Hiding Intermittent Information Leakage with Architectural Support for Blinking

Alric Althoff[1], Joseph McMahan[2], Luis Vega[3], Scott Davidson[3],
Timothy Sherwood[2], Michael B. Taylor[3], and Ryan Kastner[1]

[1]Computer Science and Engineering, University of California, San Diego
[2]Computer Science, University of California, Santa Barbara
[3]Computer Science, University of Washington
{aalthoff,kastner}@eng.ucsd.edu, {jmcmahan,sherwood}@cs.ucsb.edu, {vegaluis,stdavids,profmbt}@uw.edu

***Abstract*—As demonstrated by numerous practical attacks, the physical act of computation emits unintended and damaging information through infinitesimal variations in timing, power, and resource contention. While there are many techniques for preventing the leakage of information through power channels for specific cryptographic units, they are typically either built directly into the hardware logic or exploit intricate mathematical properties of the algorithm itself. However, such leaks are not uniform in time but, as we show, rather occur in specific bursts. Exploiting this observation we propose a set of software-controlled techniques allowing for the seamless disconnection and reconnection of general purpose programmable components in a system-on-chip. Such a system is capable of providing brief moments of electrical isolation during which the most critical computations can be performed free from both timing and power measurement. Of course, disconnection comes at a cost. To balance the resulting trade-off between overhead and security effectively, we describe a new analysis technique to uncover the "leakiest" intervals of time, we provide an algorithm to co-optimize the covering of these intervals and the performance/energy costs under a set of architecture imposed constraints, and explore the architectural and software ramifications of such intermittent disconnection. In the end we find that by hiding only between 15% and 30% of the trace, at a performance cost of between 15% and 50%, we are able to reduce the mutual information between the leakage model and key bits by 75% on average, and to nearly zero in specific cases.**

***Keywords*-hardware security; security metrics; side-channel attacks; electronic countermeasures**

I. Introduction

While there are many techniques by which information in a system can be kept secret, providing secrecy becomes even more challenging when the device is subject to physical access by an attacker. In these cases, the attacker likely has the ability to measure the dynamic power usage of the chip, which leaks subtle (but incredibly useful) information about any secret bits used during computation. The scope and sophistication of power analysis attacks on cryptographic systems has grown at a staggering pace. We are now at the point where even tiny amounts of information can be amplified through abstruse statistical machination over a set of repeated measurements. The attacker has many advantages, one of the largest being that they are free to collect an effectively unlimited number of traces in order to overcome noise.

To combat this problem, we propose a process called *computational blinking*. This architectural approach provides controlled dynamic power consumption isolation. Our system provides 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 (or "blinking"), the attacker learns nothing of value from the computation. As such, 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 [2], [31], [16], [49], this technique places the power leakage channels under software control. However, this idea brings with it many basic questions: how do we provide the core with required power while it is disconnected? How long can we perform this isolated computation? And finally, how do we optimally apply such disconnections to maximize the benefit to security and simultaneously minimize the impact on performance?

The reality of such an architecture is that one can only manage to operate in a disconnected state for very short bursts of time. Unlike traditional security mode architectures, where complete and long-running computations can take place if necessary, here we must be exceedingly judicious about what is covered by this mode. An important contribution of the paper is the technique by which we take a set of execution traces and analyze them automatically to uncover the points in the trace where the *vast majority* of information leakage is occurring. At one end of the spectrum is no protection (execution is fully subject to power analysis), and at the other is the extreme case where the entire secret operation is performed only in a series of disconnected states. We show that a security engineer is able to examine the entire spectrum of possibilities in between and make decisions that quantitatively trade-off between security and performance. For example, for a given AES implementation, our analysis shows that a designer can choose a near-perfect information blockage with a 2.7x slowdown, eliminate about half the leakage with a 12% slowdown, or choose some point in-between. Specifically, our work:

- demonstrates that information leakage over time in power traces is non-uniform and that the amount of useful information leaked at various points in an execution can actually be quantified.
- describes a novel technique that allows software to control power side-channel information leakage and to make informed decisions trading off between security and performance.

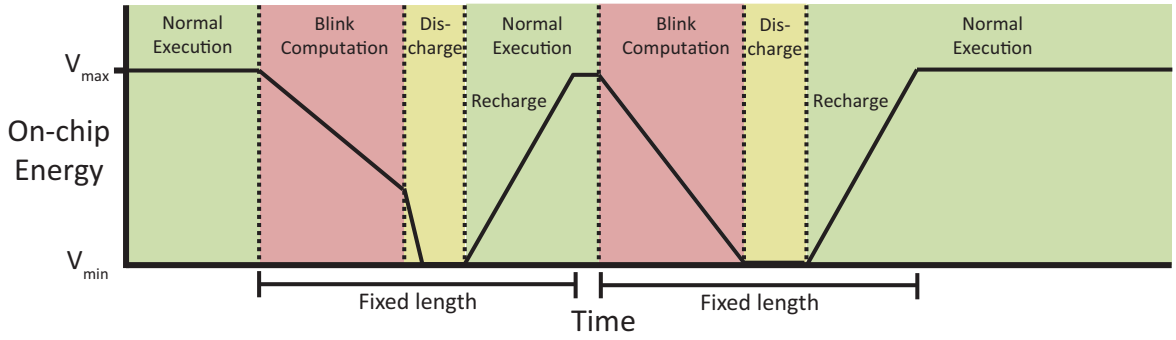

Fig. 1: A depiction of a computational blink. When a blink computation is invoked, three phases occur serially: 1) The blink computation begins and lasts as long as the maximum time needed for the worst case execution time of the code using the allotted energy. 2) A fixed "discharge" time occurs, regardless 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.

- examines the architectural impacts of such hardware/software co-operation including the ramifications for our new abstraction, the "blink", which exposes new hardware capabilities for disconnected operations to the instruction set.
- provides a careful accounting of a specific trade-off space between security, performance, and energy for secure operations using a combination of modeling and hardware measurements.

We begin in Section II with a discussion of power traces, the threats they introduce, and how the time-varying nature of their information leakage can be exploited. Section III introduces a rigorous method by which we can accurate quantify, for the first time, that time-varying information leakage and find the most useful points to target for removal. To edit those high points out, in Section IV 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 crypto algorithms in Section V and relate our contributions to prior work in Section VI before summarizing in Section VII.

II. Non-Uniform Information Leakage

Power analysis attacks are *surprisingly* powerful. It may seem counterintuitive 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 simply averaging across several runs will quickly remove that noise). This is a gross oversimplification of power analysis attacks, but gives accurate intuition as to why it is so hard to develop robust countermeasures.

Differential Power Analysis (DPA) is one of the most common side channel attacks due to its simplicity and effectiveness. It can be performed with low cost equipment in a small amount of time [27]. Attack success is conditional upon the fact that the energy consumed by the computation will be differ depending upon the values of the input data. And when that input data is confidential, such as a cryptographic key, the variance in power draw across an execution trace reveals information about the input, intermediate, and output data that is being computing. This enables the attacker to reason about portions of the secret data by making educated guesses about the execution of the algorithm, e.g., is the least significant bit of the result of a computation '0'or '1'? even when the instructions executed are exactly the same across all traces (as is common for crypto code). DPA verifies whether or not a guess is correct, which can quickly lead to determining the secret information. DPA is more powerful than prior methods in that it does not require intimate knowledge of the device under attack; typically the attacker only needs to know details about the computation being performed, e.g., the cryptographic algorithm. A successful attack is possible even with a significant amount of noise. However, it does require an additional amount of execution trace data, which depends upon the signal-to-noise ratio. As an example, a DPA attack on a particular AES software implementation requires approximately 200 traces to determine the entire key, while one hardware (ASIC) implementation is more difficult, requiring approximately 6500 traces [29]. The effectiveness of these attacks is related to the large reduction in search space, e.g., from 16×2^8 as opposed to 2^{128} for the brute force attack on an 128 bit secret key. Extensions of this idea into Correlated Power Analysis [6] and Template Attacks [10] extend these ideas with more accurate power models for the device and/or testing the physical device itself.

Typically, an attacker will target a time point that both leaks information and is easy to derive bits of the true key from.

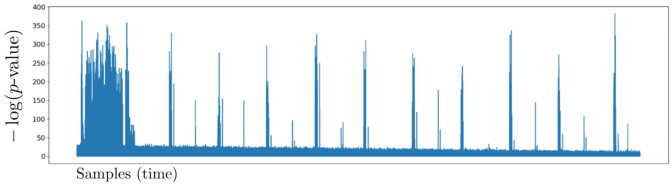

Fig. 2: Vulnerability of AES over time (as extracted from power traces). The x-axis is a unit of time and the y-axis is a measure of the leakage. Specifically the y-axis is the $-\log(p\text{-values})$ from a TVLA analysis of power traces. Larger values indicate that a location is more likely to leak information due to power variations.

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 found that the amount of useful information varied radically 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 VI, this is a first attempt to exploit the measured non-uniformity of leakage over time using a *programmable* electronic countermeasure.

A. 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 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 [26]), 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 [27], 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 [13].

B. Leakage Non-Uniformity

Figure 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. [20]. 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 [4], [20], [28], [34], [35], [46]. The questions remaining for Section III 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.

C. 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 [37] for a duration between 100-400 ms [44]. 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 [23] or at a scene change in a video [38]. 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 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 disconnected 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.

III. 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 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 3. This requires a sound information leakage model (Section III-A), a security metric to measure the "leakiest" moments in time (Section III-B), and a blink scheduling algorithm (Section III-C).

A. 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 [34] 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 III-B, 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 $f(\cdot)$ of processor behavior. In power SCA, $f(\cdot)$ would be measurements of voltage drop recorded with an oscilloscope. In general, $f(\cdot)$ 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 $f(\mathbf{t},\cdot)$ 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 $f(\mathbf{t},m,s)$. Per our convention, $f(\mathbf{t}_i,m,s)$ is the single value measured—or *leaked*—at time i, while $f(\mathbf{t},m,s)$ is a vector of leakage values (a power trace), at all times $\mathbf{t}$ for a fixed non-secret and secret. Likewise $f(\mathbf{t},\mathbf{m},\mathbf{s})$ is an order three tensor—a multidimensional array—of leakage values for multiple times, multiple non-secrets, and multiple secrets[1].

Intuitively, a system would be invulnerable to a leakage-based analysis if secrets have no effect on the measurements at

[1]A note on notation: for the remainder of this paper scalar variables will be lower case, vectors will be in bold lower-case, matrices in bold upper-case, and their indices as subscripts (e.g. respectively, y, $\mathbf{x}$, $\mathbf{A}$, and $\mathbf{x}_i$ and $\mathbf{A}_{ij}$). For sets $\mathscr{A}$, $\mathscr{A}^c$ is the complementary set, i.e. $\mathscr{A}\cup\mathscr{A}^c=\mathscr{U}$, the universal set, and $\mathscr{A}\cap\mathscr{A}^c=\emptyset$, the empty set

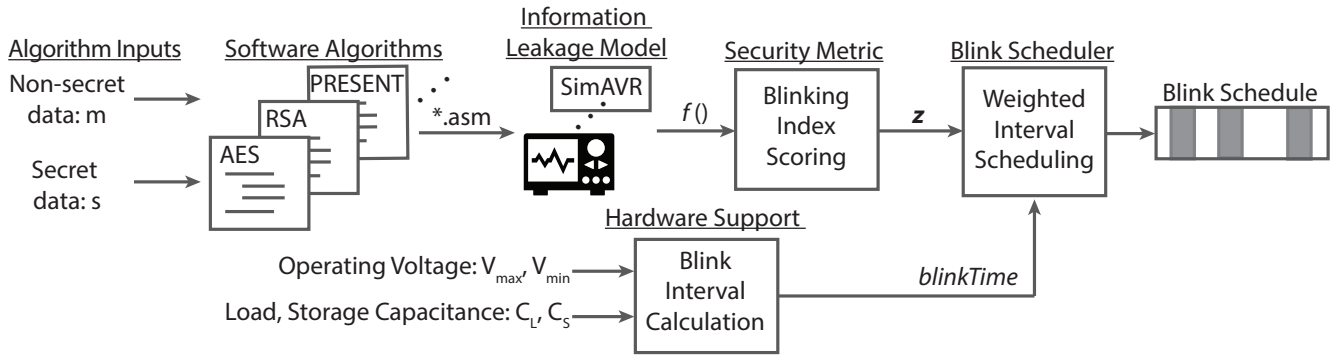

Fig. 3: An overview of the process to determine the best blink schedule for an algorithm. The algorithm inputs are labeled as secret and non-secret and the algorithm is analyzed to determine its power leakage $f(\cdot)$ either by collecting power traces or using a model. This is given to a security metric (Section III-B) that outputs a score $\mathbf{z}$ indicating the best places to blink, i.e., the times with the most leakage. A blink scheduler (Section III-C) takes this data and information from the hardware (Section IV) that determines the blink time(s) and outputs an optimal blink schedule.

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

$$f(\mathbf{t},\mathbf{m},\mathbf{s}) \stackrel{d}{=} f(\mathbf{t},\mathbf{m},\mathbf{Ps}),\ \forall \mathbf{P} \tag{1}$$

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* [18], and implies that the joint distribution leakage is unaffected by reordering secrets.

Eqn. 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 $f(\mathbf{t},\mathbf{m},\mathbf{s})$ obeys Eqn. 1, then we cannot do better than a random guess at the value of s for the leakage function f for a given device and measurement setup. Unfortunately, verifying Eqn. 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 III-B.

B. 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 IV and V we will consider practical constraints along with their security and performance ramifications.

In order to make Eqn. 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 Eqn. 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 classify 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 $\mathscr{B}$ allow us to differentiate among secrets, then we can blink them out, along with all redundant features.

For our choice of a baseline feature selection metric, we turn to [7], 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) [33], [52] is a balanced choice. The JMIFS is defined for a time index i as

$$\text{JMIFS}(i) = \sum_{j\in\mathscr{B}} I(f(\mathbf{t}_i,\hat{\mathbf{m}},\hat{\mathbf{s}}) \frown f(\mathbf{t}_j,\hat{\mathbf{m}},\hat{\mathbf{s}});\hat{\mathbf{s}}) \tag{2}$$

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, $\mathscr{B}$ 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 V.

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 $\mathscr{B}$, and remove it from the set $\mathscr{B}^c$ of remaining indices. We select the next index as the one that maximizes Eqn. 2, and so on, until there are no more indices to test, i.e. $\mathscr{B}^c = \emptyset$. We cache the values $\mathbf{J}_{ij} = I(f(\mathbf{t}_i,\hat{\mathbf{m}},\hat{\mathbf{s}}) \frown f(\mathbf{t}_j,\hat{\mathbf{m}},\hat{\mathbf{s}});\hat{\mathbf{s}})$ 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 $\mathbf{J}_{ij} = I(f(\mathbf{t}_i, \hat{\mathbf{m}}, \hat{\mathbf{s}}) \frown f(\mathbf{t}_j, \hat{\mathbf{m}}, \hat{\mathbf{s}}); \hat{\mathbf{s}})$ 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* [22]. 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, 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 [19], [30] exploit it to powerful effect. In this way our selection criteria is superior to other SCA security evaluation metrics [4], [20], [28], [34], [35], [46] that consider the power samples taken singly, or involve the use of combining functions[2] or multivariate histograms for only a few time indices. In any case, we cannot credibly assume a lack of complementarity of $f(\mathbf{t}_i, m, s)$ with $f(\mathbf{t}_{\mathscr{B}}, m, s)$, for arbitrary i and a set of time indices $\mathscr{B}$, 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.

Algorithm 1 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 $\mathbf{z}_j$.

[2]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 1: Blinking Index Scoring

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

C. 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 1, $\mathbf{z}_i$ 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 [25].

Algorithm 2 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 1. Additionally, it requires the constants *blinkTime* and *recharge*, which are dependent on the underlying blink-enabled hardware discussed in Section IV. Using $\mathbf{t}$ as the available times, and $\mathbf{z}$ as weights, Algorithm 2 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.

IV. 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

Algorithm 2: 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 $\mathscr{O}$ with optimal coverage
foreach $i \in [n]$ **do**
  $start \leftarrow i$
  $end \leftarrow i + blinkTime + recharge$
  $score \leftarrow \sum_{j=start}^{end-recharge} \mathbf{z}_j$
  $\mathbf{w}_i \leftarrow (start, end, score)$
foreach $i \in [n]$ **do**
  $\mathbf{prev}_i \leftarrow j$ s.t. $\mathbf{w}_j.end - \mathbf{w}_i.start$ is ≤ 0 and minimal
foreach $i \in [n]$ **do**
  $\mathbf{g}_i \leftarrow \max(\mathbf{w}_i.score + \mathbf{g}_{\mathbf{prevs}_i}, \mathbf{g}_{i-1})$
$i \leftarrow n$
$\mathscr{O} \leftarrow \emptyset$
while $i > 1$ **do**
  if $\mathbf{w}_i.score + \mathbf{g}_{\mathbf{prevs}_i} > \mathbf{g}_{i-1}$ **then**
    $\mathscr{O} \leftarrow \mathscr{O} \cup \{\mathbf{w}_i.start\}$
    $i \leftarrow \mathbf{prevs}_i$
  else
    $i \leftarrow i - 1$

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 [45] first described the cycled use of two switched capacitors, and more recently Tokunaga and Blaauw [50] 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

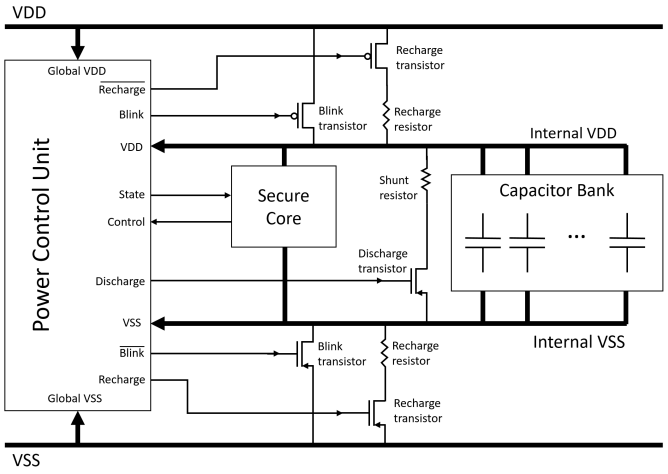

Fig. 4: Block diagram showing the function of the power control unit.

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:

$$blinkTime = \frac{2 \cdot \log\left(\frac{V_{min}}{V_{max}}\right)}{\log\left(1 - \frac{C_L}{C_S}\right)} \tag{3}$$

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 time to resume non-secure operation (see Figure 4). 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 [24], 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.

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. 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 [13], 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.

V. 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.

A. 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* [40]. 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 [6] 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 [1], [6], [32], [35] 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 (HW) 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,

$$\text{Power Leakage Model}(x,y) = \text{HW}(x \oplus y) + \text{HW}(y) \qquad (4)$$

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 [1], [6], [32], among others, empirically show that Eqn. (4) is robustly correlated with voltage drop across different devices.

B. 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 IV).

With the available blink sizes determined, the selection algorithm (Algorithm 2) can analyze a set of scores from (Algorithm 1) 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× 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 [13], 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.

	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. 1) post-blink	0.033	0.083	0.104
$1-\text{FRMI}_{\mathscr{B}}$ post-blink	0.012	0.011	0.140

TABLE I: 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 $1-\text{FRMI}_{\mathscr{B}}$ (Eqn. 6). Traces for the avrlib AES and PRESENT programs were collected via simulation using a Hamming distance power model (Eqn. 4), while DPA is from power traces from the DPA Contest v4.2 [13]. The last two allow comparison as a fraction of total leakage, where $1-\text{FRMI}_{\mathscr{B}}$ 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.

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× the stock execution time. In addition, provisioning the chip with a smaller storage capacitance may preclude optimal parameter selection for other algorithms.

C. Security Evaluation

To evaluate the security of blink-enabled hardware, we collect a set of 2^{14} traces from the leakage model described in Section V-A Eqn. 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 [13]. 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 III. 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 2, and are not data dependent. Additionally, as mentioned in Section V-B, while different blink lengths do not drain the capacitor equally, the shunt always ensures that

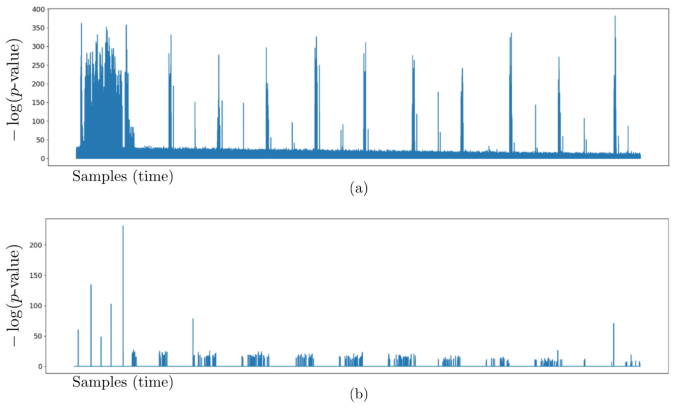

Fig. 5: a) The same as Figure 2, $-\log(p\text{-values})$ of the t-statistic of the DPAv4.2 traces prior to blinking. b) The $-\log(p\text{-values})$ from (a) after undergoing computational blinking according to our scoring and selection criteria. Note that not all of the leaky area at the front of the trace can be blocked—the cooldown period after each blink means that lengthy leaky areas cannot be completely covered (unless one stalls for recharge).

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 1, the TVLA t-test, and fractional reduction in mutual information (FRMI). We have discussed $\mathbf{z}$ and TVLA t-test previously. The mutual information is suggested in [46] as DPA security metric, and computed as

$$I(S;L) = H(S) - H(S|L) \tag{5}$$

where $H(S)$ and $H(S|L)$ are the entropy and conditional entropy[3] of random variables S and L, with $\mathbf{s}$ and $f(\mathbf{t}_i, \mathbf{m}, \mathbf{s})$ 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 [21], this metric corresponds directly to the success rate of a univariate template attack—the strongest form of attack in the information theoretic sense [10].

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

$$\text{FRMI}_{\mathscr{B}} = \frac{\sum_{i=1}^{n} I(f(\mathbf{t}_i, \hat{\mathbf{m}}, \hat{\mathbf{s}}); \hat{\mathbf{s}}) - \sum_{i \in \mathscr{B}} I(f(\mathbf{t}_i, \hat{\mathbf{m}}, \hat{\mathbf{s}}); \hat{\mathbf{s}})}{\sum_{i=1}^{n} I(f(\mathbf{t}_i, \hat{\mathbf{m}}, \hat{\mathbf{s}}); \hat{\mathbf{s}})} \tag{6}$$

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 5 shows the results before and after applying the blinking procedure on the same trace from Figure 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 I.

Table I shows our composite score, t-test vulnerability counts, and $1-\text{FRMI}_{\mathscr{B}}$ scores for the encryptions using Masked AES-128 from the DPA contest v4.2, AES-128, and PRESENT ciphers. Both $\sum_{i=1}^{n} \mathbf{z}_i$ and $1-\text{FRMI}_{\mathscr{B}}$ are equal to one prior to blinking. The Table I 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.

VI. Related Work

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 Eqn. 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.

Masking conceals 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. This enables security proofs against first order attacks [5], [39]. Masking modifies the cryptographic algorithm, and therefore cannot be directly applied to existing implementations; it requires a redesign that is costly or infeasible in many cases. If the mask can be determined (possibly through power analysis) then the security of the design reduces to that of an unprotected implementation. DPAv4.2 traces come from a masked AES implementation. This has not proved particularly effective, and many successful attacks exist.

Hiding defenses include randomly inserting dummy operations or cycles [3], shuffling the operations of the algorithm, changing the behavior of the clock and power signal [53], using multiple clock domains, and redundant logic styles to equalize transition probabilities [41], [47], [48]. 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) [12], [42]. The signal still exists and formal lower bounds on the number of traces can be determined [9], [29].

Electronic Countermeasures 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.

[3]Entropy in this sense is $H(S) = -\sum_{i=1}^{|S|} p(S_i) \log_b p(S_i)$, the expected number of symbols, from an alphabet with b total symbols, that need to be used to represent the random variable S.

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 [14]. Another approach is to insert active equalization circuitry to ensure that the power supply signal stays constant. Ratanpal et al. [43] describe a suppression circuit that reduces low frequency current fluctuations and a low-pass filter to remove high frequency variations. Corsonello et al. [15] propose a charge-pump circuit using on-chip capacitors. Muresan et al. [36], [51] 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 [45]. Tokunaga and Blaauw [50] use a switched capacitor current equalizer block to isolate an AES encryption core from the power supply line This core had a 25% area overhead, a 33% power overhead and 2× decrease in performance. This normalization technique is effective for the AES core where activity factors are highly stable at 50%. Performing this same technique on programmable devices will introduce significantly more overhead as the swing is much larger. 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 aim to quantify the vulnerability of the hardware against an attack. Side-channel vulnerability factor [17] 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 [11] detects timing channels by dynamic tracking conflict patterns, e.g., bus locks and functional unit contention. SAVAT [8] 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 III-B) focuses on power consumption. As we discuss in Section III-B, many other power side-channel security metrics exist for selecting vulnerable points, but like the TVLA test, they are univariate or require time points to be chosen a priori.

VII. 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.

VIII. Acknowledgements

The authors would like to thank the anonymous reviewers for their comments and suggestions which helped improve the quality of this paper. This material is based upon work supported by the National Science Foundation under Grants No. 1740352, 1730309, 1717779, 1563935, 1444481, 1341058, 1801052, CNS-1563767, and a gift from Cisco Systems.

References

[1] Mehdi-Laurent Akkar, Régis Bevan, Paul Dischamp, and Didier Moyart. Power analysis, what is now possible... In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 489–502. Springer, 2000.

[2] Tiago Alves and Don Felton. TrustZone: Integrated Hardware and Software Security, July 2004.

[3] Luca Benini, Alberto Macii, Enrico Macii, Elvira Omerbegovic, Fabrizio Pro, and Massimo Poncino. Energy-aware design techniques for differential power analysis protection. In *Design Automation Conference*, pages 36–41. IEEE, 2003. Design Automation Conference, 2003. Proceedings.

[4] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Nicv: normalized inter-class variance for detection of side-channel leakage. In *Electromagnetic Compatibility, Tokyo (EMC'14/Tokyo), 2014 International Symposium on*, pages 310–313. IEEE, 2014.

[5] Johannes Blömer, Jorge Guajardo, and Volker Krummel. Provably secure masking of aes. In *Selected Areas in Cryptography*, pages 69–83. Springer, 2005.

[6] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 16–29. Springer, 2004.

[7] Gavin Brown, Adam Pocock, Ming-Jie Zhao, and Mikel Luján. Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. *Journal of machine learning research*, 13(Jan):27–66, 2012.

[8] Robert Callan, Alenka Zajic, and Milos 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*, pages 242–254. IEEE, 2014.

[9] Suresh Chari, Charanjit Jutla, Josyula Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology*, pages 791–791. Springer, 1999.

[10] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 13–28. Springer, 2002.

[11] Jie Chen and Guru Venkataramani. CC-hunter: Uncovering covert timing channels on shared processor hardware. In *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, pages 216–228. IEEE, 2014.

[12] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In *Cryptographic Hardware and Embedded Systems*, pages 13–48. Springer, 2000.

[13] Christophe Clavier, Jean-Luc Danger, Guillaume Duc, M Abdelaziz Elaabid, Benoît Gérard, Sylvain Guilley, Annelie Heuser, Michael Kasper, Yang Li, Victor Lomné, et al. Practical improvements of side-channel attacks on aes: feedback from the 2nd dpa contest. *Journal of Cryptographic Engineering*, 4(4):259–274, 2014.

[14] Jean-Sébasticn Coron, Paul Kocher, and David Naccache. Statistics and secret leakage. In *Financial Cryptography*, pages 157–173. Springer, 2001.

[15] Stefania Perri Corsonello, Pasquale and Martin Margala. An integrated countermeasure against differential power analysis for secure smart-cards. In *IEEE International Symposium on Circuits and Systems*, page 4 pp. IEEE, 2006. IEEE International Symposium on Circuits and Systems.

[16] Victor Costan, Ilia A Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In *USENIX Security Symposium*, pages 857–874, 2016.

[17] John Demme, Robert Martin, Adam Waksman, and Simha Sethumadhavan. Side-channel vulnerability factor: A metric for measuring information leakage. *ACM SIGARCH Computer Architecture News*, 40(3):106–117, 2012.

[18] Persi Diaconis and David Freedman. Finite exchangeable sequences. *The Annals of Probability*, pages 745–764, 1980.

[19] Benedikt Gierlichs, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. Revisiting higher-order dpa attacks. In *Cryptographers Track at the RSA Conference*, pages 221–234. Springer, 2010.

[20] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side channel resistance validation. In *NIST noninvasive attack testing workshop*, 2011.

[21] Andreas Gornik, Amir Moradi, Jürgen Oehm, and Christof 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*, 34(8):1308–1319, 2015.

[22] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[23] Arthur Hall. The origin and purposes of blinking. *The British journal of ophthalmology*, 29(9):445, 1945.

[24] Michael Keating, David Flynn, Rob Aitken, Alan Gibbons, and Kaijian Shi. *Low Power Methodology Manual: For System-on-Chip Design*. Springer Publishing Company, Incorporated, 2007.

[25] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., 2005.

[26] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology*, pages 789–789. Springer, 1999.

[27] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, pages 1–23, 2011.

[28] Stefan Mangard. Hardware countermeasures against dpa-a statistical analysis of their effectiveness. In *ct-rsa*, volume 2964, pages 222–235. Springer, 2004.

[29] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer-Verlag New York Inc, 2007.

[30] Luke Mather, Elisabeth Oswald, and Carolyn 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*, pages 243–261. Springer, 2014.

[31] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. *HASP@ ISCA*, 10, 2013.

[32] Thomas S Messerges, Ezzy A Dabbish, and Robert H Sloan. Investigations of power analysis attacks on smartcards. pages 151–161, 1999.

[33] Patrick Emmanuel Meyer, Colas Schretter, and Gianluca Bontempi. Information-theoretic feature selection in microarray data using variable complementarity. *IEEE Journal of Selected Topics in Signal Processing*, 2(3):261–274, 2008.

[34] Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage detection with the χ^2-test. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):209–237, 2018.

[35] Amir Moradi and François-Xavier Standaert. Moments-correlating dpa. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, pages 5–15. ACM, 2016.

[36] Radu Muresan and Stefano Gregori. Protection circuit against differential power analysis attacks for smart cards. *IEEE Transactions on Computers*, 57(11):1540–1549, 2008.

[37] Tamami Nakano, Makoto Kato, Yusuke Morito, Seishi Itoi, and Shigeru Kitazawa. Blink-related momentary activation of the default mode network while viewing videos. *Proceedings of the National Academy of Sciences*, 110(2):702–706, 2013.

[38] Tamami Nakano, Yoshiharu Yamamoto, Keiichi Kitajo, Toshimitsu Takahashi, and Shigeru Kitazawa. Synchronization of spontaneous eyeblinks while viewing video stories. *Proceedings of the Royal Society of London B: Biological Sciences*, page rspb20090828, 2009.

[39] Stefan Mangard Norbert Pramstaller Oswald, Elisabeth and Vincent Rijmen. A side-channel analysis resistant description of the aes s-box. In *Fast Software Encryption*, pages 199–228. Springer, 2005.

[40] Michel Pollet. SimAVR. https://github.com/buserror/simavr, 2017.

[41] Thomas Popp and Stefan Mangard. Masked dual-rail pre-charge logic: Dpa-resistance without routing constraints. *Cryptographic Hardware and Embedded Systems*, pages 172–186, 2005.

[42] NIST FIPS Pub. 197: Advanced Encryption Standard (AES). *Federal information processing standards publication*, 197(441):0311, 2001.

[43] Ronald D. Williams Ratanpal, Girish B. and Travis N. Blalock. An on-chip signal suppression countermeasure to power analysis attacks. *IEEE Transactions on Dependable and Secure Computing*, 1(3):179–189, 2004.

[44] Harvey Richard Schiffman. *Sensation and perception: An integrated approach* . John Wiley & Sons, 1990.

[45] Adi Shamir. Protecting smart cards from passive power analysis with detached power supplies. In *Cryptographic Hardware and Embedded Systems*, pages 121–132. Springer, 2000.

[46] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Eurocrypt*, volume 5479, pages 443–461. Springer, 2009.

[47] Kris Tiri, Moonmoon Akmal, and Ingrid 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*, pages 403–406, 2002.

[48] Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure dpa resistant asic or fpga implementation. In *Design, Automation and Test in Europe*, volume 1, pages 246–251 Vol.1, 2004.

[49] Mohit Tiwari, Jason K. Oberg, Xun Li, Jonathan Valamehr, Tim Levin, Ben Hardekopf, Ryan Kastner, Fredric T. Chong, and Timothy Sherwood. Crafting a usable microkernel, processor, and i/o system with strict and provable information flow security. In *International Symposium of Computer Architecture (ISCA)*, 2011.

[50] Carlos Tokunaga and David Blaauw. Secure aes engine with a local switched-capacitor current equalizer. In *IEEE International Solid-State Circuits Conference*, pages 64–65,65a, 2009.

[51] Radu Muresan Vahedi, Haleh and Stefano Gregori. On-chip current flattening circuit with dynamic voltage scaling. In *IEEE International Symposium on Circuits and Systems*. IEEE, 2006.

[52] Howard Hua Yang and John Moody. Data visualization and feature selection: New algorithms for nongaussian data. In *Advances in Neural Information Processing Systems*, pages 687–693, 2000.

[53] Wayne Wolf Narayanan Vijaykrishnan Dimitrios N. Serpanos Yang, Shengqi and Yuan Xie. Power attack resistant cryptosystem design: A dynamic voltage and frequency switching approach. In *Design, Automation and Test in Europe*, pages 64–69 Vol. 3. IEEE, 2005. Design, Automation and Test in Europe.