Maya: Using Formal Control to Obfuscate Power Side Channels

Raghavendra Pradyumna Pothukuchi, Sweta Yamini Pothukuchi, Petros G. Voulgaris*, Alexander Schwing, and Josep Torrellas

University of Illinois at Urbana-Champaign *University of Nevada, Reno

***Abstract*—The security of computers is at risk because of information leaking through their power consumption. Attackers can use advanced signal measurement and analysis to recover sensitive data from this side channel.**

To address this problem, this paper presents *Maya*, a simple and effective defense against power side channels. The idea is to use *formal control* to *re-shape* the power dissipated by a computer in an application-transparent manner—preventing attackers from learning any information about the applications that are running. With formal control, a controller can reliably keep power close to a desired target function even when runtime conditions change unpredictably. By selecting the target function intelligently, the controller can make power to follow any desired shape, appearing to carry activity information which, in reality, is unrelated to the application. Maya can be implemented in privileged software, firmware, or simple hardware. In this paper, we implement Maya on three machines using privileged threads only, and show its effectiveness and ease of deployment. Maya has already thwarted a newly-developed remote power attack.

***Index Terms*—power side channels, physical side channels, security, obfuscation, control theory, machine learning.**

I. Introduction

The physical signals of a computer, such as its power consumption, temperature, and electromagnetic (EM) emissions, are strongly correlated with the computer's activity, and have been exploited as potent side and covert channels. Through these physical channels, attackers have been able to exfiltrate a variety of information about the applications running, including keystrokes and passwords [41], [76], location, browser and camera activity [41], [47], [78], and encryption keys [39], [42]. Many types of platforms have been successfully attacked using these physical channels, including smartphones, personal computers, cloud servers, multi-tenant datacenters, and home appliances [22], [32], [37], [41], [44], [47], [76], [78], [84].

The methods used to acquire power signals have significantly grown in number and stealth. Attackers use software techniques such as reading counters (e.g., PLATYPUS [42]), estimating power from unprivileged information, and analyzing code to estimate energy consumption [1], [15], [17], [55], [81], [83]. Attackers can also use hardware methods such as direct probing, antennas, indirect measurement by tapping electrical outlets and power supply networks, and using trojan chips, FPGAs and circuits [2], [26], [61], [63], [68], [75]. Since most of these techniques simply collect measurements, they cause little interference in the target computer and are hard to detect.

Recently, it has even been shown that the detailed power activity of a computer can be measured from a different room in a building, as long as the victim and the attacker computers are connected to the same power delivery network [63]. The attacker needs no physical access, and can measure power using widely-available equipment. This approach greatly amplifies the risk of leaking information through power signals.

Unfortunately, research on defenses against power side channels has not kept pace. One limitation is that most prior research on defenses has focused on encryption circuits (e.g., [7], [20], [39], [60], [65], [79], [80]). In practice, there are many attacks that are easy to mount, and which use system- or chip-level power measurements to steal sensitive information not related to encryption, like program activity, passwords and browsing data [19], [22], [32], [41], [44], [47], [76], [78], [84].

Another limitation of many proposed defense techniques is that they require new hardware and, hence, leave existing computers in the field vulnerable. Finally, mechanisms such as keeping constant power, inserting noise, or randomizing DVFS levels are unsuccessful because they do not completely mask application activity [41], [57], [63], [84].

An alternative approach is to modify *each* application individually, so that its activity is not visible through physical side channels [2]. However, this is a costly proposition.

Overall, there is an urgent need to develop effective defenses against power side channels that do not rely on special hardware, and which can be implemented as firmware or privileged software in an application-transparent manner. It is relevant to note that many common attacks that steal personal data like keystrokes or browser activity, analyze signals by sampling at intervals of several milliseconds or longer—suggesting that a firmware or software defense is a good choice.

To address this problem, this paper proposes *Maya*, a new defense technique that uses *formal control* [64] to intelligently *re-shape* the power dissipated by a computer in an application-transparent manner. When Maya is used, attackers cannot extract sensitive data of the running applications from the power signal. Maya's controller changes a computer's parameters to reliably keep the computer's power close to a given time-varying target, even under unpredictable runtime conditions. By setting this target intelligently, power can be shaped in any desired form, appearing to carry activity information which, in fact, is unrelated to the application. Such obfuscation removes leakage through power and, in addition, through temperature and EM signals, as they are related to power [13], [14], [44].

Maya can be implemented in privileged software, firmware, or simple hardware, and relies on commonly-available actuators to change power. These actuators are the DVFS level, which is supported by nearly all mainstream processors [5], [10], [59], [66], the injection of idle cycles to the execution [40], [69], and a custom "balloon" application whose power consumption can be increased on demand. In this paper, we implement Maya on three machines using privileged threads.

Shao et al. [63] recently describe a covert-channel attack across a building's power network. Four victim computers are connected to electrical power outlets. The attacker is also connected to an electrical power outlet in another part the building, at a distance of 90 feet, tapping on the same power network. The attacker samples voltage with an oscilloscope every $2\,\mu$s and, over a period of 33 ms, is able to decode one bit of information from the victims. Shao et al. then implement Maya, deploy it with defense actions taken every 40 ms, and show that Maya thwarts their covert channel.

In this paper, we introduce Maya and evaluate it against machine learning-based attacks on three different machines, in one case tapping an electrical power outlet, and show Maya's high effectiveness. The contributions of our work are:

1) Maya, a new defense technique against power side channels using *formal control* to *re-shape* the power signal. It is the first application of formal control to side-channel defense.
2) An implementation of Maya using only privileged software. To our knowledge, this is the first defense against power side channels that is readily-deployable and application-transparent. It operates at millisecond-level sampling, and thwarts power attacks without requiring physical access.
3) Evaluation of Maya against machine learning-based attacks.

The design of Maya's formal controller is available in [54].[1]

II. Background

A. Physical Side Channels

Physical side channels like power, temperature, and EM emissions can be used to uncover many details about an execution. Attackers have used these signals to infer the characters typed by a user [41], to identify the running application, the length of passwords on smartphones [76], the browser activity on personal computers [18], to disrupt operation in multi-tenant datacenters [33], and even to recover encryption keys from a cryptosystem [38].

Physical side channels appear because, as semiconductor devices switch, they consume dynamic power. The switching activity varies with instructions, which leave distinct fingerprints in the power trace [17], [41], [62], [70], [76]. Temperature and EM emissions are related to the computer's power, and leave similarly-analyzable patterns [13], [14], [44].

1) Signal Measurement: Attackers can capture physical signals in many ways, most of which are non-intrusive. For example, like PLATYPUS [42], attackers can use a malicious application that reads unprivileged hardware and OS counters for power or temperature [4], [44], [58]. In cloud systems, an application can use the thermal coupling between cores to infer the temperature or power profile of a co-located application using its own counters [44]. When power/thermal counters are unavailable, attackers can estimate power from OS metrics like utilization or from code analysis [55]. Malicious smart-batteries are another source of energy counters [41].

Power can also be measured by tapping AC electricity outlets [29], [30], power distribution units (PDUs) [32], and public USB charging booths [78]. If proximity to the victim is possible, low-cost infrared thermometers and antennas can be used to read temperature and EM emissions respectively [24], [25]. With direct access to the computer, attackers can use multimeters or oscilloscopes [39]. Such high-end equipment is usually necessary to extract encryption keys.

Trojan hardware such as chips, co-processors, FPGAs, and other IP modules that are co-located with the target chip can also surreptitiously measure the target's chip-level power or temperature [26], [61], [75], [85]. Cloud systems share FPGAs across processors and accelerators, and can be exploited for remote power measurement [26], [67], [85]. In multicore systems, the hierarchical power management policies can be abused to act as power covert channels between cores [37].

A computer's power activity can even be measured by an attacker hooked to an AC electrical power outlet connected to the same power delivery network, from a different location in a large building [63]. The attacker needs no physical access, and can use existing commercial equipment.

2) Signal Analysis: To extract sensitive information from signals, attackers can apply machine learning (neural networks), signal processing, and statistical analysis techniques [16], [41], [76]. Such techniques can identify information-carrying patterns in the signal, like its phase behavior and peak locations over time, and its frequency spectrum after a Fourier transform.

To extract encryption keys, attackers either use simple power analysis (SPA) on a single trace [24], or differential power analysis (DPA) over thousands of traces [39], [84].

The timescale over which the signals are analyzed is determined by the information that attackers seek and the available measurement channels. Most attacks steal information like the identity of the running applications, keystrokes, or browser data, and are performed with samples at intervals of milliseconds or more [41], [76], [78]. These are the timescales that this paper focuses on. For cryptographic keys, it is typically necessary to record and analyze signals with samples at intervals of a few microseconds or less [39].

B. State-of-the-Art Defenses

Prior defenses against power side-channel attacks have mostly focused on encryption circuits. They try to mask activity information by keeping physical signals at constant levels or by adding noise [7], [20], [39], [60], [65], [79], [80]. Unfortunately, all of these defenses need new hardware and, hence, cannot protect existing systems in the field.

Some of these defenses have additional limitations. For example, adding noise [7] or randomizing DVFS in the encryption circuits is easily countered by averaging multiple signal samples [57]. Furthermore, some of these circuit defenses first measure the encryption circuit's power and then change their

[1]Maya's source code is hosted at https://github.com/mayadefense/maya.

own activity to keep the overall power constant. Unfortunately, since the defense reacts only after observing the power changes, these defenses cannot fully hide application activity [20].

It is possible to implement software versions of these defenses to protect against information leaking through chip-level or system-level power signals. However, as we will show later, these software schemes also have limitations.

An alternative strategy is to modify applications so that they do not leak information through physical signals [2]. This is possible for a few critical applications (e.g., OpenSSL) but is impractical for the rest—like browsers, video or camera applications. To our knowledge, there are no defenses that can be readily used in existing machines in the field against power side channels in an application-transparent manner.

C. Formal Control Techniques

Using formal control [64], one can design a controller K that manages a system S (i.e., a computer) as shown in Figure 1. The system has outputs y (e.g., the power consumed) and configurable inputs u (e.g., the DVFS level). We want the outputs to be kept close to the output target functions r. The controller reads the deviations of the outputs from their targets ($\Delta y = r - y$), and sets the inputs appropriately.

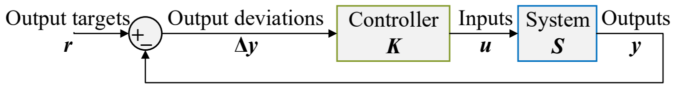


Fig. 1: Control loop.

The controller is a hardware or software state machine characterized by a state vector, $x(T)$, which evolves over time T. At a given time, it generates the system inputs $u(T)$ by reading the output deviations $\Delta y(T)$, and advances its state to $x(T+1)$:

$$x(T+1) = A \times x(T) + B \times \Delta y(T) \qquad x(0) = 0 \qquad (1)$$
$$u(T) = C \times x(T) + D \times \Delta y(T)$$

A, B, C, and D are matrices that encode the controller.

Designers specify multiple parameters in the control system [64]. They include the maximum bounds on the deviations of the outputs from their targets, the magnitude of the unmodeled effects that the controller must be tolerant of (i.e., the uncertainty guardband), and the relative priority of changing the different inputs (i.e., the input weights) [53]. With these parameters, controller design is automated [27], [51].

III. Threat Model

We consider power side-channel attacks that perform signal analysis at the timescale of milliseconds, and which use pattern recognition techniques such as machine learning, signal processing, and statistics to analyze the signal. Such attacks do not need physical access and can use widely-available commercial equipment. These attacks can steal information like the identity of the running applications, the keystrokes typed, and the browser data accessed. This threat model covers the majority of attacks [16], [17], [26], [33], [34], [37], [41], [44], [47], [63], [76], [78] described in Section II-A1, except for those attacks identifying encryption keys [2], [24], [25], [38], [85]. The latter attacks are harder to mount, and typically need more detailed knowledge of the cryptosystem being attacked.

We assume that attackers can know the algorithm used by Maya to reshape the computer's power. They can run Maya's algorithm and see its impact on the time-domain and frequency-domain behavior of applications. Using these observations, they can develop machine learning models to adapt to the defense and try to defeat it.

Finally, we assume that the firmware or privileged software that implements the control system for reshaping power is uncompromised. In a software implementation, the OS scheduler and DVFS interfaces need to be uncompromised.

IV. Obfuscating Power with Control

We propose that a computer system defend itself against power attacks by distorting its power consumption. Unfortunately, this is hard to perform successfully because simple distortions like adding noise can be removed by attackers using signal processing. This is especially the case if, as we assume in this paper, the attacker knows the defense algorithm used to distort the signal. Indeed, past approaches have been unable to provide a solution to this problem. In this paper, we propose the new approach of using *formal control* to re-shape power. In the following, we describe the architecture of Maya, the rationale behind using formal control, and the generation of effective distortions.

A. Maya Defense Architecture

Figure 2 shows the Maya architecture. Maya has a *Mask Generator*, a *Controller*, and mechanisms or inputs to change the power of a computer running an application. The mask generator creates the target power function to mislead attackers and communicates it to the controller. The controller reads this target and the actual power consumed by the computer, as given by power sensors. Then, it actuates all the inputs so that power is brought to the target.

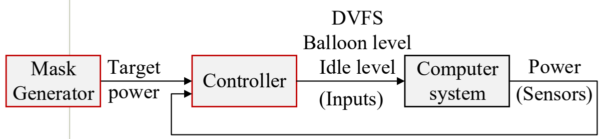


Fig. 2: High-level architecture of Maya.

The inputs that the controller actuates are the levels of DVFS, the *balloon* task, and the idle activity. A balloon task is one that performs power-consuming operations (e.g., floating-point operations) in a tight, tunable loop. The balloon level determines the number of power-consuming operations. The idle activity level determines the percentage of processor cycles in which the processor is forced into an idle state.

To understand the environment targeted by Maya, consider Table I. The table shows two types of power side-channel environments, which we call *InScope* and *OutOfScope*. Our envisioned Maya design targets the InScope environment.

TABLE I: Two types of power side-channel environments.

Characteristic	InScope	OutOfScope
Attacks	[16], [17], [26], [33], [34], [37], [41], [44], [47], [63], [76], [78]	[2], [24], [25], [38], [85]
Attacker's sensors	Counters, electrical line tapping with oscilloscopes	High-frequency probes, on-die trojan circuits
Signal analysis	milliseconds	$\leq$microseconds
Controller type	Matrix-based controller in firmware or privileged software	Table-based controller in hardware
Controller response time	5–10 μs	$\approx$ 10 ns
Example actuations	Change frequency and voltage, regulate balloon and idle levels	Insert compute instructions and bubbles in pipeline
Example uses	Hide what application runs or the keystrokes typed	Hide features of a crypto algorithm

In InScope, attackers measure power with methods like reading counters or tapping electrical power outlets, even with oscilloscopes [63]. Since the signal analysis is at the granularity of milliseconds, one can use typical matrix-based controllers as described in Section II-C. They are implemented in firmware or privileged software. The controller can respond in 5–10 μs, setting the DVFS level and regulating the balloon and idle activity levels. This implementation can hide information like the identity of the application running or the keystrokes typed. This environment is the focus of this paper, and is relevant because it is widely used.

Table I also shows the OutOfScope environment, which would require a different design for Maya. Here, attackers use better sensors, such as high-frequency probes, antennas, or on-die trojan circuits, and perform signal analysis at the micro- to nanosecond timescale. In this case, the controller has to be fast, and hence, cannot use the matrix-based approach. Instead, it has to use a table of pre-computed values from which it quickly reads the action to be taken. This controller must be implemented in hardware and have a response time of no more than $\approx$10 ns. Possible actuators in this environment are hardware modules that insert compute-intensive instructions or bubbles into the pipeline. With such fast actuation, this implementation could be used, e.g., to prevent information leaking from crypto-algorithms. We do not consider this environment in this paper.

B. Why Use Formal Control?

Formal control is necessary to reliably keep the computer's power close to the target power given by the mask generator. To understand the importance of formal control, consider the following scenario. We measure the power consumed by an application at fixed timesteps p_i, as shown in Figure 3a. To prevent information leakage, we want to distort the trace into a different, uncorrelated shape using the balloon application and idle activity.

One way to mislead the attacker is to keep power consumption at a constant level P (Figure 3b). To achieve this, we can

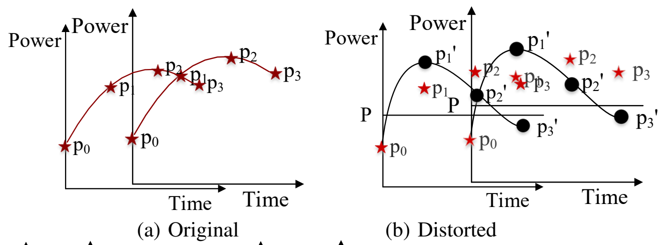


Fig. 3: Example of a power trace for an application.

measure the difference between P and the actual power p_i at each timestep, and schedule a combination of balloon threads and idle level based on $P - p_i$.

Unfortunately, this approach is too simplistic to be effective. It ignores that the application's power itself changes. To see why, consider Figure 3b. The original power trace is shown with stars, and the resulting one is shown with circles. For example, at the 0th timestep, a balloon thread is scheduled based on $P - p_0$. However, the power in the 1^{st} timestep ends up being p_1' rather than our target P, because the power of the application itself has also gone up.

If this control algorithm is repeatedly applied, it will always miss the target. We will obtain the trace in Figure 3b, where the measured power is not close to the target, and in fact has many features of the original trace.

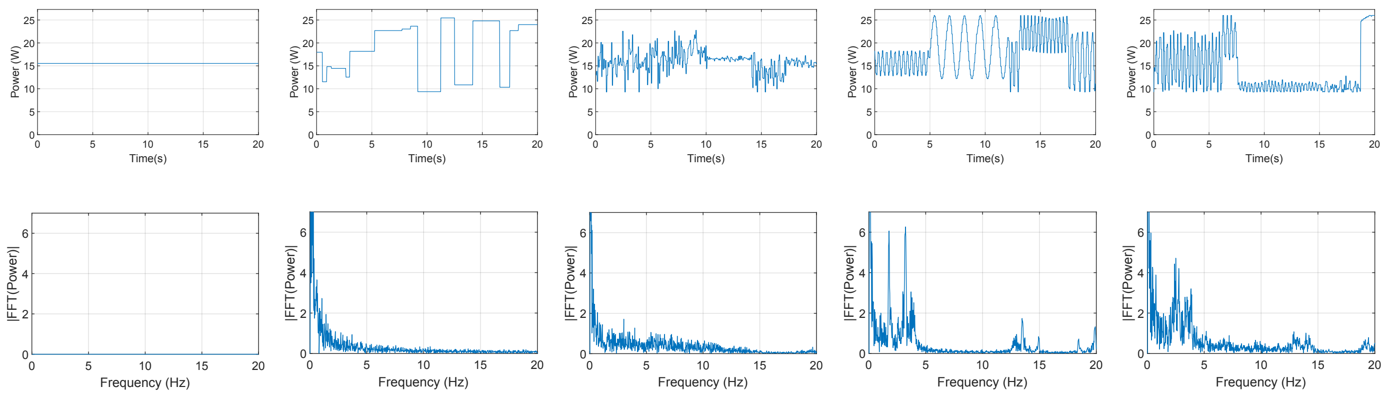
An approach that uses control theory is able to get much closer to the target power level. This is because the controller makes more informed power changes at every interval, based on history. To see why, we rewrite the equations of the controller's operation (Equation 1) slightly:

$$\begin{aligned} State(T+1) &= A \times State(T) + B \times Error(T) \\ Action(T) &= C \times State(T) + D \times Error(T) \end{aligned} \quad (2)$$

The second equation shows that the action taken at time T (in our case, regulating the balloon and idle activity) is a function of the tracking error observed at time T (in our case, $P - p_0$) *and* the controller's *state*. The state is an accumulation of the controller's experience in regulating the computer's power. The new state generated for the next timestep is determined by the current state and error. The "accumulated experience" in the state helps to get closer to the target.

Furthermore, the controller's actions and state evolution are influenced by the constant matrices A, B, C, and D, which were generated when the controller was designed. That process included running a set of *training* applications while scheduling the balloon and idle threads and measuring the resulting power changes. Hence, these matrices embed the intrinsic behavior of the applications under these conditions.

Note also that the controller has the ability to change multiple inputs at a time, which increases control accuracy. Overall, with formal control, the outputs can be kept close to the targets even when runtime behavior is unpredictable [64], which is often the case with computers. Hence, with a formal controller, the resulting power trace will be much closer to the target. If the target signal function is chosen appropriately, the attacker will be unable to obtain application information.



(a) Constant (b) Uniformly Random (c) Gaussian (d) Sinusoid (e) Gaussian Sinusoid

Fig. 4: Examples of masks. In each case, the time-domain curve is at the top, and the frequency-domain one is at the bottom.

C. *Generating Effective Targets (Masks)*

The target power function (or *mask*) should be constructed such that it can hide application activity effectively. Consider what happens if the target is simply set to a constant. As the application activity changes, any method to maintain the computer's power at a fixed level would have to first observe power deviating from the target, and then set the inputs accordingly. Hence, the output signal would have power activity leaking at all change-points in the application.

On the other hand, choosing a random target power at every timestep is not a good design either. The attacker could run the application many times, and then use signal processing techniques to remove the random noise. Then, the native change-points in the application would stand out. Therefore, the targets must be changed deliberately to hide such inadvertent leakage, and this is the role of Maya's mask generator.

An effective mask must hide information in both the time domain and the frequency domain (i.e., after obtaining its FFT). We postulate that such a mask must have three properties. First, the mask should have several phases, each with a different combination of mean and variance levels (e.g., Figure 4c top).

Second, the mask must have repetitive activity with varying periodicity. This will create several *peaks* in the power signal's FFT (Figure 4d bottom). Applications naturally create peaks in the FFT domain if they have loops. By introducing repetitive activity, any natural peaks are overwritten and/or hidden.

Finally, the phase transitions must have different rates from smooth to abrupt. As a consequence, the FFT of the mask is *spread* over a range of frequencies (e.g., Figure 4e bottom). If the mask has the above properties, the resulting power signal will have many artificially-induced change-points that will erase and/or hide the original ones.

We now examine generating a mask with the above properties using standard signals. Table II lists some well-known signals, showing whether the signal changes its mean and its variance in the time domain, and if it creates spread and peaks in the FFT (or frequency) domain. Figure 4 shows the signals.

A *Constant* mask (Figure 4a) has no change in either time or frequency domain. As discussed earlier, perfectly constant power cannot be realized in practice. When a constant target is used, information leaks at application change-points.

TABLE II: Some standard signals and what they change in the time and frequency domains.

	Time-domain		Frequency-domain	
Signal	Mean	Variance	Spread	Peaks
Constant	–	–	–	–
Uniformly Random	Yes	–	Yes	–
Gaussian	Yes	Yes	Yes	–
Sinusoid	Yes	Yes	–	Yes
Gaussian Sinusoid	Yes	Yes	Yes	Yes

In a *Uniformly Random* mask (Figure 4b), a value is chosen randomly from a range, and is used as a target for a random duration. After this period, another value and duration are selected, and the process repeats. This signal changes the mean but not the variance in the time domain. In the frequency domain, the signal is spread across a range but has no peaks. This mask too is a bad choice, since any repeating activity in the program would be hard to hide in the time domain.

The *Gaussian* mask (Figure 4c) is constructed by sampling values from a Gaussian distribution whose mean and variance are randomly changed over time. The resulting FFT is spread over multiple frequencies, but does not have peaks.

The *Sinusoid* mask (Figure 4d) generates a sinusoid and keeps changing the frequency, amplitude, and the offset randomly with time. This signal changes the mean and variance in the time domain. In the FFT, it has clear sharp peaks at each of its sinusoid frequencies. However, there is no spread. Consequently, the peaks can potentially be filtered out.

Finally, the *Gaussian Sinusoid* (Figure 4e) is the addition of the previous two signals. This signal has all the properties that we want (Table II): it changes the mean and variance in the time domain, and has spread and peaks in the frequency domain. Specifically, consider the FFT plots. The Gaussian noise (Figure 4c) has a noisy spectrum that is spread across a continuous range of values. In contrast, the Sinusoid signal (Figure 4d) has sharp and tall peaks. Therefore, the combination of the two signals (Figure 4e) results in a spectrum that has peaks that are both large and spread across a range. This is the mask that we propose.

Why Maya works: Maya works because it reshapes power instead of adding noise. With the latter, the original and distorted power signals differ only by noise, which can be filtered. Instead, Maya specifies a varying target shape first, and attains this power by actuating on the computer. So, the distortions are not simply separable noise; they are made to appear as carrying information. Attackers cannot isolate the disortions even if they know Maya's defense, as long as they cannot reproduce the random numbers used by Maya.

V. Implementation on Three Systems

We implement Maya to protect the three different computers listed in Table III. *Sys1* is a consumer-class machine with 6 physical cores, each with 2-way SMT, totaling 12 logical cores. *Sys2* is a server with 2 sockets, each having 10 cores of 2-way SMT, for a total of 40 logical cores. *Sys3* is another consumer-class machine with 4 physical cores, each with 2-way SMT. On all systems, the architecture of Maya is the same (Figure 2). We target the InScope attack environment of Table I. The controller and mask generator run as privileged software.

TABLE III: Implementation platforms.

Name	Configuration	RAPL sensors
Sys1	Sandy Bridge (12 cores) + CentOS 7.6	Cores+L1+L2
Sys2	Sandy Bridge (40 cores) + CentOS 7.6	Packages
Sys3	Haswell (8 cores) + CentOS 7.7	Cores+L1+L2

The Maya controller measures the power used by the cores plus L1 and L2 caches (Sys1 and Sys3), and by the two packages (Sys2) using RAPL [49] every 20 ms. It actuates three inputs: the DVFS level of all cores, the percentage of idle activity, and the balloon power level. DVFS levels are set through the `cpufreq` utility [12], and are 1.2-2.0 GHz (Sys1), 1.2-2.6 GHz (Sys2), and 0.8-3.5 GHz (Sys3) with 0.1 GHz increments.

The idle activity level is changed using Intel's `powerclamp` driver interface [69], and can be 0%-48% in steps of 4%. The powerclamp system launches as many kernel-level threads as the number of cores. These threads repeatedly displace other running threads and force the cores into idleness, until the desired level of idleness is achieved.

We develop a simple balloon application that runs floating-point operations in a loop. The percentage of the balloon activity is set using a `sysfs` file and can be 0%-100% in steps of 10%. The balloon application first spawns as many threads as the total number of cores. Then, in the main loop, the master thread configures each thread to run a loop of matrix multiply operations for a few ms followed by sleep cycles. If the desired power balloon level is high, the fraction of sleep is low and vice-versa. One iteration of the main loop (read level–run compute–sleep loop), takes $\approx$10 ms. The balloon threads are created with OpenMP, and run with root priority.

Maya introduces performance overheads in the system. The slowdown appears because the idle and balloon threads interrupt and displace the application tasks. The controller and mask generator, by themselves, are simple functions, and their overheads are low. We discuss the overheads in Section VII-E.

There are multiple ways to reduce Maya's overhead. One approach is to selectively activate Maya only in sections of the application where it is needed, similar to how power governors can be invoked in Linux [12]. Another approach is to run the application and power balloon threads on separate SMT contexts to avoid context switch overhead. Yet another approach is to implement Maya in firmware, to eliminate the software calls to read and modulate power. Finally, one can implement the power-burning circuits in hardware, to eliminate software overheads. In this paper, we do not evaluate these approaches and, hence, show Maya's worst-case performance impact.

A. Designing the Controller

We design the controller using robust control [64]. For this, we need to: (i) obtain a dynamic model of the computer system used, and (ii) set three parameters of the controller (Section II-C), namely input weights, uncertainty guardband, and output deviation bounds [51], [53].

To develop the model, we use the System Identification [43] modeling methodology. In this approach, we run a training set of applications on the computer system and, during execution, change the system inputs. We log the observed outputs and the inputs. From this data, we construct a dynamic polynomial model of the computer:

$$\begin{aligned} y(T) = a_1 \times y(T-1) + \ldots + a_m \times y(T-m) + \\ b_1 \times u(T) + \ldots + b_n \times u(T-n+1) \end{aligned} \tag{3}$$

In this equation, $y(T)$ and $u(T)$ are the outputs and inputs, respectively, at time T. This model describes the outputs at any time T as a function of the *m* past outputs, and the current and *n-1* past inputs. The constants a_i and b_i are obtained by least squares minimization from the experimental data [43].

We perform system identification by running two applications from PARSEC 3.0 (*swaptions* and *ferret*) [9] and two from SPLASH-2x (*barnes* and *raytrace*) [9] on Sys1. The models we obtain have a dimension of 4 (i.e., $m = n = 4$ in Equation 3). The system identification approach is a powerful way to capture the relationship between the inputs and outputs.

The input weights are set depending on the relative overhead of changing each input. In our system, all inputs have largely similar actuating overheads. Hence, we set all the input weights to 1. Next, we specify the uncertainty guardband by evaluating several choices. For each uncertainty guardband choice, Matlab tools [27] give the smallest output deviation bounds the controller can provide. Based on insights from prior work [51], [52], [53], we set the guardband to be 40%, which allows the output deviation bounds for power to be within 10%.

With the model and these specifications, standard tools [27] generate the A, B, C, and D matrices that encode the controller (Section II-C). The controller's dimension is 11, namely its state vector in Equation 1 has 11 elements. The controller runs every 20 ms; we set this duration based on the update rate of RAPL sensors and the latencies to change inputs.

B. Mask Generator

As stated in Section IV-C, we use a gaussian sinusoid mask (Figure 4e) to generate the targets. This signal is the sum of a sinusoid and gaussian noise, and its value at any time T is:

$$\left[Offset + Amp \times \sin\left(\frac{2\pi \times T}{Freq}\right)\right] + Noise(\mu, \sigma) \qquad (4)$$

where the Offset, Amp, Freq, μ and σ parameters keep changing. Each of these parameters is selected at random from a range of values, subject to two constraints. First, the maximum power target is always below the Thermal Design Power (TDP) of the system. Second, the sinusoid's frequency ($Freq$) cannot exceed 25 Hz because the power measurement rate itself is 50 Hz (from the 20 ms sampling interval). The power measurement has to be at least twice as fast as the sinusoid (Nyquist criteria).

Once a particular set of parameters is chosen, the mask generator uses them for N_{hold} samples, after which the parameters are updated again. N_{hold} itself varies randomly between 6 to 120 samples.

VI. Evaluation Methodology

A. Machine Learning-Based Power Attacks

We consider multiple common attacks based on machine learning as listed in Table IV. These attacks try to identify which application is running on the machine, which video is being encoded, and what is the user's browsing activity. They are widely reported in prior work [18], [30], [41], [47], [71], [76], [78]. The defense (i.e., Maya's controller) samples power at 20 ms intervals because RAPL provides reliable measurements only at this timescale. The attacker also samples power at 20 ms intervals except in Sys3 where, as we will see, the sampling interval is 50 ms because the measurements are taken from an AC power outlet cycling at 60 Hz.

TABLE IV: Machine learning-based power attacks.

Attacker's goal	Victim computer	Signal-capturing method
Detect running application	Sys1	Counters
Identify video being encoded	Sys2	Counters
Identify webpages visited	Sys3	AC outlet power

1. Detecting the running application: This is a well-known fundamental attack [30], [41], [71], [76]. Attackers capture many power traces of the applications they want to identify and build a machine learning classifier to recognize the application running from a power trace. We launch this attack on Sys1 using unprivileged RAPL counters to measure power. As in prior work, we assume that a malicious module installed by the attacker captures these counters [41], [76].

We run applications from PARSEC 3.0 (blackscholes, bodytrack, canneal, freqmine, raytrace, streamcluster, vips) and SPLASH-2x (radiosity, volrend, water_nsquared, and water_spatial) with native datasets and record 1,000 traces for each application. From each trace, we extract multiple segments of 15,000 RAPL measurements, and average the 5 consecutive measurements in each segment to remove the effects of noise. For accurate training, we quantize the power values into 10 levels and encode the traces in one-hot format. We use 60% of the data we collect for training, 20% for validation, and report the results for the remaining 20% test set.

For classification, we use a three-layer multilayer perceptron (MLP) neural network. The network uses ReLU units for its hidden layers and the output layer uses Logsoftmax.

2. Detecting video data: There is a video encoder that operates on multiple videos, and the attacker's goal is to identify the video being encoded. This is also a common attack [41], [47], [71], [76], [78]. We perform it on *Sys2* targeting the FFmpeg video encoder [23]. As with the previous attack, power signals are captured through RAPL.

We take four common test videos saved in raw format: tractor, riverbed, wind, and sunflower [74]. We transcode each video using FFmpeg's x264 compression for 200 runs and record the power traces. From each trace, we obtain multiple windows of 1,000 samples long, quantize the power values, and use one hot encoding to train our MLP classifier.

3. Detecting webpages: This is a popular attack [18], [41], [76], [78], and we set it up on *Sys3*. Unlike in the previous attacks, we capture the power traces by measuring the power from an *AC electrical outlet*. Figure 5 shows a picture of our test platform. We tap the electrical outlet used by the victim computer with wires connected to a multimeter. This multimeter (Yokogawa WT310) passes its measurements into another computer using a USB connection. This is a powerful and stealthy attack because information is obtained by simply rigging electrical outlets without installing any modules on the victim. Since the frequency of AC is 60Hz (corresponding to 16.6 ms cycles), the multimeter collects the root mean square (RMS) power samples every 50 ms (i.e., every 3 AC cycles).

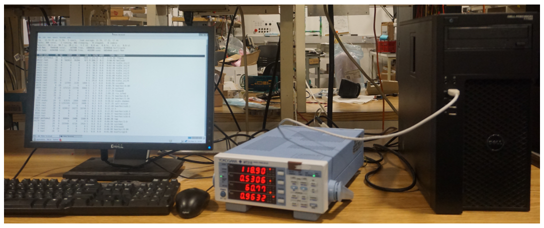
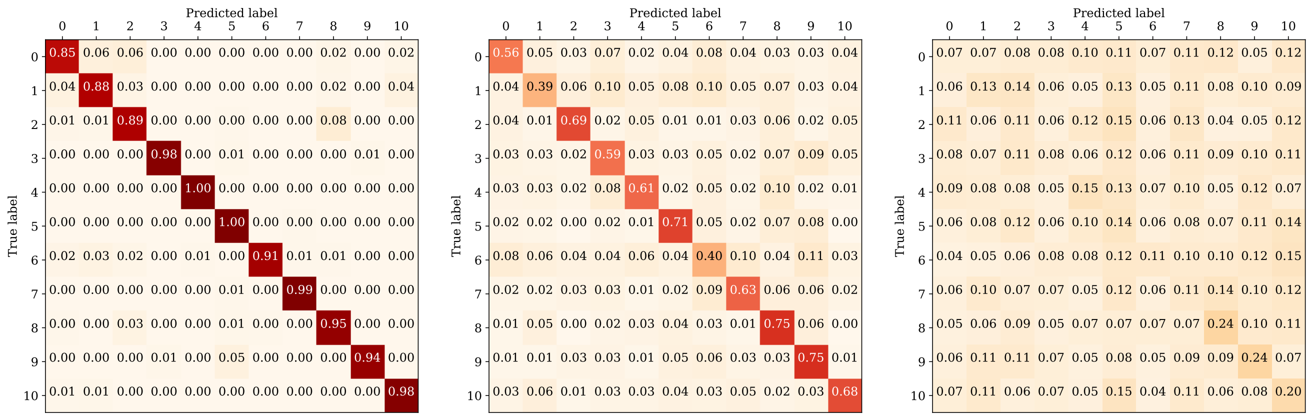


Fig. 5: Tapping an AC electrical outlet.

We record 100 power traces when visiting the popular websites google.com, ted.com, youtube.com, chase.com, ieeexplore.ieee.org/Xplore/home.jsp (IEEE Xplore), amazon.com and paypal.com using the Google Chrome browser. Each trace is nearly 15 seconds long. Unlike before, we use the signals' FFT to train our MLP because browser activity has varying rates of change in a short duration. The FFT captures it better.

B. Designs Compared

Since existing application-transparent defenses against power attacks need new hardware (e.g., [20], [79]) and cannot be tested on our machines, we build software defenses based on them. Table V lists the designs that we implement and compare.



(a) Random Inputs (Avg. accuracy 94%) (b) Maya Constant (Avg. accuracy 62%) (c) Maya GS (Avg. accuracy 14%)

Fig. 6: Confusion matrices for detecting the running application from power signals.

Baseline is a high performance insecure machine without any noise or mask to obfuscate the application to an attacker. In *Noisy Baseline*, each run of the application is executed with new DVFS, idle activity, and balloon levels that are picked randomly before the application starts, and kept fixed for the duration of the whole execution.

TABLE V: Designs compared.

Design	Description
Baseline	High-perf. insecure system without added noise
Noisy Baseline	Each run has a new DVFS, idle and balloon level
Random Inputs	DVFS, idle, and balloon levels change randomly at runtime
Maya Constant	Maya (Figure 2) with a constant mask
Maya GS	Maya with a Gaussian Sinusoid mask (Proposed)

Random Inputs changes the values of the DVFS, idle activity, and balloon levels randomly at runtime. Once a set of values is chosen, it is kept unchanged for a randomly selected duration, after which another set of values is selected. This makes the application's power profile significantly noisy.

Maya Constant uses Maya's formal controller but the power target is a constant. Finally, *Maya GS* is our proposal that uses the formal controller and a gaussian sinusoid mask generator.

We evaluate the security of the designs in Table V in an environment where attackers adapt to each defense. Specifically, attackers collect data to train their MLP classifier when the victims run with their defense on (i.e., *Random Inputs*, *Maya Constant*, or *Maya GS*). Then, they use their MLP to recognize new obfuscated traces from the *same* defense.

VII. Results

We first describe the effectiveness of the defenses. Then, we consider attacks at higher frequency, discuss the effectiveness of formal control, examine the defense overheads, and finally show that Maya can protect against PLATYPUS.

A. *Effectiveness of the Defenses*

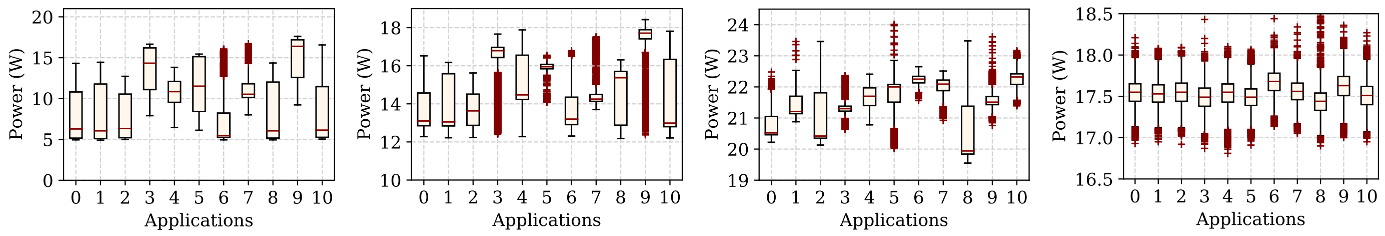
Detecting the running application: We show the effectiveness of the attack on different defenses from Table V using confusion matrices. A confusion matrix is a table where each row corresponds to the true labels of the applications (0 to 10 for the 11 applications) and each column has the fraction of the signals classified as the predicted labels by the attacker's MLP classifier. The matrices are shown in Figure 6. For example, the entry in the 0^{th} row and 1^{st} column gives the fraction of the signals that had a true label of 0 and were classified as application 1. The diagonal entries give the correct predictions, and averaging all the diagonal entries gives the overall average accuracy. Note that the random chance of correct classification is ≈9%, as there are 11 applications. An accuracy around this value indicates a classification failure.

Figure 6 shows the confusion matrices on the three main defenses that we test. Entries with higher fractions are darker. The average classification accuracy is 94% for *Random Inputs*, 62% for *Maya Constant*, and 14% for *Maya GS*. *Random Inputs* fails because randomly changing the DVFS, idle, and balloon levels does not hide the application's inherent activity. For example, changing DVFS has a different impact in compute and memory bound phases of the application. The MLP catches such differences.

Maya Constant manipulates the DVFS, idle, and balloon levels to maintain constant power. It has better obfuscation than *Random Inputs*, but is ultimately ineffective. As described in Section IV-C, ensuring constant power is not realistic, as information leaks at application change-points.

Finally, the attack on *Maya GS* has only 14% average accuracy. This is close to the random chance prediction accuracy of 9%. The difference occurs because the MLP's classification is biased towards a few labels (e.g., labels 9, 10). This occurs sometimes when the MLP cannot find patterns to learn in the training data. We verify this by training another MLP to predict the running application based on the target power mask generated by the mask generator. These power target masks have no correlation with the application that runs, and yet we see an accuracy of 12%.

Overall, *Maya GS* achieves excellent obfuscation. The gaussian sinusoid mask and the formal controller thoroughly



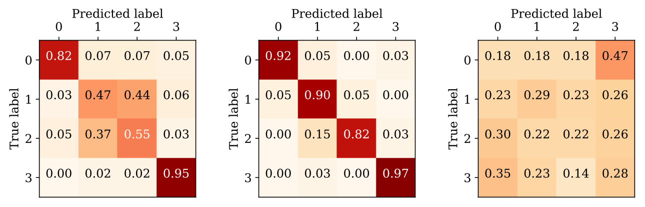
(a) Noisy Baseline (b) Random Inputs (c) Maya Constant (d) Maya GS

Fig. 7: Summary statistics of the average of 1,000 signals. The Y axis of each chart is drawn to a different scale.

hide any original patterns in the application with false activity. Since *Maya GS* produces a different trace in each run, the MLP cannot find any common pattern.

Detecting video data: Figure 8 shows the confusion matrices for the video detection attacks. Here, the accuracy of random chance classification is 25%, as we have four videos. The average accuracy of the MLP attack is 72%, 90% and 24% for *Random Inputs*, *Maya Constant* and *Maya GS*, respectively. As with the previous attack, *Random Inputs* and *Maya Constant* fail to obfuscate activity, and only *Maya GS* can hide activity.

The MLP has a lower accuracy against *Random Inputs* than against *Maya Constant*. It cannot clearly distinguish traces of video 1 (tractor) from video 2 (wind) with *Random Inputs*. Originally, these videos have similar traces except for a few peaks. The noise caused by *Random Inputs* results in misclassification. In contrast, *Maya Constant* makes the peaks more prominent because the signal is otherwise constant. Thus, the MLP has a higher success rate with *Maya Constant*.



(a) Random Inputs (Avg. accuracy 72%) (b) Maya Constant (Avg. accuracy 90%) (c) Maya GS (Avg. accuracy 24%)

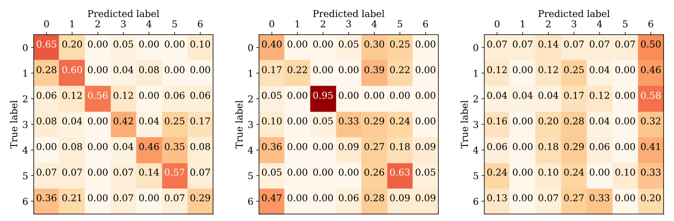
Fig. 8: Confusion matrices for the video detection attacks.

Detecting browser data: We run this attack using FFT values from AC outlet power traces. Here, the accuracy of random chance classification is 14%, as we have seven webpages. Figure 9 shows that the average accuracy of the MLP models is 51% for *Random Inputs*, 40% for *Maya Constant* and 10% for *Maya GS*. Websites like Google (0), Youtube (2), Chase banking (3) and Amazon (5) are recognized even with *Maya Constant*, thus endangering privacy. In contrast, *Maya GS* achieves high obfuscation.

Overall, these attacks show that *Maya GS* is successful in obfuscating power side channels. It resists attacks where the attacker trains with thousands of signals generated from Maya.

B. Signal Statistics and Analysis

For more insights, we analyze the signals produced by the defenses of Table V using signal summary statistics and change-point analysis.



(a) Random Inputs (Avg. accuracy 51%) (b) Maya Constant (Avg. accuracy 40%) (c) Maya GS (Avg. accurary 10%)

Fig. 9: Confusion matrices for webpage detection.

Signal summary statistics: We perform the following analysis for each defense. For each application, we collect all the power traces produced by the defense across runs and average them. Then, we examine the distribution of power values in this averaged signal, and compare it to the distribution of power values in *other* applications. An effective defense would produce similar distributions in all applications, so the applications are hard to distinguish.

Figure 7 shows the box plots of power values in the averaged traces for *Noisy Baseline*, *Random Inputs*, *Maya Constant*, and *Maya GS*. The averages are obtained from 1,000 raw traces of each application. Each chart labels the applications on the horizontal axis from 0 to 10. Each box includes the 25^{th} to 75^{th} percentile values for the application. The line inside the box is the median value. The whiskers of the box extend up to the maximum and minimum. The dark-red '+' markers represent values detected statistically as outliers in the distribution. For legibility, the Y axis on each chart is drawn to a *different* scale.

With *Noisy Baseline* (Figure 7a), the value distribution is distinct for each application and acts like a fingerprint. In *Random Inputs* (Figure 7b), the boxes shrink in size, but the relative difference remains the same. With *Maya Constant* (Figure 7c), the boxes shrink further (see the change in Y axis scale) and the median values of applications become closer to each other. However, the distribution is sufficiently different for the attacker to identify each application.

Finally, with *Maya GS* (Figure 7d) the distributions are *near-identical* (see the Y axis scale). The median values are nearly the same because *Maya GS* produces a different trace in each run that is uncorrelated with other runs. Moreover, each run uses the whole range of allowed values. Therefore, averaging traces cancels out the patterns. Hence, the median, mean, variance, and the distribution of the samples are close, indicating a high degree of obfuscation.

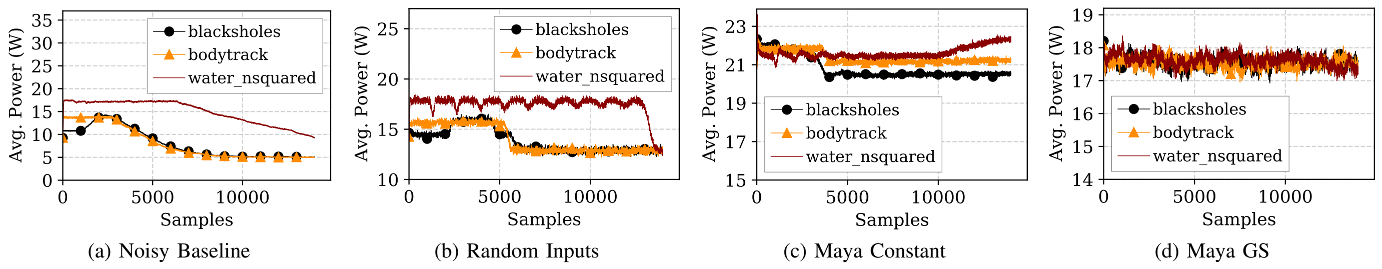


Fig. 10: Average of 1,000 traces for blackscholes, bodytrack and water_nsquared (labels 0, 1 and 9). The Y axis of each chart is drawn to a different scale.

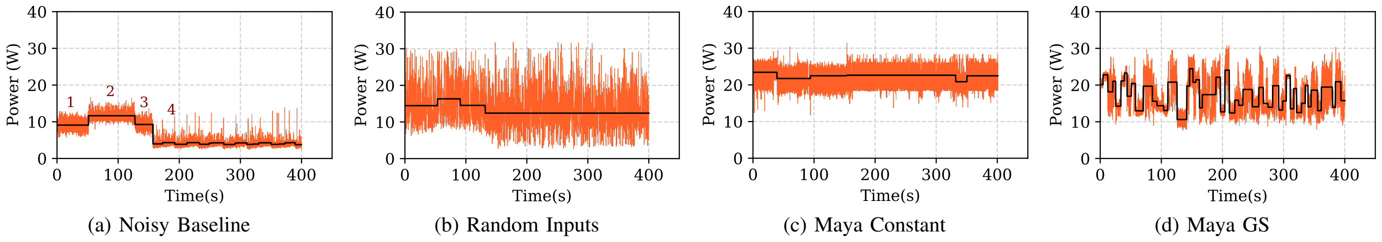


Fig. 11: Change-point detection in blackscholes using traces over time. Figure 11(a) shows all four phases being detected.

As an example of the differences across applications, Figure 10 shows the averaged signals of blackscholes, bodytrack, and water_nsquared for all the defenses. Again, the Y axis for each chart is drawn to a *different* scale. With *Noisy Baseline* and *Random Inputs*, the applications have different, recognizable patterns. With *Noisy Baseline* the differences are less visible (see the Y axis scale) but still obvious. It is only with *Maya GS* that the average traces are indistinguishable from each other. This results in the highest degree of obfuscation.

Change-point Detection: This is a signal-processing technique used to identify the times when the properties of a signal change. The properties can be the signal mean, variance, edges, or fourier coefficients. We use a standard change-point detection algorithm [45] to identify the phases found in the re-shaped signals. We present the highlights of this analysis using the *blackscholes* application.

With *Noisy Baseline* (Figure 11a), four phases of the application are clearly seen: (1) sequential, (2) parallel, (3) sequential and (4) idleness after the application ends. The difference between the phases is not too large, and there is some noise because of interference with idle and balloon activity. However, the algorithm detects the four major phases.

With *Random Inputs*, (Figure 11b), the profile is significantly noisy. However, since the noise is random, the inherent application activity is uniformly perturbed and hence, any phases in the application are still visible. The change-point detection algorithm identifies all the phases.

With *Maya Constant* (Figure 11c), the power profile is mostly around 25 W because the mask is held constant at that value. However, the algorithm can still recover all the phases. The constant target cannot prevent activity from leaking at phase transitions. There are sharp peaks at phase change points. The FFT of the signal (not shown) also preserves such changes.

With *Maya GS* (Figure 11d), change-point analysis detects many phases, *but these are all artificial*. The signal and its FFT (not shown) are totally different from the original signal. In fact, it is also *impossible to infer when the application completed*. The application completed around 121 s, but the signal shows no notable difference at that time.

We also used other signal processing techniques like dynamic time warping (DTW) [48], using standard distributions to fit data, and computing signal correlations. None of these methods was able to identify the true information carrying patterns with *Maya GS*.

C. Attacks at Higher Frequency

We repeat the running application detection attack on Sys1, but *reduce the attacker's power sampling interval from 20 ms to 10, 5, and 2 ms*. In all cases, the defense Maya samples power at 20 ms. Figure 12 shows the average application detection accuracy across the sample intervals. We see that the average detection accuracy does not change much, and remains low like in Figure 6c, even with the faster sampling by the attacker.

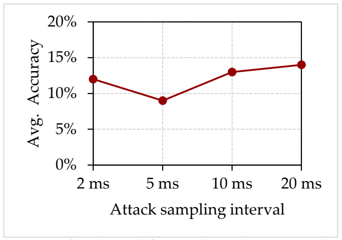
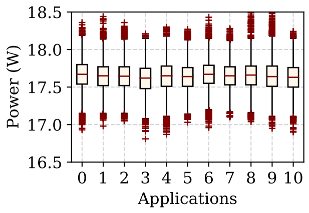


Fig. 12: Accuracy of classifiers in the application detection attack on *Maya GS* for different attacker sampling intervals.

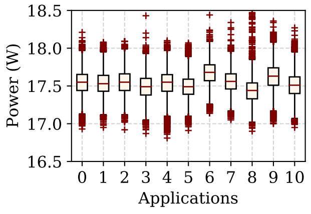
Faster sampling does not improve detection accuracy in this attack because the distinguishing patterns of the applications (i.e., phases) occur at longer timescales than the sampling intervals. Furthermore, even though Maya's controller only changes inputs every 20 ms, the idle and balloon threads are always running, and add noise to the application. Finally, faster sampling inherently has more noise, affecting detection.

D. Effectiveness of Formal Control

Figure 13 shows the distribution of power values in the averaged signals as given by: (i) the gaussian sinusoid mask generator (Figure 13a) and (ii) the actual power measured from the computer (Figure 13b). The latter is the same as Figure 7d. It can be seen that the formal controller is effective at making the measured power appear close to the target mask. Indeed, this accurate tracking is what makes Maya effectively re-shape the system's power and hide application activity.



(a) Given by the *Maya GS* mask.



(b) Measured from the computer.

Fig. 13: Distribution of power values in the averaged signals.

E. Maya Overheads and Impact on Power and Performance

We examine the implementation overheads of Maya, and its impact on the power and performance of applications.

Overheads of Maya: Maya runs as a software process that wakes up at regular intervals to read the power sensors, generate the next mask value, run the controller, and initiate the actuations.

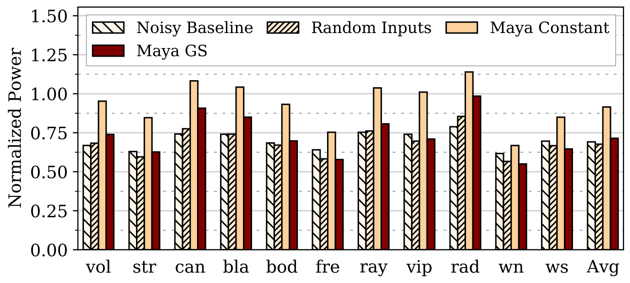
Generating the next mask value requires obtaining one (pseudo) random number to sample from the gaussian distribution. However, when the properties of the gaussian and sinusoid functions are changed (Equation 4), more random numbers need to be generated. In our implementation, we use the C++ STL library. In the worst case, getting all the required random numbers takes about a μs. Optimized implementations can generate the random numbers faster [31], [35].

Running the controller involves computing Equation 1 using the difference between the target and the measured power values to obtain the DVFS, idle, and balloon levels. The controller has an 11-element state vector $x(T)$ (Equation 1). It can be shown that running the controller needs ≈200 fixed-point operations, which complete within 1 μs. The controller needs less than 1 Kbyte of storage.

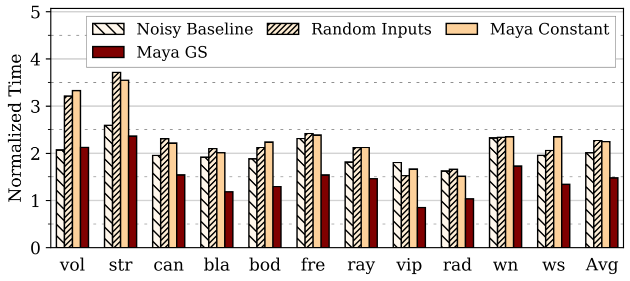
Maya needs few resources to operate, making it attractive for firmware, software, or even hardware implementations. The primary bottlenecks in our implementation are the sensing and actuation latencies, which can reach a ms or more.

Impact on Application Power and Performance: We run the PARSEC and SPLASH-2x applications on *Sys1* with the different defense designs and *Baseline*. *Baseline* runs applications at the highest available frequency without inserting idle or balloon threads. We measure power and execution time.

Figure 14 shows the power and execution time of all the defense designs, normalized to that of the high-performance *Baseline*. In Figure 14a, we see that the average power consumed by the applications with *Noisy Baseline*, *Random Inputs*, *Maya Constant*, and *Maya GS* is 30%, 31%, 11% and 29% lower than *Baseline*. The power with the defenses is typically lower than in *Baseline* because all defenses use idle threads and sometimes low DVFS values.



(a) Power



(b) Execution Time

Fig. 14: Power and execution time of the defense designs on *Sys1* relative to the high-performance insecure *Baseline*.

Figure 14b shows the normalized execution times. On average, the execution times of *Noisy Baseline*, *Random Inputs*, *Maya Constant*, and *Maya GS* are 100%, 127%, 124% and 47% higher, respectively, than Baseline. *Noisy Baseline* and *Random Inputs* are slow because of the continuous interference of the idle and balloon threads with the application. *Maya Constant* uses a single power target throughout execution, which is often lower than the power at which Baseline runs. Therefore, the execution time overhead of *Maya Constant* is high. *Maya GS* has a relatively lower execution time overhead than the other defenses because its execution uses a wide range of power levels thanks to its many choices. Hence, it allows applications to run steadily at high power occasionally. Note that, of all the defenses, only *Maya GS* provides security.

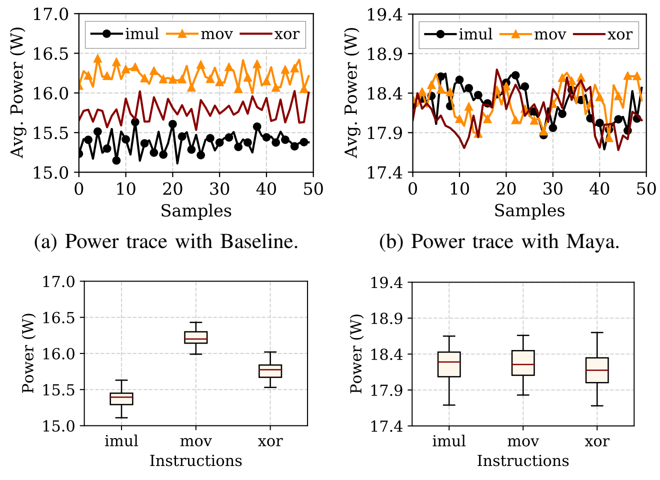
We believe that the lower performance of *Maya GS* relative to *Baseline* is acceptable given the high level of security that it provides without needing any hardware support at all. To reduce the overhead of Maya, secure applications can selectively activate Maya only during sensitive sections of the application [28]. Furthermore, Section V has discussed more advanced ways to reduce Maya's execution overhead.

It can be shown that the power and performance overheads of the *Sys2* and *Sys3* designs are like those of *Sys1*. This shows that Maya is robust across different machines.

Impact on Total Application Energy: *Maya GS* consumes approximately the same total energy as *Baseline*: it consumes 29% lower power (Figure 14a) but the execution takes 47% longer (Figure 14b). The resulting product of power and time is similar to that of the original execution.

F. Defending against PLATYPUS-type Attacks

Initial results show that Maya can defend against an attack like PLATYPUS [42]. In our experiments, we run tight loops with `mov`, with `xor`, or with `imul` (multiply) instructions on *Baseline*, while measuring the power. We repeat the experiments 200 times and compute the average power traces. As shown in Figures 15a and 15c, the profiles of these three power traces differ. Then, we enable *Maya GS* and regenerate the power profiles. As shown in Figures 15b and 15d, the profiles of the three power traces are now practically indistinguishable. Hence, a defender can use *Maya GS* to hide the instruction that is being executed, thwarting the attack. We will extend this analysis in future work.



(a) Power trace with Baseline.
(b) Power trace with Maya.
(c) Power distribution with Baseline.
(d) Power distribution with Maya.

Fig. 15: *Maya GS* is able to defend against PLATYPUS.

VIII. Related Work

Attacks: Many attacks identify sensitive information from physical signals using machine learning (ML)-based pattern recognition [16]. Yan et al. recover the running application's identity and the number of keystrokes typed [76], by measuring unprivileged power counters. Lifshits et al. identify browser, camera and location activity, and the characters typed [41], using measurements from a malicious battery. Chen et al. recover Android app usage information from power signals [17]. Yang et al. show that compromised public USB charging booths can recover the user's browser activity [78]. Das et al. train ML networks that identify encryption keys from multiple target devices despite inter-device variations in their power signals [21]. They also show that their ML-based key recovery is faster and requires fewer traces than conventional correlation power analysis (CPA).

Michalevsky et al. show that malicious smartphone applications can track the user's location without GPS, by only using unprivileged OS-level counters [47]. Conti et al. [19] show that a laptop's power signals can be used to identify the laptop's original user among other occasional users. Wei et al. use power measurements to recover the input image processed by an FPGA convolutional neural network accelerator with only some information about the network's structure [72].

Clark et al. [18] and Hlavacs et al. [30] identify the webpages and virtual machine applications, respectively, using the server's electrical outlet power. Guri et al. exfiltrate data from air-gapped power lines [29]. Shao et al. develop a covert channel that uses the power delivery network to which a computer is connected [63]. Khatamifard et al. build a power covert-channel based on the hierarchical power management policies in multicores [37].

Some attacks use trojan chips, circuits and FPGAs to measure physical signals of co-located chips [26], [61], [68], [75], [85]. Cloud systems are offering FPGA platforms, and are vulnerable to power analysis attacks [26], [67], [85]. Kocher et al. [39] give a detailed overview of recovering encryption keys using SPA and DPA when physical access to a cryptosystem is possible. Attackers also use software analysis or modeling to estimate power when direct measurement is difficult [1], [15], [76], [81], [83]. Wei et al. show that several malware applications have power signatures and use ML to detect them [73].

There are attacks that capture EM emissions using antennas [2], [14], [24], [25], [36], [71]. Enev et al. find that television content can be distinguished based on the EM signatures of the television's powersupply [22].

Islam et al. show the vulnerability of multi-tenant datacenters to voltage, thermal and acoustic (from cooling devices) side-channels [32]–[34] that arise due to activity variations. Masti et al. develop a covert channel in multicores based on temperature coupling between the cores [44].

Defenses: Trusted execution environments like Intel SGX [46] or ARM Trustzone [6] do not contain physical signals [11], [13], [42]. Therefore, several countermeasures against power side channels have been proposed [3], [7], [20], [39], [56], [60], [65], [77], [79], [80], [82], [84]. Their goal is primarily to protect encryption circuits, and require new hardware.

Known defenses usually operate by either suppressing power signal changes that arise due to changing activity [56], [60], [65], [82], or adding noise to drown activity [39], [84], or both [20]. A common approach to adding noise is to randomize DVFS levels using special hardware [79], [80]. Avirneni and Somani also propose new circuits for randomizing DVFS, but change voltage and frequency independently [7].

Baddam and Zwolinski show that randomizing DVFS alone is not a viable defense because attackers can identify clock frequency changes through high-resolution power traces [8]. Yang et al. suggest randomly scheduling the encryption task among the cores in a multicore [77], apart from randomly setting the clock frequency and phase [77]. Real et al. show that adding noise or empty activity can be filtered out, and is ineffective [57].

A different approach is to temporarily cut-off a circuit from the outside and run it with a small amount of energy stored inside itself [3]. Alternatively, each application can be modified so that its physical outputs do not carry sensitive information [2]. Finally, there are defenses that use adversarial ML [28], [50] to exploit weaknesses in the ML-based attack classifiers. The idea is to add perturbations to the power signals so that the classifiers produce the incorrect result.

IX. Conclusions

This paper presented a simple and effective solution against power side channels. The scheme, called *Maya*, uses for the first time, formal control to distort, in an application-transparent way, the power consumed by a computer—so that the attacker cannot obtain information about the applications running. Maya is very effective at obfuscating application activity, and has already thwarted a newly-developed remote power attack.

Acknowledgments

This work was supported by NSF grant CNS 1763658. We thank Kevin Colravy for giving us access to and setting up the electrical line tapping apparatus, and Dipanjan Das for providing the instrumentation software for this experiment. We are grateful to Prof. Tarek Abdelzaher for providing access to his cloud computing testbed.

References

[1] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shojafar, A. I. A. Ahmed, S. A. Madani, K. Saleem, and J. J. Rodrigues, “A Survey on Energy Estimation and Power Modeling Schemes for Smartphone Applications,” *Int. Journal of Comm. Sys.*, vol. 30, no. 11, p. e3234, 2017.

[2] M. Alam, H. A. Khan, M. Dey, N. Sinha, R. Callan, A. Zajic, and M. Prvulovic, “One&Done: A Single-Decryption EM-Based Attack on OpenSSL’s Constant-Time Blinded RSA,” in *USENIX Security*, 2018.

[3] A. Althoff, J. McMahan, L. Vega, S. Davidson, T. Sherwood, M. Taylor, and R. Kastner, “Hiding Intermittent Information Leakage with Architectural Support for Blinking,” in *ISCA*, Jun. 2018.

[4] Android, “Power Profiles for Android,” Android Open Source Project. [Online]. Available: https://source.android.com/devices/tech/power/

[5] ARM, “ARM$^{®}$ Cortex$^{®}$-A15 Processor.” [Online]. Available: https://www.arm.com/products/processors/cortex-a/cortex-a15.php

[6] ARM, “ARM Security Technology Building a Secure System using TrustZone Technology,” Apr. 2009. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

[7] N. D. P. Avirneni and A. K. Somani, “Countering Power Analysis Attacks Using Reliable and Aggressive Designs,” *IEEE Trans. Comput.*, vol. 63, no. 6, pp. 1408–1420, Jun. 2014.

[8] K. Baddam and M. Zwolinski, “Evaluation of Dynamic Voltage and Frequency Scaling as a Differential Power Analysis Countermeasure,” in *VLSID*, Jan. 2007.

[9] Y. Bao, C. Bienia, and K. Li, “PARSEC 3.0: The Parsec Benchmark Suite,” 2011. [Online]. Available: https://parsec.cs.princeton.edu/parsec3-doc.htm

[10] N. Beck, S. White, M. Paraschou, and S. Naffziger, ““Zeppelin”: An SoC for Multichip Architectures,” in *ISSCC*, 2018.

[11] E. M. Benhani and L. Bossuet, “DVFS as a Security Failure of TrustZone-enabled Heterogeneous SoC,” in *International Conference on Electronics, Circuits and Systems (ICECS)*, Dec. 2018.

[12] D. Brodowski and N. Golde, “Linux CPUFreq Governors,” Online Documentation. [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[13] S. K. Bukasa, R. Lashermes, H. Le Bouder, J.-L. Lanet, and A. Legay, “How TrustZone Could Be Bypassed: Side-Channel Attacks on a Modern System-on-Chip,” in *Information Security Theory and Practice*, G. P. Hancke and E. Damiani, Eds., 2018.

[14] R. Callan, N. Popovic, A. Daruna, E. Pollmann, A. Zajic, and M. Prvulovic, “Comparison of Electromagnetic Side-channel Energy Available to the Attacker from Different Computer Systems,” in *2015 IEEE International Symposium on Electromagnetic Compatibility (EMC)*, Aug. 2015.

[15] Y. Cao, J. Nejati, M. Wajahat, A. Balasubramanian, and A. Gandhi, “Deconstructing the Energy Consumption of the Mobile Page Load,” *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 1, Jun. 2017.

[16] N. Chawla, A. Singh, M. Kar, and S. Mukhopadhyay, “Application Inference using Machine Learning based Side Channel Analysis,” in *IJCNN*, 2019.

[17] Y. Chen, X. Jin, J. Sun, R. Zhang, and Y. Zhang, “POWERFUL: Mobile app fingerprinting via power analysis,” in *INFOCOM*, 2017.

[18] S. S. Clark, H. Mustafa, B. Ransford, J. Sorber, K. Fu, and W. Xu, “Current Events: Identifying Webpages by Tapping the Electrical Outlet,” in *Computer Security – ESORICS 2013*, J. Crampton, S. Jajodia, and K. Mayes, Eds., 2013.

[19] M. Conti, M. Nati, E. Rotundo, and R. Spolaor, “Mind The Plug! Laptop-User Recognition Through Power Consumption,” in *International Workshop on IoT Privacy, Trust, and Security*, 2016.

[20] 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 *HOST*, May 2017.

[21] D. Das, A. Golder, J. Danial, S. Ghosh, A. Raychowdhury, and S. Sen, “X-DeepSCA: Cross-Device Deep Learning Side Channel Attack,” in *DAC*, 2019.

[22] M. Enev, S. Gupta, T. Kohno, and S. N. Patel, “Televisions, Video Privacy, and Powerline Electromagnetic Interference,” in *CCS*, 2011.

[23] FFmpeg Developers, “FFmpeg tool,” http://ffmpeg.org/, 2016.

[24] D. Genkin, L. Pachmanov, I. Pipman, E. Tromer, and Y. Yarom, “ECDSA Key Extraction from Mobile Devices via Nonintrusive Physical Side Channels,” in *CCS*, 2016.

[25] D. Genkin, I. Pipman, and E. Tromer, “Get Your Hands Off My Laptop: Physical Side-Channel Key-Extraction Attacks on PCs,” in *Workshop on Cryptographic Hardware and Embedded Systems - Volume 8731*, 2014.

[26] I. Giechaskiel, K. Rasmussen, and J. Szefer, “C3APSULe: Cross-FPGA Covert-Channel Attacks through Power Supply Unit Leakage,” in *IEEE S&P*, May 2020.

[27] D.-W. Gu, P. H. Petkov, and M. M. Konstantinov, *Robust Control Design with MATLAB*, 2nd ed. Springer, 2013.

[28] R. Gu, P. Wang, M. Zheng, H. Hu, and N. Yu, “Adversarial Attack Based Countermeasures against Deep Learning Side-Channel Attacks,” 2020, arXiv 2009.10568v1. [Online]. Available: https://arxiv.org/abs/2009.10568

[29] M. Guri, B. Zadov, D. Bykhovsky, and Y. Elovici, “PowerHammer: Exfiltrating Data From Air-Gapped Computers Through Power Lines,” *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 1879–1890, 2020.

[30] H. Hlavacs, T. Treutner, J. Gelas, L. Lefevre, and A. Orgerie, “Energy Consumption Side-Channel Attack at Virtual Machines in a Cloud,” in *DASC*, Dec. 2011.

[31] Intel, “Random Number Generator IP Core User Guide,” Feb. 2017. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/documentation/dmi1455632999173.html

[32] M. A. Islam and S. Ren, “Ohm’s Law in Data Centers: A Voltage Side Channel for Timing Power Attacks,” in *CCS*, 2018.

[33] M. A. Islam, S. Ren, and A. Wierman, “Exploiting a Thermal Side Channel for Power Attacks in Multi-Tenant Data Centers,” in *CCS*, 2017.

[34] M. A. Islam, L. Yang, K. Ranganath, and S. Ren, “Why Some Like It Loud: Timing Power Attacks in Multi-tenant Data Centers Using an Acoustic Side Channel,” *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 1, pp. 6:1–6:33, Apr. 2018.

[35] John P Mechalas, “Intel Digital Random Number Generator (DRNG) Software Implementation Guide,” Oct. 2018. [Online]. Available: https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide

[36] T. Kasper, D. Oswald, and C. Paar, “EM Side-Channel Attacks on Commercial Contactless Smartcards Using Low-Cost Equipment,” in *Information Security Applications*, H. Y. Youm and M. Yung, Eds., 2009.

[37] S. K. Khatamifard, L. Wang, A. Das, S. Kose, and U. R. Karpuzcu, “POWERT Channels: A Novel Class of Covert Communication Exploiting Power Management Vulnerabilities,” in *HPCA*, 2019.

[38] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in *Annual International Cryptology Conference*, 1999.

[39] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, “Introduction to differential power analysis,” *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 5–27, Apr. 2011.

[40] D. Lezcano, “Idle Injection,” Linux Plumbers Conference, 2018. [Online]. Available: https://www.linuxplumbersconf.org/event/2/contributions/184/attachments/42/49/LPC2018_-_Thermal_-_Idle_injection_1.pdf

[41] P. Lifshits, R. Forte, Y. Hoshen, M. Halpern, M. Philipose, M. Tiwari, and M. Silberstein, "Power to Peep-all: Inference Attacks by Malicious Batteries on Mobile Devices," *PoPETs*, vol. 2018, no. 4, pp. 141–158, 2018.

[42] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "PLATYPUS: Software-based Power Side-Channel Attacks on x86," in *IEEE S&P*, 2021.

[43] L. Ljung, *System Identification: Theory for the User*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.

[44] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun, "Thermal Covert Channels on Multi-core Platforms," in *USENIX Security*, 2015.

[45] MathWorks, "Find abrupt changes in signal," Accessed: April, 2019. [Online]. Available: https://www.mathworks.com/help/signal/ref/findchangepts.html

[46] 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," in *HASP*, 2013.

[47] Y. Michalevsky, A. Schulman, G. A. Veerapandian, D. Boneh, and G. Nakibly, "PowerSpy: Location Tracking Using Mobile Device Power Analysis," in *USENIX Security*, 2015.

[48] K. Paliwal, A. Agarwal, and S. Sinha, "A Modification over Sakoe and Chiba's Dynamic Time Warping Algorithm for Isolated Word Recognition," in *ICASSP*, vol. 7, May 1982.

[49] S. Pandruvada, "Running Average Power Limit – RAPL," Published: June, 2014. [Online]. Available: https://01.org/blogs/2014/running-average-power-limit--rapl

[50] S. Picek, D. Jap, and S. Bhasin, "Poster: When Adversary Becomes the Guardian – Towards Side-Channel Security With Adversarial Attacks," in *CCS*, 2019.

[51] R. P. Pothukuchi, A. Ansari, P. Voulgaris, and J. Torrellas, "Using Multiple Input, Multiple Output Formal Control to Maximize Resource Efficiency in Architectures," in *ISCA*, Jun. 2016.

[52] R. P. Pothukuchi, J. L. Greathouse, K. Rao, C. Erb, L. Piga, P. Voulgaris, and J. Torrellas, "Tangram: Integrated Control of Heterogeneous Computers," in *MICRO*, Oct. 2019.

[53] R. P. Pothukuchi, S. Y. Pothukuchi, P. Voulgaris, and J. Torrellas, "Yukta: Multilayer Resource Controllers to Maximize Efficiency," in *ISCA*, Jun. 2018.

[54] R. P. Pothukuchi, S. Y. Pothukuchi, P. Voulgaris, and J. Torrellas, "Designing a Robust Controller for Obfuscating a Computer's Power," June 2021, Tech. Rep., University of Illinois at Urbana-Champaign. [Online]. Available: http://iacoma.cs.uiuc.edu/iacoma-papers/isca21_1_tr.pdf

[55] Y. Qin and C. Yue, "Website Fingerprinting by Power Estimation Based Side-Channel Attacks on Android 7," in *TrustCom/BigDataSE*, 2018.

[56] G. B. Ratanpal, R. D. Williams, and T. N. Blalock, "An On-Chip Signal Suppression Countermeasure to Power Analysis Attacks," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 3, pp. 179–189, Jul. 2004.

[57] D. Real, C. Canovas, J. Clediere, M. Drissi, and F. Valette, "Defeating Classical Hardware Countermeasures: a New Processing for Side Channel Analysis," in *DATE*, Mar. 2008.

[58] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, "Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, Mar. 2012.

[59] E. Rotem, "Intel Architecture, Code Name Skylake Deep Dive: A New Architecture to Manage Power Performance and Energy Efficiency," Intel Developer Forum, Aug. 2015. [Online]. Available: https://en.wikichip.org/w/images/8/83/Intel_Architecture%2C_Code_Name_Skylake_Deep_Dive-_A_New_Architecture_to_Manage_Power_Performance_and_Energy_Efficiency.pdf

[60] H. Saputra, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, R. Brooks, S. Kim, and W. Zhang, "Masking the Energy Behavior of DES Encryption [Smart Cards]," in *DATE*, Mar. 2003.

[61] F. Schellenberg, D. R. E. Gnad, A. Moradi, and M. B. Tahoori, "Remote Inter-chip Power Analysis Side-channel Attacks at Board-level," in *ICCAD*, 2018.

[62] Y. S. Shao and D. Brooks, "Energy Characterization and Instruction-level Energy Model of Intel's Xeon Phi Processor," in *ISLPED*, Sep. 2013.

[63] Z. Shao, M. A. Islam, and S. Ren, "Your Noise, My Signal: Exploiting Switching Noise for Stealthy Data Exfiltration from Desktop Computers," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 1, May 2020.

[64] S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and Design*. John Wiley & Sons, 2005.

[65] K. Tiri and I. Verbauwhede, "Design Method for Constant Power Consumption of Differential Logic Circuits," in *DATE*, Mar. 2005.

[66] Z. Toprak-Deniz, M. Sperling, J. Bulzacchelli, G. Still, R. Kruse, S. Kim, D. Boerstler, T. Gloekler, R. Robertazzi, K. Stawiasz, T. Diemoz, G. English, D. Hui, P. Muench, and J. Friedrich, "Distributed System of Digitally Controlled Microregulators Enabling Per-Core DVFS for the POWER8™Microprocessor," in *ISSCC*, Feb. 2014.

[67] S. Trimberger and S. McNeil, "Security of FPGAs in Data Centers," in *International Verification and Security Workshop (IVSW)*, 2017.

[68] T. Trippel, K. G. Shin, K. B. Bush, and M. Hicks, "ICAS: An Extensible Framework for Estimating the Susceptibility of IC Layouts to Additive Trojans," in *IEEE S&P*, May 2020.

[69] A. van de Ven and J. Pan, "Intel Powerclamp Driver." [Online]. Available: https://www.kernel.org/doc/Documentation/thermal/intel_powerclamp.txt

[70] E. Vasilakis, "An Instruction Level Energy Characterization of Arm Processors," 2015. [Online]. Available: https://www.ics.forth.gr/carv/greenvm/files/tr450.pdf

[71] X. Wang, Q. Zhou, J. Harer, G. Brown, S. Qiu, Z. Dou, J. Wang, A. Hinton, C. A. Gonzalez, and P. Chin, "Deep Learning-based Classification and Anomaly Detection of Side-channel Signals," in *Proc. SPIE 10630, Cyber Sensing*, 2018.

[72] 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," 2018, arXiv 1803.05847. [Online]. Available: http://arxiv.org/abs/1803.05847

[73] S. Wei, A. Aysu, M. Orshansky, A. Gerstlauer, and M. Tiwari, "Using Power-Anomalies to Counter Evasive Micro-Architectural Attacks in Embedded Systems," in *HOST*, May 2019.

[74] Xiph.org Video Test Media, "Derf's collection." [Online]. Available: https://media.xiph.org/

[75] Z. Xu, T. Mauldin, Z. Yao, S. Pei, T. Wei, and Q. Yang, "A Bus Authentication and Anti-Probing Architecture Extending Hardware Trusted Computing Base Off CPU Chips and Beyond," in *ISCA*, 2020.

[76] L. Yan, Y. Guo, X. Chen, and H. Mei, "A Study on Power Side Channels on Mobile Devices," in *Proceedings of the Asia-Pacific Symposium on Internetware*, 2015.

[77] J. Yang, F. Dai, J. Wang, J. Zeng, Z. Zhang, J. Han, and X. Zeng, "Countering Power Analysis Attacks by Exploiting Characteristics of Multicore Processors," *IEICE Electronics Express*, vol. adv pub, 2018.

[78] Q. Yang, P. Gasti, G. Zhou, A. Farajidavar, and K. S. Balagani, "On Inferring Browsing Activity on Smartphones via USB Power Analysis Side-Channel," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 5, pp. 1056–1066, May 2017.

[79] S. Yang, P. Gupta, M. Wolf, D. Serpanos, V. Narayanan, and Y. Xie, "Power Analysis Attack Resistance Engineering by Dynamic Voltage and Frequency Scaling," *ACM Trans. Embed. Comput. Syst.*, vol. 11, no. 3, pp. 62:1–62:16, Sep. 2012.

[80] 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 *DATE*, 2005.

[81] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "AppScope: Application Energy Metering Framework for Android Smartphones Using Kernel Activity Monitoring," in *USENIX ATC*, 2012.

[82] W. Yu, O. A. Uzun, and S. Köse, "Leveraging On-chip Voltage Regulators as a Countermeasure Against Side-channel Attacks," in *DAC*, 2015.

[83] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones," in *International Conference on Hardware/Software Codesign and System Synthesis*, 2010.

[84] L. Zhang, L. Vega, and M. Taylor, "Power Side Channels in Security ICs: Hardware Countermeasures," 2016, arXiv 1605.00681. [Online]. Available: https://arxiv.org/abs/1605.00681

[85] M. Zhao and G. E. Suh, "FPGA-based Remote Power Side-channel Attacks," in *IEEE S&P*, 2018.