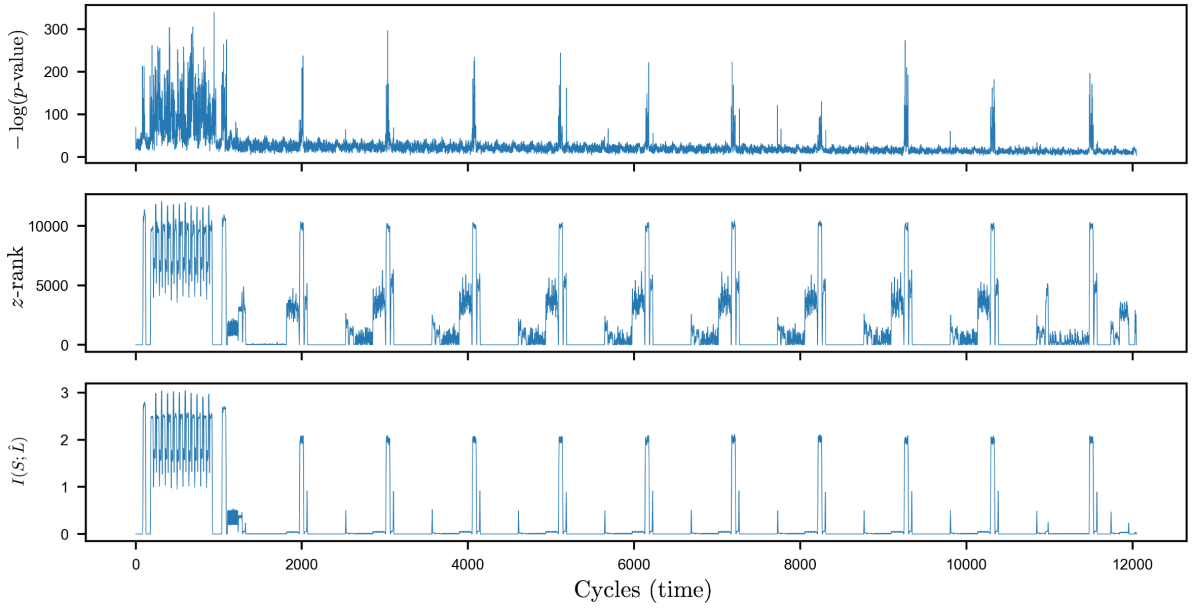


Figure 4.4: A visual comparison of TVLA $-\log(p\text{-values})$ (top), $z$-rank (center—our method), and mutual information (bottom) vulnerability detection schemes. Our approach correctly determines both invulnerable regions—which TVLA does not—redundant regions, and regions that are complementary—which univariate mutual information does not.

For our choice of a baseline feature selection metric, we turn to [116], which includes an empirical study determining which of many modern information theoretic feature selectors score well over several trade-offs. Based on this study, the *Joint Mutual Information* feature

selector (JMIFS) [117, 118] is a balanced choice. The JMIFS is defined for a time index $i$ as

$$\text{JMIFS}(i) = \sum_{j \in \mathcal{B}} I(L(\mathbf{t}_i, \hat{\mathbf{m}}, \hat{\mathbf{s}}) \frown L(\mathbf{t}_j, \hat{\mathbf{m}}, \hat{\mathbf{s}}); \hat{\mathbf{s}}) \tag{4.2}$$

where $x \frown y$ concatenates $x$ and $y$, $\hat{\mathbf{m}}$ and $\hat{\mathbf{s}}$ are vectors of messages and secrets chosen independently and uniformly at random, $\mathcal{B}$ is the set of time indices already chosen to blink, and $I(\cdot;\cdot)$ is the mutual information between variables. Mutual information is a measure of association between its arguments, and we will describe it with more detail in Section 4.5.

We select features using the JMIFS criteria recursively: The first index selected will be the time point in a trace with the maximum mutual information with the secret, we add this to the set of indices to blink $\mathcal{B}$, and remove it from the set $\mathcal{B}^c$ of remaining indices. We select the next index as the one that maximizes Eqn. 4.2, and so on, until there are no more indices to test, i.e. $\mathcal{B}^c = \emptyset$. We cache the values $\mathbf{J}_{ij} = I(L(\mathbf{t}_i, \hat{\mathbf{m}}, \hat{\mathbf{s}}) \frown L(\mathbf{t}_j, \hat{\mathbf{m}}, \hat{\mathbf{s}}); \hat{\mathbf{s}})$ for use in later steps.

Our feature selection problem has an idiosyncrasy specific to security; redundant time indices present other—equally strong—attack vectors. However, feature selection algorithms are designed with a bias against selecting redundant indices. In order to derive a good blinking criteria from JMIFS, we group the features with respect to their mutual redundancy, and re-score them accordingly. Specifically, if $\mathbf{J}_{ij} = I(L(\mathbf{t}_i, \hat{\mathbf{m}}, \hat{\mathbf{s}}) \frown L(\mathbf{t}_j, \hat{\mathbf{m}}, \hat{\mathbf{s}}); \hat{\mathbf{s}})$ computed during an ordinary JMIFS run, are equal, up to numerical error, to the mutual information between leakage at index $i$ and the secret values, then leakage at $j$ is redundant given $i$. Our algorithm constructs a Boolean matrix $\mathbf{R}$ such that $\mathbf{R}_{i,j} = 1$ if time $i$ is redundant with time $j$ and $\mathbf{R}_{i,j} = 0$ if it is not. Once we have identified the redundant indices, we re-score those time indices in the same redundant set with a group label according to their selection order by JMIFS. We use this group label as a ranking so that redundant indices are *all* given the "worst"/maximal score from among their redundant group. This means that more leaky

regions have higher ranks, and regions that have more redundancy will have a larger gap between themselves and regions ranked immediately below them than will regions with smaller redundant sets. We do not weight the ranking in this work but this is certainly possible to do, and could be used to place greater importance on particular regions, or prioritize easy attack vectors to ensure they are blinked out.

An important feature of JMIFS is that it considers combinations with the other times we've selected so far. This is a way of handling *variable complementarity* [119]. The following is an example: if Boolean variables $x_1$ and $x_2$ are statistically independent, then $x_1 \oplus x_2 = y$ is independent of each individually, implying $I(x_1; y) = I(x_2; y) = 0$, however, $I(x_1 \frown x_2; y) > 0$ because $x_1 \frown x_2$ completely determines $y$. JMIFS detects indices leaking secret information even if they have an XOR-type relationship and when weighted by the mutual information sets it to a high priority, while moment-based tests, or those relying on univariate statistical independence, are insensitive to this case.

We have found via experimentation that complementarity exists in power traces, and attacks such as [76, 38] exploit it to powerful effect. In this way our selection criteria is superior to other SCA security evaluation metrics [20, 109, 110, 19, 111, 73] that consider the power samples taken singly, or involve the use of combining functions[1] or multivariate histograms for only a few time indices. In any case, we cannot credibly assume a lack of complementarity of $L(\mathbf{t}_i, m, s)$ with $L(\mathbf{t}_\mathcal{B}, m, s)$, for arbitrary $i$ and a set of time indices $\mathcal{B}$, due to the latent factors $m$ and $s$ which are held constant during a run of the underlying security-sensitive algorithm. An analysis using a univariate metric can therefore only reveal how vulnerable a design is *at minimum*. This further motivates our decision to use the JMIFS criteria. We emphasize that it is not sufficient to consider time indices one-by-one when determining whether to blink them out.

[1]Combining functions often violate the assumptions of underlying statistical hypothesis tests, and are often derived from attack methods with particular assumptions themselves. $p$-values from tests using combining functions should be treated as heuristic unless this is explicitly corrected.

**Algorithm 5:** Blinking Index Scoring

**Input:** experimental keys $\hat{\mathbf{s}}$ and power traces $L(\mathbf{t}, \hat{\mathbf{m}}, \hat{\mathbf{s}})$ for experimental plaintexts $\hat{\mathbf{m}}$
**Output:** a vector $\mathbf{z}$ of scores ranking time points $\mathbf{t}$ by vulnerability to attack
**while** $\mathcal{B}^c \neq \emptyset$ **do**
  **foreach** $i \in \mathcal{B}^c$ **do**
    **if** $g^* < \mathit{JMIFS}(i)$ **then**
      $i^* \leftarrow i$
      $g^* \leftarrow \text{JMIFS}(i)$
    **end**
  **end**
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{i^*\}$
  $\mathcal{B}^c \leftarrow \mathcal{B}^c \backslash \{i^*\}$
  $\mathbf{g}_{i^*} \leftarrow g^*$
**end**
// $\mathbf{J}$ is already computed in previous steps
Let $\mathbf{J}_{i,j} = I(L(\mathbf{t}_i, \hat{\mathbf{m}}, \hat{\mathbf{s}}) \frown L(\mathbf{t}_j, \hat{\mathbf{m}}, \hat{\mathbf{s}}); \hat{\mathbf{s}})$
**foreach** $(i, j) \in |\mathcal{B}| \times |\mathcal{B}|$ **do**
  $\mathbf{R}_{i,j} = \begin{cases} 1 & \text{if } |\mathbf{J}_{i,j} - I(L(\mathbf{t}_i, \hat{\mathbf{m}}, \hat{\mathbf{s}}); \hat{\mathbf{s}})| \leq \varepsilon \\ 0 & \text{otherwise} \end{cases}$
**end**
$\mathbf{z}$-rank $\leftarrow$ rank of max $\mathbf{g}$ in each redundant set indicated by $\mathbf{R}$
// (Optionally) weight the scores
$\forall i \in [n], \mathbf{z}_i \leftarrow W_i(\mathbf{z}_i)$

Algorithm 5 describes the blinking index scoring algorithm in full, which returns a vector $\mathbf{z}$ so that $\mathbf{z}_i = \mathbf{z}_j$ means that those time indices $i$ and $j$ are equally vulnerable to attack, and $\mathbf{z}_i > \mathbf{z}_j$ means that $\mathbf{z}_i$ provides more information about the secret than $\mathbf{z}_j$.

### 4.3.3 Turning Measurements into Blink Regions

Now that we have a way to score the leakage of different time indices in a traces, we must determine the best blinking schedule. After executing Algorithm 5, $\mathbf{z}_i$ contains the leakage ranking for all time points under the given measurement/simulation setup. The problem of selecting globally optimal locations to begin a blink can be reduced to a *weighted interval scheduling* (WIS) problem. It can be solved optimally in $O(n \log_2 n)$ operations where, in our

case, $n$ is the number of samples in the measurement/simulation vectors [120].

**Algorithm 6:** Blinking Window Scheduling

**Input:** length-$n$ vectors $\mathbf{t}$ and $\mathbf{z}$, of times and scores respectively, and constants $blinkTime$ and $recharge$
**Output:** The blinking schedule $\mathcal{O}$ with optimal coverage
**foreach** $i \in [n]$ **do**
    $start \leftarrow i$
    $end \leftarrow i + blinkTime + recharge$
    $score \leftarrow \sum_{j=start}^{end-recharge} \mathbf{z}_j$
    $\mathbf{w}_i \leftarrow (start, end, score)$
**end**
**foreach** $i \in [n]$ **do**
    $\mathbf{prev}_i \leftarrow j$ s.t. $\mathbf{w}_j.end - \mathbf{w}_i.start$ is $\leq 0$ and minimal
**end**
**foreach** $i \in [n]$ **do**
    $\mathbf{g}_i \leftarrow \max(\mathbf{w}_i.score + \mathbf{g}_{\mathbf{prevs}_i}, \mathbf{g}_{i-1})$
**end**
$i \leftarrow n$
$\mathcal{O} \leftarrow \emptyset$
**while** $i > 1$ **do**
    **if** $\mathbf{w}_i.score + \mathbf{g}_{\mathbf{prevs}_i} > \mathbf{g}_{i-1}$ **then**
        $\mathcal{O} \leftarrow \mathcal{O} \cup \{\mathbf{w}_i.start\}$
        $i \leftarrow \mathbf{prevs}_i$
    **end**
    **else**
        $i \leftarrow i - 1$
    **end**
**end**

Algorithm 6 shows the WIS algorithm used to optimally derive the blinking window schedule. The algorithm accepts as input the vectors $\mathbf{t}$ of time indices available for blinking and $\mathbf{z}$ of corresponding leakage scores from Algorithm 5. Additionally, it requires the constants $blinkTime$ and $recharge$, which are dependent on the underlying blink-enabled hardware discussed in Section 4.4. Using $\mathbf{t}$ as the available times, and $\mathbf{z}$ as weights, Algorithm 6 solves the WIS problem to determine the best locations to blink so that the sum of $\mathbf{z}$ in non-blinking regions is minimized. This approach is globally optimal in the sense that it will

select locations in the program to begin a blink such that the sum of the scores covered by the blinked-out regions is the maximum possible under the constraints.

## 4.4 Hardware Support for Blinking

Now that we have a new and powerful method of analyzing execution behavior to find the leakiest regions, we describe a proof-of-concept class of architectures that exploit that new knowledge. At a high level, the hardware support for blinking can be divided in three parts: 1) hardware support to switch the power supply for the security domain from the main power distribution network to an on-chip bank of capacitors, 2) the processor modifications to allow it control over its own power supply, and 3) the capacitor bank itself. The idea of using on-chip capacitance as a way to hide cryptographic information is well established both in theory and in practice. Our aim is to expand upon these ideas to show how to make this programmable to protect any general purpose computation.

One of the major issues with blinking is how to get enough capacitance on chip to completely power a small core. Shamir [103] first described the cycled use of switched capacitors, and more recently Tokunaga and Blaauw [104] proposed a switched capacitor current equalizer to isolate a fixed function block from the power supply line. The key insight we bring over these hardware-only approaches is that the information leakage over time in a trace is highly irregular. Instead of attempting to cover the entirety of execution with a large capacitor bank, we can instead blink during those parts of the computation that are most leaky, and thus effectively remove the power consumption during those times from attackers view. We do this in a manner that gives direct control to the programmer over the security-performance trade-off. However, to make this trade-off quantitatively we need to better understand the hardware-imposed constraints on our blinking schedule.

Another issue is determining a feasible *blink time*, i.e., the number of instructions

that can be executed during a single blink. For this, we need to know four characteristics of the hardware: load capacitance ($C_L$), storage capacitance ($C_S$), maximum operating voltage ($V_{max}$), and minimum operating voltage ($V_{min}$). The **load capacitance** $C_L$ is the capacitance per instruction; the amount of capacitance required to store enough energy for the core to execute a single instruction. The smaller the load capacitance, the more instructions we can execute during a blink. To minimize load capacitance, blink-enabled hardware should use low-powered processing cores that aim to minimize joules per instruction. The **storage capacitance** $C_S$ is the amount of capacitance available for the core to draw energy from during a blink. Once the blink has started, all of the energy required for the core to continue execution will come from the storage capacitance. The more storage capacitance available to the core, the more instructions that can be executed during a single blink. In blink-enabled hardware, empty sections of the die area can be filled with decoupling capacitance cells that will increase the storage capacitance available throughout the chip. The **maximum operating voltage** $V_{max}$ is the highest voltage the core will run at during a blink. This is the voltage at the start of the blink which we assume is the normal operating voltage of the core. As the core executes instructions during a blink, its operating voltage will decrease. Eventually, the core will reach a **minimum operating voltage** $V_{min}$ where the core can no longer execute any instructions. We define the time that it takes to move from $V_{max}$ to $V_{min}$ as the $blinkTime$.

Using these four characteristics, we can calculate the $blinkTime$ as:

$$blinkTime = \frac{2 \cdot \log\left(\frac{V_{min}}{V_{max}}\right)}{\log\left(1 - \frac{C_L}{C_S}\right)} \tag{4.3}$$

To allow the core to control when a blink happens, we add an extension to the core's ISA to allow the core to communicate with a *power control unit*. The power control unit is responsible for initiating a blink to begin secure operation, discharging the capacitor bank when secure operation is complete, and recharging the capacitor bank after a fixed amount of

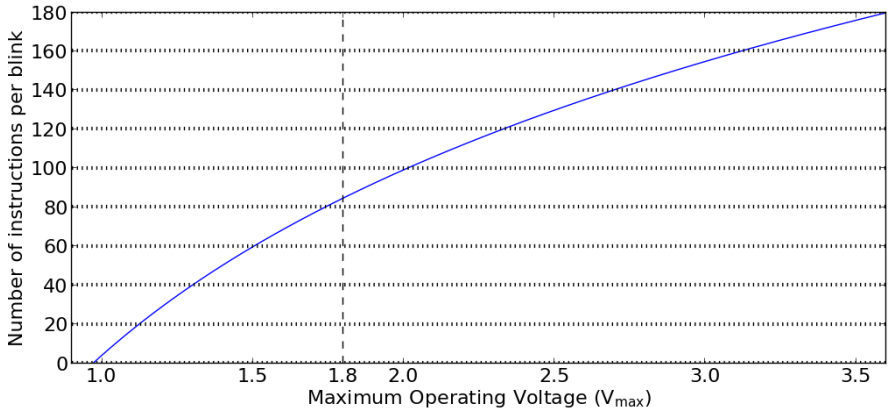


Figure 4.5: Graph of maximum possible instructions per blink as a function of $V_{max}$ in volts. The vertical dotted line shows the value available on the taped out RISC-V chip we tested. Note that as $V_{max}$ increases, so does performance due to increased maximum clock rate *outside* of blink-enabled regions. In blink-enabled regions, however, we must set the clock to the rate lower so that the chip will operate at voltage $V_{min}$.

time to resume non-secure operation (see Figure 4.6). When the core indicates it would like to start a blink, the power control unit will turn off the blink and recharge transistors, and disable input and output to the core, thereby electrically isolating the core and capacitor bank from the main power rails, as well as the input and output pins. The core executes self-sufficiently from its local scratchpad instruction and data memories. The power control unit will monitor the voltage across the internal power rails making sure that the voltage is never less than the minimum operating voltage $V_{min}$. If the core finishes secure operation before reaching $V_{min}$, the power control unit will activate a shunt resistor to discharge the capacitor bank to $V_{min}$. After a fixed amount of time from the beginning of the blink, the power control unit will start the recharge phase by turning on the recharge transistors. One concern during the recharging phase is the in-rush current observed when the depleted capacitor bank is reattached to the main power rails. A large in-rush current can corrupt the state of the core [121], therefore we place *recharge resistors* to limit the maximum in-rush current. Once recharged, the power control unit will turn on the blink transistors again to minimize IR drop across the recharge resistors.

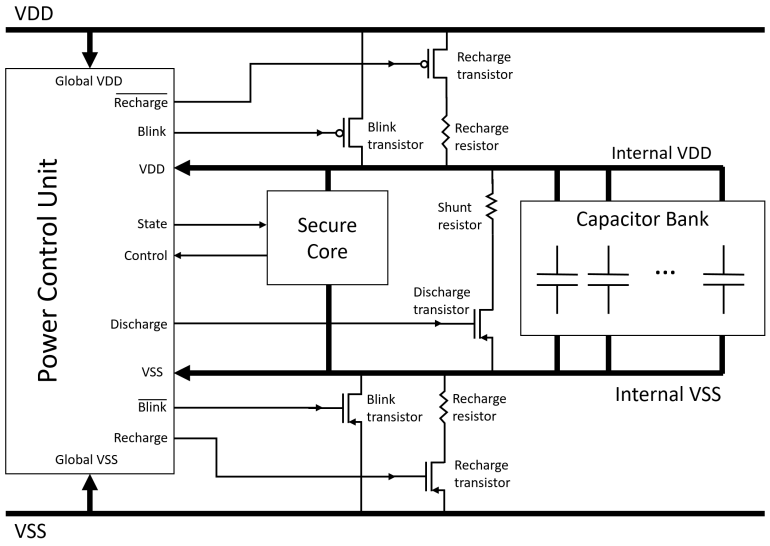


Figure 4.6: Block diagram showing the connection of the power control unit and capacitor bank with the secure core.

To demonstrate the potential for blink-enabled hardware in the real world, we evaluated a chip taped out in TSMC 180nm to determine what load capacitance, storage capacitance, and minimum operating voltage we could achieve. The chip contains a low power RISC-V processor, which we considered as our security processor. The core is a 32 bit, 5-stage pipeline, in-order processor running the RV32IM ISA with 4KB of instruction memory and 4KB of data memory and has an area of 1.27mm$^2$. To find the core's load capacitance, we simulated the chip in PrimeTime Suite using a switching activity vector representative of an average workload. We found the core had an average energy consumption of 515pJ per instruction running at 1.8V. Therefore we can calculate the required amount of capacitance needed to store 515pJ at 1.8V which is 317.9pF.

This chip contains full-custom decoupling capacitance cells to fill empty areas of the die. These cells make up a majority of the storage capacitance found on-chip. Using RC extraction, we found that these decoupling capacitance cells have a unit capacitance of 4.69fF/um$^2$. On-chip there was a total of 4.68mm$^2$ of the 25mm$^2$ die area filled with decoupling capacitance for a total of 21.95nF of storage capacitance.

The core's nominal voltage is 1.8V which we use as the core's maximum operating voltage. To find the minimum operating voltage, we put the chip on a motherboard with direct control over the chip's core voltage and clocking frequency. We continuously drop the core's voltage and clock frequency until there was no clock frequency where the core could still operate. This occurs at 0.97V, so that is the smallest value we can use for the chip's minimum operating voltage $V_{min}$.

Using Eqn. 4.3, we find that for this particular chip every 1mm$^2$ of decoupling capacitance allows the core to execute roughly 18 additional instructions per blink. The AES-128 implementation used in the DPA contest takes 12,269 cycles to run [108], including key expansion and overhead. If we were *not to pause for recharging*, then 12,269 cycles would require about 670mm$^2$ of decoupling capacitance, 528$\times$ more area than the core itself. This is why we require recharge periods when blinking, along with algorithms to schedule blinking time windows: Simply blinking the entire computation would require impractically large capacitive area.

## 4.5 Evaluation

We have laid out a general blinking framework and shown how to enable this in hardware. This section evaluates that framework's effectiveness. First, we provide details on framework itself, and then we use the framework to perform blinking on several applications. We show that blinking provides orders of magnitude reduction in leakage.

### 4.5.1 Blinking Framework

Estimating the power side channel vulnerability begins with data collection. This usually entails connecting an oscilloscope to the circuit under analysis, and running random or specially chosen inputs until thousands of power traces are recorded. However, it may be

unreasonable to expect a software engineer to collect these data each time they make modifications to their source code. Furthermore, it is not possible to collect power traces from an architecture that is under design. Thus, we developed a simulator of power side-channel leakage to help facilitate our evaluation.

Our power simulator leverages the open-source tool *SimAVR* [122]. SimAVR executes binaries compiled by the avr-gcc toolchain, and provides instruction-level access to program state. This means our simulated traces result from execution of the binary as it would be run on an AVR microcontroller, including all optimizations implemented by the compiler, instead of an analysis of the source code. Our SimAVR modifications implement the Hamming distance leakage model [65] used in nearly all CPA attacks. For each opcode the simulated power trace is the Hamming distance between the previous value in the target register (resp. memory location) and the new value to be written. Our modified tool outputs this Hamming distance value for as many cycles as the current opcode takes to execute on the particular target AVR chip before moving on to the next opcode in the compiled binary.

The Hamming distance model assumes that toggling a bit leaks/consumes one unit of power, regardless of other factors, and leaving a bit in its current state consumes no power. While this is a highly simplified model, it is nearly ubiquitous in the power SCA literature, (see [123, 65, 124, 111] among others,) due to its consistent correlation with power consumption and its effectiveness for correlation-based attacks. While other research has focused on modeling exact characteristics of the circuit in question, these have not been widely adopted, because 1) the Hamming distance model captures sufficient detail with a minimal effort on the part of the attacker, and 2) an exact model can be obtained from the system itself.

Optionally, our tool adds the Hamming weight ($\omega$) of the data moved by an instruction into the simulated power output. We have found that this better accommodates the effects of load and store instructions. This is because supplying electric charge to the buses and RAM cells of the memory system consumes power in proportion to the data itself, rather than the

bit-wise change in value. Together, we can write that our model output is, for prior state $x$ and current state $y$,

$$\text{Power Leakage Model}(x, y) = \hat{L}(x, y) = \omega(x \oplus y) + \omega(y) \tag{4.4}$$

It may be unintuitive that Hamming distance and weight can be used for a power-based analysis. However, power SCAs only require a model that is *consistently correlated* with the measured quantity in order to succeed. This is one of the reasons power SCA is so effective. In power SCA, the leakage function $f$ that we measure is usually not power directly, but voltage drop during an operation. The studies of [123, 65, 124], among others, empirically show that Equation (4.4) is robustly correlated with voltage drop across different devices.

### 4.5.2 Balancing Security, Energy, and Performance

In architecting a blinking device, we are faced with the task of balancing many related parameters. Clock speed is determined by the lowest operating voltage, which will occur at the end of a blink; the storage capacitance and minimum operating voltage determine the maximum blink length; longer blinks drain more voltage, affecting operating voltage (and thus clock speed) as well as energy consumption; blink sizes affect security, the cost of static scheduling, and performance.

The maximum blink window size is a function of the stored capacitance and the $V_{min}$ of the chip. Smaller blink lengths are also possible, meaning less capacitance will be drained and the chip can resume operation at $V_{op} > V_{min}$. We considered a range of storage capacitances from 5nF to 140nF (corresponding to 1 to 30mm$^2$ of decoupling capacitance) to better evaluate the design space. We assume that each instruction requires, on average, a load capacitance $C_L$ of 317.9pF (as computed in Section 4.4).

With the available blink sizes determined, the selection algorithm (Algorithm 6) can

analyze a set of scores from (Algorithm 5) and find the optimal coverage that minimizes global secret information leakage. The algorithm notably does not consider performance; this would require the algorithm to make trade-offs between performance and security, which we leave to the designers or as future work.

There is a constant switching overhead to disconnect and connect the core at the beginning and end of a blink. According to our simulations, disconnection can occur fully within 2 cycles, while power shunting and reconnecting happens in under 1 cycle. Real-world execution, fabrication thresholds, and unpredictable conditions can easily push this disconnect time much higher. In our design space explorations, we use a penalty of 5 cycles per blink. Energy is potentially wasted in every blink, as excess charge in the capacitor must be shunted to avoid leaking information. From our power simulations, the most energy-intensive instructions consume 1.6$\times$ the energy of an average instruction. Provisioning for the worst case, on average the extra 60% capacitance is wasted. Depending on the algorithm and voltage, this was between 5 and 35% in our simulations.

We have found that different algorithms (or implementations of the same algorithm) can have very different leakage characteristics, requiring different blinking strategies. In addition to real AES traces from the DPA Contest v4.2 [108], we simulate power traces for both 128-bit AES and PRESENT from AVR-crypto-lib using our modified version of SimAVR. For both implementations of AES, for the parameters we examined, there is not an optimal point with regard to both security and performance, leaving the space open to designers to choose the most appropriate configuration.

Each algorithm, for certain capacitance, voltage, and clock combinations, tends to have a few near-perfect outliers with respect to security, with varying degrees of slowdown. These points likely had an available window size that aligned well with the program's phase behavior, blocking out leaky portions such that critical instructions are not missed in the recharge period after the blinks. In most cases, small capacitors and shorter blink lengths give these best

results, allowing fine-grained control of which instructions are covered. Shorter but more frequent blinks, however, incur more switching over; for both AES implementations, these optimal points are around $2\times$ the stock execution time. In addition, provisioning the chip with a smaller storage capacitance may preclude optimal parameter selection for other algorithms.

### 4.5.3 Security Evaluation

Table 4.1: Information leakage after blinking for three different cryptographic programs. The metrics are the number of $-\log(p\text{-values})$ above the TVLA threshold (i.e., the number of vulnerable points for univariate attacks), the sum of the residuals of our leakage ranking $\mathbf{z}$, and $1-\text{FRMI}_{\mathcal{B}}$ (Eqn. 4.6). Traces for the avrlib AES and PRESENT programs were collected via simulation using a Hamming distance power model (Eqn. 4.4), while DPA is from power traces from the DPA Contest v4.2 [108]. The last two allow comparison as a fraction of total leakage, where $1-\text{FRMI}_{\mathcal{B}}$ is univariate, and $\sum_{i=1}^{n} \mathbf{z}_i$ is multivariate. That is, the "pre-blink" results of both are equal to 1. The $t$-test result shows that even in the worst-case, blinking provides an order of magnitude reduction in attack vectors.

| | AES (DPA) | AES (avrlib) | PRESENT |
|---|---|---|---|
| $t$-test # $-\log p >$ threshold | 19836 | 285 | 1236 |
| $t$-test post-blink | 342 | 0 | 141 |
| $\sum_{i=1}^{n} \mathbf{z}_i$ (Alg. 5) post-blink | 0.033 | 0.083 | 0.104 |
| $1-\text{FRMI}_{\mathcal{B}}$ post-blink | 0.012 | 0.011 | 0.140 |

To evaluate the security of blink-enabled hardware, we collect a set of $2^{14}$ traces from the leakage model described in Section 4.5.1 Eqn. 4.4 for experimental plaintext and key vectors, $\hat{\mathbf{m}}$ and $\hat{\mathbf{s}}$, respectively from the AES-128 and PRESENT ciphers from AVR-Crypto-Lib. We also evaluated our metrics on the physical power measurements provided for the DPA Contest v4.2 [108]. Then we compute three metrics of security on these traces before and after blinking. We use blinking patterns chosen by our scoring and scheduling pipeline from Section 4.3. We allow the scheduler to choose between using blinks with three, data independent, lengths—one large, and one of half and a quarter that size. Note that these differing $blinkTimes$ are placed statically according to the results of Algorithm 6, and are not data dependent. Additionally, as mentioned in Section 4.5.2, while different blink lengths

do not drain the capacitor equally, the shunt always ensures that the voltage reaches the same level at the end of a blink. This means that an attacker would measure uniform power draw during blink intervals.

Our evaluation metrics are our multivariate vulnerability metric $\mathbf{z}$ from Algorithm 5, the TVLA $t$-test, and fractional reduction in mutual information (FRMI). We have discussed $\mathbf{z}$ and TVLA $t$-test previously. The mutual information, discussed in more detail in Section 3.1.2, is suggested in [73] as DPA security metric, and computed as

$$I(S; L) = H(S) - H(S|L) \tag{4.5}$$

where $H(S)$ and $H(S|L)$ are the entropy and conditional entropy of random variables $S$ and $L$, with $\mathbf{s}$ and $L(\mathbf{t}_i, \mathbf{m}, \mathbf{s})$ being their respective realizations. Intuitively, this says that $I(S; L)$ is the expected reduction in our uncertainty about secrets given knowledge of leakage at a single point in time $i$. As noted by [125], this metric corresponds directly to the success rate of a univariate template attack—the strongest form of attack in the information theoretic sense [39].

In order to form a composite score, we compute the FRMI given blinking. That is,

$$\text{FRMI}_{\mathcal{B}} = \frac{\sum_{i=1}^{n} I(L(\mathbf{t}_i, \hat{\mathbf{m}}, \hat{\mathbf{s}}); \hat{\mathbf{s}}) - \sum_{i \in \mathcal{B}} I(L(\mathbf{t}_i, \hat{\mathbf{m}}, \hat{\mathbf{s}}); \hat{\mathbf{s}})}{\sum_{i=1}^{n} I(L(\mathbf{t}_i, \hat{\mathbf{m}}, \hat{\mathbf{s}}); \hat{\mathbf{s}})} \tag{4.6}$$

which gives us the relative decrease in mutual information after we blink. We also compute a comparable composite score, $\sum_{i=1}^{n} \mathbf{z}_i$, from our own metric.

Figure 4.7 shows the results before and after applying the blinking procedure on the same trace from Figure 4.2. This visually shows that blinking eliminates the vast majority of the vulnerable points as specified by the $t$-test. Note that it cannot eliminate all of the blinking regions due to constraints on when we can blink (e.g., we must recharge). This improvement is quantified in Table 4.1.

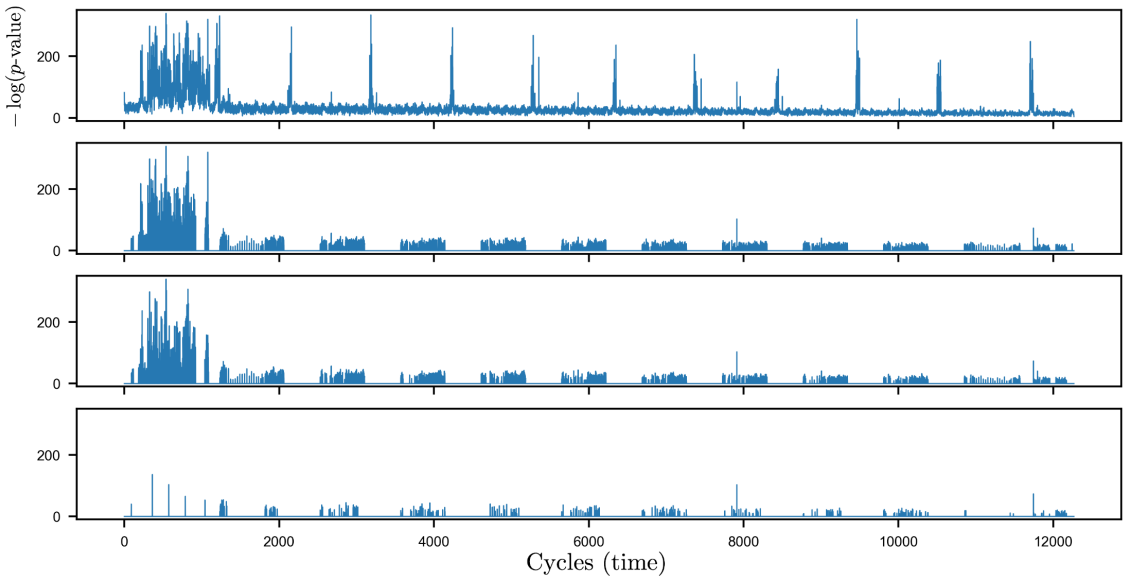


Figure 4.7: Pre and post-blinking comparison with multiple *blinkTime*s for masked AES-128 from the DPAv4.2 contest traces. The $-\log(p\text{-values})$ from the TVLA $t$-test before blinking (top, same as Figure 4.2) show marked reduction in the sparse vulnerable regions using full-length blinks (second from top) and full and $1/2$-length blinks (third from top) while the beginning section benefits greatly when using full, $1/2$-length, and $1/4$-length blinks (bottom). Note that not all of the leaky area at the front of the trace can be blocked—the recharge period after each blink means that lengthy leaky areas cannot be completely covered (unless one stalls for recharge). This recharge time is also why it is advantageous to have multiple blink lengths available for scheduling.

Table 4.1 shows our composite score, $t$-test vulnerability counts, and $1-\text{FRMI}_{\mathcal{B}}$ scores for the encryptions using Masked AES-128 from DPA contest v4.2, AES-128, and PRESENT ciphers. Both $\sum_{i=1}^{n} \mathbf{z}_i$ and $1-\text{FRMI}_{\mathcal{B}}$ are equal to one prior to blinking. The Table 4.1 values show the remainder afterward. We can see that with a good choice of $blinkTimes$ our scheduling approach eliminates nearly all of the leakage under the mutual information metric, and reduces attack vectors found by the $t$-test by an order of magnitude in the worst case. The PRESENT cipher implementation is consistently leaky throughout, but we still achieve a large improvement. These results should scale for any algorithm with intermittent, non-uniform leakage of secret information.

## 4.6 Conclusion

Power side channels are notoriously easy to exploit, yet difficult to mitigate. Electronic countermeasures attempt to eliminate the leakage through the power consumption and have shown to be successful in application-specific instances. Our work takes these one step further, by providing a software-controlled technique to disconnect and reconnect critical computation from the power supply. Our computational blinking technique enables the software to schedule brief moments of isolation in order to mitigate the power side channel. We show that our proposed system of computational blinking is general enough to apply to multiple different software systems and robust enough to achieve near-optimal information reduction. By giving the application explicit control of the trade-offs between security and performance, the architecture opens a useful new continuum of design points between fully-hidden and fully-exposed.

Chapter 4, in part, is a reprint of the material as it appears in the Proceedings of the 45th Annual International Symposium on Computer Architecture (ISCA), June 2018. Althoff, Alric; McMahan, Joseph; Vega, Luis; Davidson, Scott; Sherwood, Timothy; Taylor, Michael; Kastner, Ryan, ACM 2018. The dissertation author was the primary investigator and author of this paper.

# Bibliography


[1] F. Veljković, V. Rožić, and I. Verbauwhede, "Low-cost implementations of on-the-fly tests for random number generators," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2012, pp. 959–964.

[2] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[4] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[5] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[6] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[7] W. Cheng, C. Zheng, Y. Cao, Y. Zhou, H. Zhang, S. Guilley, and L. Sauvage, "How far can we reach? breaking rsm-masked aes-128 implementation using only one trace," *IACR Cryptology ePrint Archive*, 2017.

[8] L. Wei, Y. Liu, B. Luo, Y. Li, and Q. Xu, "I know what you see: Power side-channel attack on convolutional neural network accelerators," *arXiv preprint arXiv:1803.05847*, 2018.

[9] J. Park and A. Tyagi, "Using power clues to hack iot devices: The power side channel provides for instruction-level disassembly." *IEEE Consumer Electronics Magazine*, vol. 6, no. 3, pp. 92–102, 2017.

[10] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "Cachezoom: How sgx amplifies the power of cache attacks," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 69–90.

[11] Microsoft. (2018) Adv180002 — guidance to mitigate speculative execution side-channel vulnerabilities. [Online]. Available: https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/ADV180002

[12] M. Tiwari, H. Wassel, B. Mazloom, S. Mysore, F. Chong, and T. Sherwood, "Complete information flow tracking from the gates up," in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2009.

[13] D. P. Guelev, M. Ryan, and P. Y. Schobbens, "Model-checking access control policies," in *International Conference on Information Security*. Springer, 2004, pp. 219–230.

[14] P. K. Manadhata, K. M. Tan, R. A. Maxion, and J. M. Wing, "An approach to measuring a system's attack surface," Carnegie Mellon University, Tech. Rep., 2007.

[15] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Transactions on Software Engineering*, no. 3, pp. 371–386, 2010.

[16] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 83–89.

[17] D. Das, S. Maity, S. B. Nasir, S. Ghosh, A. Raychowdhury, and S. Sen, "High efficiency power side-channel attack immunity using noise injection in attenuated signature domain," in *Hardware Oriented Security and Trust (HOST), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 62–67.

[18] RAMBUS. (2015) Test vector leakage assessment (TVLA) derived test requirements (DTR) with AES. [Online]. Available: https://www.rambus.com/test-vector-leakage-assessment-tvla-derived-test-requirements-dtr-with-aes/

[19] A. Moradi, B. Richter, T. Schneider, and F.-X. Standaert, "Leakage detection with the $\chi^2$-test," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, no. 1, 2018.

[20] S. Bhasin, J.-L. Danger, S. Guilley, and Z. Najm, "Nicv: normalized inter-class variance for detection of side-channel leakage," in *Electromagnetic Compatibility, Tokyo (EMC'14/Tokyo), 2014 International Symposium on*. IEEE, 2014, pp. 310–313.

[21] M. Nassar, Y. Souissi, S. Guilley, and J.-L. Danger, "Rsm: a small and fast countermeasure for aes, secure against 1st and 2nd-order zero-offset scas," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2012, pp. 1173–1178.

[22] National Institute of Standards and Technology. (2001) Federal Information Processing Standards (FIPS) Publications: FIPS 140–2, Security Requirements for Cryptographic Modules. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf

[23] E. Barker and J. Kelsey, "Recommendation for random number generation using deterministic random bit generators," *NIST Special Publication*, vol. 800, p. 90A, 2012.

[24] M. S. Turan, "Recommendation for the entropy sources used for random bit generation," *NIST Special Publication*, vol. 800, p. 90B, 2018.

[25] B. Yang, V. Rožić, N. Mentens, and I. Verbauwhede, "On-the-fly tests for non-ideal true random number generators," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2015, pp. 2017–2020.

[26] B. Yang, V. Ro, N. Mentens, W. Dehaene, and I. Verbauwhede, "Total: Trng on-the-fly testing for attack detection using lightweight hardware," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 127–132.

[27] A. Vaskova, C. Lopez-Ongil, A. Jimenez-Horas, E. San Millan, and L. Entrena, "Robust cryptographic ciphers with on-line statistical properties validation," in *2010 IEEE 16th International On-Line Testing Symposium*, 2010.

[28] A. Tisserand, "Circuits for true random number generation with on-line quality monitoring," in *Claude Shannon Institut Workshop on Coding and Cryptography*, 2011.

[29] D. Hotoleanu, O. Cret, A. Suciu, T. Gyorfi, and L. Vacariu, "Real-time testing of true random number generators through dynamic reconfiguration," in *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*. IEEE, 2010, pp. 247–250.

[30] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," DTIC Document, Tech. Rep., 2001.

[31] P. Ducklin. (2013) Anatomy of a pseudorandom number generator - visualising cryptocat's buggy prng. [Online]. Available: https://goo.gl/UF1BGk

[32] T. W. Anderson and D. A. Darling, "A test of goodness of fit," *Journal of the American statistical association*, vol. 49, no. 268, pp. 765–769, 1954.

[33] O. H. Amman, T. von Kármán, and G. B. Woodruff, "The failure of the tacoma narrows bridge," *A Report to the Honorable John M. Carmody Administrator, Federal Works Agency Washington, D. C.*, 1941.

[34] K. Tiri, D. Hwang, A. Hodjat, B.-C. Lai, S. Yang, P. Schaumont, and I. Verbauwhede, "Prototype ic with wddl and differential routing–dpa resistance assessment," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2005, pp. 354–365.

[35] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen, "A side-channel analysis resistant description of the aes s-box," in *Fast Software Encryption*. Springer, 2005, pp. 199–228.

[36] ISO/IEC. (2016) CD 17825:2016, Non-invasive attack mitigation test metrics for cryptographic modules. [Online]. Available: https://www.iso.org/standard/60612.html

[37] R. Easter, "Side channel testing requirements in 19790," in *International Cryptographic Module Conference*, 2016. [Online]. Available: https://icmconference.org/wp-content/uploads/C11b-Easter.pptx.pdf

[38] L. Mather, E. Oswald, and C. Whitnall, "Multi-target dpa attacks: Pushing dpa beyond the limits of a desktop computer," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2014, pp. 243–261.

[39] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2002, pp. 13–28.

[40] M. A. El Aabid, S. GUILLEY, and P. HOOGVORST, "Template attacks with a power model," *IACR Cryptology ePrint Archive*, 2007.

[41] T. Alves and D. Feltggon, "Trustzone: Integrated hardware and software security," July 2004. [Online]. Available: http://www.arm.com/products/esd/trustzone_home.html

[42] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution." *Hardware and Architectural Support for Security and Privacy at International Symposium on Computer Architecture (HASP@ISCA)*, vol. 10, 2013.

[43] V. Costan, I. A. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation." in *USENIX Security Symposium*, 2016, pp. 857–874.

[44] M. Tiwari, J. K. Oberg, X. Li, J. Valamehr, T. Levin, B. Hardekopf, R. Kastner, F. T. Chong, and T. Sherwood, "Crafting a usable microkernel, processor, and i/o system with strict and provable information flow security," in *International Symposium on Computer Architecture (ISCA)*, 2011.

[45] J. Kelsey, B. Schneier, and N. Ferguson, "Yarrow-160: Notes on the design and analysis of the yarrow cryptographic pseudorandom number generator," in *International Workshop on Selected Areas in Cryptography*. Springer, 1999, pp. 13–33.

[46] A. Stefanov, N. Gisin, O. Guinnard, L. Guinnard, and H. Zbinden, "Optical quantum random number generator," *Journal of Modern Optics*, vol. 47, no. 4, pp. 595–598, 2000.

[47] J. Walker. (1996) Hotbits: Genuine random numbers, generated by radioactive decay. [Online]. Available: http://www. fourmilab.ch/hotbits

[48] S. Callegari, R. Rovatti, and G. Setti, "Embeddable adc-based true random number generator for cryptographic applications exploiting nonlinear signal processing and chaos," *IEEE Transactions on Signal Processing*, vol. 53, no. 2, pp. 793–805, 2005.

[49] P. Bayon, L. Bossuet, A. Aubert, V. Fischer, F. Poucheret, B. Robisson, and P. Maurine, "Contactless electromagnetic active attack on ring oscillator based true random number generator," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2012, pp. 151–166.

[50] A. T. Markettos and S. W. Moore, "The frequency injection attack on ring-oscillator-based true random number generators," in *Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer, 2009, pp. 317–331.

[51] K. Wold and C. H. Tan, "Analysis and enhancement of random number generator in fpga based on oscillator rings," *International Journal of Reconfigurable Computing*, vol. 2009, p. 4, 2009.

[52] B. Sunar, W. J. Martin, and D. R. Stinson, "A provably secure true random number generator with built-in tolerance to active attacks," *IEEE Transactions on computers*, vol. 56, no. 1, 2007.

[53] C. A. Hoare, "Algorithm 65: find," *Communications of the ACM*, vol. 4, no. 7, pp. 321–322, 1961.

[54] M. Greenwald and S. Khanna, "Space-efficient online computation of quantile summaries," in *ACM SIGMOD Record*, vol. 30, no. 2. ACM, 2001, pp. 58–66.

[55] D. Felber and R. Ostrovsky, "A randomized online quantile summary in $o(1/\varepsilon \log 1/\varepsilon)$ words," *CoRR*, vol. abs/1503.01156, 2015. [Online]. Available: http://arxiv.org/abs/1503.01156

[56] Z. S. Karnin, K. Lang, and E. Liberty, "Almost optimal streaming quantiles algorithms," *CoRR*, vol. abs/1603.05346, 2016. [Online]. Available: http://arxiv.org/abs/1603.05346

[57] L. Tierney, "A space-efficient recursive procedure for estimating a quantile of an unknown distribution," *SIAM Journal on Scientific and Statistical Computing*, vol. 4, no. 4, pp. 706–711, 1983.

[58] J. Kim, W. B. Powell, and R. A. Collado, "Quantile optimization for heavy-tailed distribution using asymmetric signum functions," *Princeton University*, 2011.

[59] M. S. Turan, E. Barker, J. Kelsey, K. McKay, M. Baish, and M. Boyle, "Recommendation for the entropy sources used for random bit generation," *NIST Special Publication 800-90B (2nd Draft)*, 2016.

[60] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American statistical association*, vol. 58, no. 301, pp. 13–30, 1963.

[61] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 1, pp. 3–30, 1998.

[62] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, 1948.

[63] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, pp. 1–23, 2011.

[64] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer-Verlag New York Inc, 2007, vol. 31.

[65] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2004, pp. 16–29.

[66] A. Heuser and M. Zohner, "Intelligent machine homicide," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2012, pp. 249–264.

[67] K. Tiri and I. Verbauwhede, "Simulation models for side-channel information leaks," in *Proceedings of the 42nd annual Design Automation Conference*. ACM, 2005, pp. 228–233.

[68] S. K. Rao, D. Krishnankutty, R. Robucci, N. Banerjee, and C. Patel, "Post-layout estimation of side-channel power supply signatures," in *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 92–95.

[69] D. Bellizia, S. Bongiovanni, P. Monsurro, G. Scotti, and A. Trifiletti, "Univariate power analysis attacks exploiting static dissipation of nanometer cmos vlsi circuits for cryptographic applications," *IEEE Transactions on Emerging Topics in Computing*, no. 1, pp. 1–1, 2016.

[70] A. Chakraborty, A. Mondal, and A. Srivastava, "Correlation power analysis attack against stt-mram based cyptosystems." *IACR Cryptology ePrint Archive.*, pp. 413–418, 2017.

[71] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology*. Springer, 1999, pp. 789–789.

[72] K. Pearson, "Note on regression and inheritance in the case of two parents," *Proceedings of the Royal Society of London*, vol. 58, pp. 240–242, 1895.

[73] F.-X. Standaert, T. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks." in *Eurocrypt*, vol. 5479. Springer, 2009, pp. 443–461.

[74] G. Becker, J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, T. Kouzminov, A. Leiserson, M. Marson, and P. Rohatgi, "Test vector leakage assessment (tvla) methodology in practice," in *International Cryptographic Module Conference*, vol. 1001, 2013, p. 13.

[75] F.-X. Standaert, "How (not) to use welch's t-test in side-channel security evaluations." *Cryptology ePrint Archive, Report 2017/138*, 2017.

[76] B. Gierlichs, L. Batina, B. Preneel, and I. Verbauwhede, "Revisiting higher-order dpa attacks," in *Cryptographers' Track at the RSA Conference*. Springer, 2010, pp. 221–234.

[77] S. Bhasin. (2015) Dpa contest v4.2. [Online]. Available: http://www.dpacontest.org/v4/index.php

[78] L. Zhang, A. A. Ding, F. Durvaux, F.-X. Standaert, and Y. Fei, "Towards sound and optimal leakage detection procedure." *IACR Cryptology ePrint Archive*, vol. 2017, p. 287, 2017.

[79] F. Durvaux, F.-X. Standaert, and N. Veyrat-Charvillon, "How to certify the leakage of a chip?" in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2014, pp. 459–476.

[80] O. Reparaz, B. Gierlichs, and I. Verbauwhede, "Fast leakage assessment," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 387–399.

[81] T. Schneider and A. Moradi, "Leakage assessment methodology," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 495–513.

[82] E. Prouff, M. Rivain, and R. Bevan, "Statistical analysis of second order differential power analysis," *IEEE Transactions on computers*, vol. 58, no. 6, pp. 799–811, 2009.

[83] H. F. Inman, "Karl pearson and ra fisher on statistical tests: A 1935 exchange from nature," *The American Statistician*, vol. 48, no. 1, pp. 2–11, 1994.

[84] P. Diaconis and D. Freedman, "Finite exchangeable sequences," *The Annals of Probability*, pp. 745–764, 1980.

[85] T. Schneider, A. Moradi, F.-X. Standaert, and T. Güneysu, "Bridging the gap: advanced tools for side-channel leakage estimation beyond gaussian templates and histograms," in *International Conference on Selected Areas in Cryptography*. Springer, 2016, pp. 58–78.

[86] Satoh Lab. (2014) Sakura hardware security project. [Online]. Available: http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html

[87] J. Blomer, J. Guajardo, and V. Krummel, "Provably secure masking of aes," in *Selected Areas in Cryptography*. Springer, 2005, pp. 69–83.

[88] M. Rivain and E. Prouff, "Provably secure higher-order masking of aes," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2010, pp. 413–427.

[89] L. Benini, E. Omerbegovic, A. Macii, M. Poncino, E. Macii, and F. Pro, "Energy-aware design techniques for differential power analysis protection," in *Design Automation Conference*. IEEE, 2003, pp. 36–41, design Automation Conference, 2003. Proceedings.

[90] S. Yang, W. Wolf, N. Vijaykrishnan, D. N. Serpanos, and Y. Xie, "Power attack resistant cryptosystem design: A dynamic voltage and frequency switching approach," in *Design, Automation and Test in Europe*. IEEE, 2005, pp. 64–69 Vol. 3, design, Automation and Test in Europe.

[91] T. Popp and S. Mangard, "Masked dual-rail pre-charge logic: Dpa-resistance without routing constraints," *Cryptographic Hardware and Embedded Systems*, pp. 172–186, 2005.

[92] K. Tiri, M. Akmal, and I. Verbauwhede, "A dynamic and differential cmos logic with signal independent power consumption to withstand differential power analysis on smart cards," in *Solid-State Circuits Conference*, 2002, pp. 403–406.

[93] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure dpa resistant asic or fpga implementation," in *Design, Automation and Test in Europe*, vol. 1, 2004, pp. 246–251 Vol.1.

[94] C. Clavier, J. S. Coron, and N. Dabbous, "Differential power analysis in the presence of hardware countermeasures," in *Cryptographic Hardware and Embedded Systems*. Springer, 2000, pp. 13–48.

[95] Pub, NIST FIPS, "197: Advanced Encryption Standard (AES)," *Federal information processing standards publication*, vol. 197, no. 441, p. 0311, 2001.

[96] S. Chari, C. Jutla, J. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Advances in Cryptology*. Springer, 1999, pp. 791–791.

[97] S. Guilley, S. Chaudhuri, L. Sauvage, T. Graba, J.-L. Danger, P. Hoogvorst, V.-N. Vong, M. Nassar, and F. Flament, "Shall we trust wddl?" in *Future of Trust in Computing*. Springer, 2009, pp. 208–215.

[98] J. S. Coron, P. Kocher, and D. Naccache, "Statistics and secret leakage," in *Financial Cryptography*. Springer, 2001, pp. 157–173.

[99] G. B. Ratanpal, R. D. Williams, and T. N. Blalock, "An on-chip signal suppression countermeasure to power analysis attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 3, pp. 179–189, 2004.

[100] P. Corsonello, S. Perri, and M. Margala, "An integrated countermeasure against differential power analysis for secure smart-cards," in *IEEE International Symposium on Circuits and Systems*. IEEE, 2006, p. 4 pp., iEEE International Symposium on Circuits and Systems.

[101] R. Muresan and S. Gregori, "Protection circuit against differential power analysis attacks for smart cards," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1540–1549, 2008.

[102] H. Vahedi, R. Muresan, and S. Gregori, "On-chip current flattening circuit with dynamic voltage scaling," in *IEEE International Symposium on Circuits and Systems*. IEEE, 2006.

[103] A. Shamir, "Protecting smart cards from passive power analysis with detached power supplies," in *Cryptographic Hardware and Embedded Systems*. Springer, 2000, pp. 121–132.

[104] C. Tokunaga and D. Blaauw, "Secure aes engine with a local switched-capacitor current equalizer," in *IEEE International Solid-State Circuits Conference*, 2009, pp. 64–65,65a.

[105] J. Demme, R. Martin, A. Waksman, and S. Sethumadhavan, "Side-channel vulnerability factor: A metric for measuring information leakage," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3, pp. 106–117, 2012.

[106] J. Chen and G. Venkataramani, "CC-hunter: Uncovering covert timing channels on shared processor hardware," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*. IEEE, 2014, pp. 216–228.

[107] R. Callan, A. Zajic, and M. Prvulovic, "A practical methodology for measuring the side-channel signal available to the attacker for instruction-level events," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*. IEEE, 2014, pp. 242–254.

[108] C. Clavier, J.-L. Danger, G. Duc, M. A. Elaabid, B. Gérard, S. Guilley, A. Heuser, M. Kasper, Y. Li, and V. Lomné, "Practical improvements of side-channel attacks on aes: feedback from the 2nd dpa contest," *Journal of Cryptographic Engineering*, vol. 4, no. 4, pp. 259–274, 2014.

[109] B. J. Gilbert Goodwill, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," in *NIST Non-invasive attack testing workshop*, 2011.

[110] S. Mangard, "Hardware countermeasures against dpa-a statistical analysis of their effectiveness," in *ct-rsa*, vol. 2964. Springer, 2004, pp. 222–235.

[111] A. Moradi and F.-X. Standaert, "Moments-correlating dpa," in *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*. ACM, 2016, pp. 5–15.

[112] T. Nakano, M. Kato, Y. Morito, S. Itoi, and S. Kitazawa, "Blink-related momentary activation of the default mode network while viewing videos," *Proceedings of the National Academy of Sciences*, vol. 110, no. 2, pp. 702–706, 2013.

[113] H. R. Schiffman, *Sensation and perception: An integrated approach* . John Wiley & Sons, 1990.

[114] A. Hall, "The origin and purposes of blinking," *The British journal of ophthalmology*, vol. 29, no. 9, p. 445, 1945.

[115] T. Nakano, Y. Yamamoto, K. Kitajo, T. Takahashi, and S. Kitazawa, "Synchronization of spontaneous eyeblinks while viewing video stories," *Proceedings of the Royal Society of London B: Biological Sciences*, p. rspb20090828, 2009.

[116] G. Brown, A. Pocock, M.-J. Zhao, and M. Luján, "Conditional likelihood maximisation: A unifying framework for information theoretic feature selection," *Journal of machine learning research*, vol. 13, no. Jan, pp. 27–66, 2012.

[117] P. E. Meyer, C. Schretter, and G. Bontempi, "Information-theoretic feature selection in microarray data using variable complementarity," *IEEE Journal of Selected Topics in Signal Processing*, vol. 2, no. 3, pp. 261–274, 2008.

[118] H. H. Yang and J. Moody, "Data visualization and feature selection: New algorithms for nongaussian data," in *Advances in Neural Information Processing Systems*, 2000, pp. 687–693.

[119] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[120] J. Kleinberg and E. Tardos, *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., 2005.

[121] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Low Power Methodology Manual: For System-on-Chip Design*. Springer Publishing Company, Incorporated, 2007.

[122] M. Pollet. (2017) SimAVR. [Online]. Available: https://github.com/buserror/simavr

[123] M.-L. Akkar, R. Bevan, P. Dischamp, and D. Moyart, "Power analysis, what is now possible..." in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2000, pp. 489–502.

[124] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Investigations of power analysis attacks on smartcards." in *Proceedings of USENIX Workshop on Smartcard Technology*, 1999, pp. 151–161.

[125] A. Gornik, A. Moradi, J. Oehm, and C. Paar, "A hardware-based countermeasure to reduce side-channel leakage: Design, implementation, and evaluation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 8, pp. 1308–1319, 2015.