Maya: Using Formal Control to Obfuscate Power Side Channels

Raghavendra Pradyumna Pothukuchi, *Yale University, New Haven, CT, 06511, USA*
Sweta Yamini Pothukuchi, *University of Illinois at Urbana-Champaign, Urbana, IL, 61801, USA*
Petros G. Voulgaris, *University of Nevada, Reno, NV, 89557, USA*
Alexander Schwing and Josep Torrellas, *University of Illinois at Urbana-Champaign, Urbana, IL, 61801, USA*

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 article 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 consumption 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 and hardware. We implement Maya on three machines using only privileged threads against machine learning based attacks, and show its effectiveness and ease of deployment. Maya has already thwarted a newly developed remote power attack.

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, location, browser, and camera activity, and encryption keys.[1,2] Many types of platforms have been successfully attacked using these physical channels, including smartphones, personal computers, cloud servers, multitenant datacenters, and home appliances.[1]

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[2]), estimating power from unprivileged information, and analyzing code to estimate energy consumption. They can also use hardware methods, such as direct probing, antennas, indirect measurement by tapping electrical outlets and power supply networks, trojan chips, field-programmable gate arrays (FPGAs), and circuits.[3,4] 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.[3] 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. In practice, there are many attacks

Digital Object Identifier 10.1109/MM.2022.3166886
Date of publication 12 April 2022; date of current version 30 June 2022.

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.[1]

THIS ARTICLE DESCRIBES MAYA, A NEW DEFENSE TECHNIQUE THAT USES FORMAL CONTROL TO INTELLIGENTLY RESHAPE THE POWER DISSIPATED BY A COMPUTER IN AN APPLICATION-TRANSPARENT MANNER.

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 dynamic voltage and frequency scaling (DVFS) levels are unsuccessful because they do not completely mask application activity.[1,3]

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

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.

To address this problem, this article describes *Maya*, a new defense technique that uses *formal control*[5] to intelligently *reshape* the power dissipated by a computer in an application-transparent manner. 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.

Maya can be implemented in privileged software, firmware, or simple hardware. It relies on commonly available actuators to change power, such as the DVFS level, injection of idle execution cycles, and a custom "balloon" application whose power consumption can be increased on demand. In this article, we implement Maya using only privileged software. This is the first defense against power side channels that is readily deployable and application transparent.

We evaluate Maya against statistical and machine learning-based power analysis attacks on three different machines, in one case tapping an electrical power outlet. We show Maya's high effectiveness. Maya is the first application of formal control to side-channel defense. Maya is publicly available at https://github.com/mayadefense/maya and the design of its formal controller is detailed in http://iacoma.cs.uiuc.edu/iacoma-papers/isca21_1_tr.pdf.

Shao *et al.*[3] recently described 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 of the building, at a distance of 90 ft, tapping on the same power network. The attacker samples voltage with an oscilloscope every 2 μs and, over a period of 33 ms, is able to decode one bit of information from the victims. Shao *et al.* then deployed a power-reshaping defense based on Maya that acts every 40 ms, and show that it completely thwarts their covert channel.

BACKGROUND

Physical Side Channels

Physical side channels, such as 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,[1] to identify the running application, the length of passwords on smartphones, the browser activity on personal computers, to disrupt operation in multitenant datacenters, and even to recover encryption keys from a cryptosystem.

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.[1] Temperature and EM emissions are related to the computer's power, and leave similarly analyzable patterns.

Attackers can capture physical signals in many ways, most of which are nonintrusive. For example, in PLATYPUS,[2] attackers can use a malicious application that reads unprivileged hardware and operating system (OS) counters for power or temperature. 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.[6] When power/thermal counters are unavailable, attackers can estimate power from OS metrics like utilization or from code analysis. Malicious smart batteries are another source of energy counters.[1]

Power can also be measured by tapping ac electricity outlets, power distribution units, and public USB charging

booths. If proximity to the victim is possible, low-cost infrared thermometers and antennas can be used to read temperature and EM emissions, respectively. With direct access to the computer, attackers can use multimeters or oscilloscopes. Such high-end equipment is usually necessary to extract encryption keys.

Trojan hardware, such as chips, co-processors, FPGAs, and other IP modules like chiplets that are co-located with the target chip can also surreptitiously measure the target's chip-level power or temperature. Cloud systems share FPGAs across processors and accelerators, and can be exploited for remote power measurement.[4] In multicore systems, the hierarchical power management policies can be abused to act as power covert channels between cores.

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.[3] The attacker needs no physical access, and can use existing commercial equipment.

To extract sensitive information from signals, attackers can apply machine learning (neural networks), signal processing, and statistical analysis techniques.[1] 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 on a single trace, or differential power analysis over thousands of traces.

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.[1] For cryptographic keys, it is typically necessary to record and analyze signals with samples at intervals of a few microseconds or less.[7]

State-of-the-Art Defenses

Prior defenses against power side-channel attacks have mostly targeted encryption circuits. They try to mask activity information by keeping physical signals at constant levels or by adding noise.[8] Unfortunately, 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[8] or randomizing DVFS in the encryption circuits is easily countered by averaging multiple signal samples. Furthermore, some of these circuit defenses first measure the encryption circuit's power and then change their 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.

It is possible to implement the software versions of these defenses to protect against information leaking through chip- 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. 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.

Formal Control Techniques

Using formal control,[5] 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.

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 $x(T+1)$

$$\begin{aligned} x(T+1) &= A \times x(T) + B \times \Delta y(T) \\ u(T) &= C \times x(T) + D \times \Delta y(T) \end{aligned} \quad x(0) = 0. \tag{1}$$

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

Designers specify multiple parameters in the control system.[5] 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).[9] With these parameters, controller design is automated.[10]

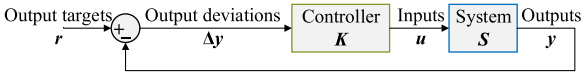

FIGURE 1. Control loop.

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, keystrokes typed, and browser data accessed. This threat model covers the majority of attacks described earlier (e.g., the work of Lifshits *et al.*,[1] Shao *et al.*,[3] Giechaskiel *et al.*,[4] and Masti *et al.*[6]), except for those attacks identifying encryption keys.[7] 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.

MAYA: 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 article, 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 article, we propose the new approach of using *formal control* to reshape power.

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 that is 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 the sensors. Then, it actuates all the inputs so that power is brought to the target.

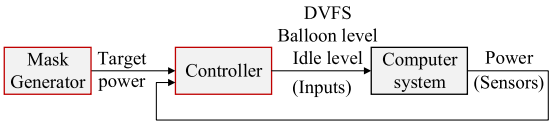

FIGURE 2. High-level architecture of Maya.

The inputs that the controller actuates are the levels of DVFS, the *balloon* task, and the idle activity. The balloon task performs power-consuming operations (e.g., floating-point operations) in a tight, tunable loop. The balloon level determines the fraction 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 1. The table shows two types of power side-channel environments, which we call *InScope* and *OutOfScope*. Our envisioned Maya design targets the InScope environment.

In InScope, attackers measure power with methods like reading counters or tapping electrical power outlets, even with oscilloscopes.[3] The signal analysis is at the granularity of milliseconds and, therefore, one can use typical matrix-based controllers, as described in the "Formal Control Techniques" section. They are implemented in firmware or privileged software. The controller can respond in 5–10 μs, setting the DVFS

TABLE 1. Two types of power side-channel environments.

Characteristic	InScope	OutOfScope
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

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 article, and is relevant because it is widely used.

Table 1 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 precomputed 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 ≈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 article.

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. Simple approaches cannot adapt to the changes in power arising from the application itself. A formal controller makes more informed power changes at every interval, based on history. Specifically, the action taken at time T is a function of the tracking error observed at time T and the controller's *state* [$x(T)$ in (1)]. The state is an accumulation of the controller's experience in regulating the computer's power.

Moreover, obtaining the controller's matrices (A, B, C, and D) involves capturing the response 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, which increases control effectiveness.

Finally, the controller can change multiple inputs at a time, which increases control accuracy. Thus, it can keep power close to the target even when runtime behavior is unpredictable,[5] which is often the case with computers. If the target signal function is chosen appropriately, the attacker will be unable to obtain application information.

Generating Effective Targets

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 will 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 the masked signal's Fourier transform). 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. Second, the mask must have repetitive activity with varying periodicity. This will create several *peaks* in the power signal's Fourier transform. Applications naturally create peaks in the Fourier transform domain if they have loops. By introducing repetitive activity, any natural peaks are overwritten or hidden. Finally, the phase transitions must have different variation rates, ranging from smooth to abrupt. With such ranges of variation, the Fourier transform of the mask will be *spread* over a range of frequencies. If the mask has the abovementioned three properties, the resulting power signal will have many artificially induced change-points that will erase or hide the original ones.

We find that a randomly changing *Gaussian sinusoid* signal, which is the addition of a Gaussian distribution and a sinusoid, has all the properties we want. It changes the mean and variance in the time domain, and has spread and peaks in the frequency domain. In the frequency domain, the Gaussian has a noisy spectrum that is spread across a continuous range of values. In contrast, the sinusoid has sharp and tall peaks. Therefore, the combination of the two signals results in a spectrum that has peaks that are both large and spread across a range. This is the mask that we propose. More details are found in our conference paper.

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 distortions even if they know Maya's defense, as long as they cannot reproduce the random numbers used by Maya.

IMPLEMENTATION OF MAYA

We implement Maya to protect the three different computers listed in Table 2. Sys1 is a consumer-class machine with six physical cores, each with 2-way SMT, totaling 12 logical cores. Sys2 is a server with two sockets, each having 10 cores of 2-way SMT, for a total of 40 logical cores. Sys3 is another consumer-class machine with four physical cores, each with 2-way SMT. On all systems, the architecture of Maya is the same (see Figure 2). We target the InScope attack environment of Table 1. The controller and mask generator run as privileged software.

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 running average power limit (RAPL) counters every 20 ms. It actuates on three inputs: the DVFS level of all cores, percentage of idle activity, and balloon power level.

DVFS levels are set through the `cpufreq` utility. The idle activity level is changed using Intel's `powerclamp` driver interface. 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 ≈10 ms. The balloon threads are created with OpenMP, and run with root priority.

TABLE 2. Implementation platforms.

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

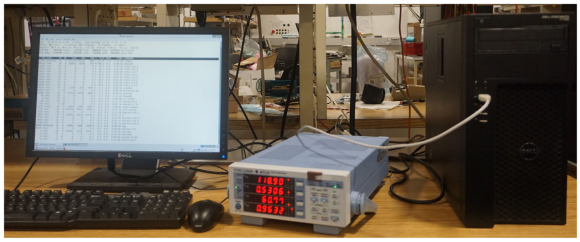

FIGURE 3. Tapping an ac electrical outlet.

We design Maya's controller following standard procedures[9] and using two applications from PARSEC 3.0 (*swaptions* and *ferret*)[a] and two from SPLASH-2x (*barnes* and *raytrace*) running on Sys1 as training applications.

ATTACKS AND DEFENSES

We consider multiple common attacks based on machine learning as listed in Table 3. 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.[1]

To defend, 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 the sampling interval is 50 ms because the measurements are taken from an ac power outlet cycling at 60 Hz. Figure 3 shows a picture of our Sys3 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.

We use a three-layer multilayer perceptron (MLP) neural network for classification in the three attacks. The network uses ReLU units for its hidden layers and the output layer uses Logsoftmax. This network takes the power traces over time for the application detection and video identification attacks. For webpage identification, the network is trained with the fast Fourier transform (FFT) of the power signals recorded when browsing popular websites.

[a]PARSEC 3.0: https://parsec.cs.princeton.edu/parsec3-doc.htm.

TABLE 3. 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

TABLE 4. Designs compared.

Design	Description
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 (see Figure 2) with a constant mask
Maya GS	Maya with a Gaussian sinusoid mask (Proposed)

Defenses

Our *Baseline* environment is a high performance insecure machine without any noise or mask to obfuscate the application to an attacker. On top of this environment, we build the software defenses shown in Table 4. Specifically, 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.

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 4 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.

EVALUATION HIGHLIGHTS

To highlight our results, we use the attack that detects the application running. We run a total of 11 applications from PARSEC 3.0 and SPLASH-2x. We compare the different defenses from Table 4 using confusion matrices. A confusion matrix is a table where each row corresponds to the true labels of the applications (0–10 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 4. For example, the entry in the 0th row and 1st 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.

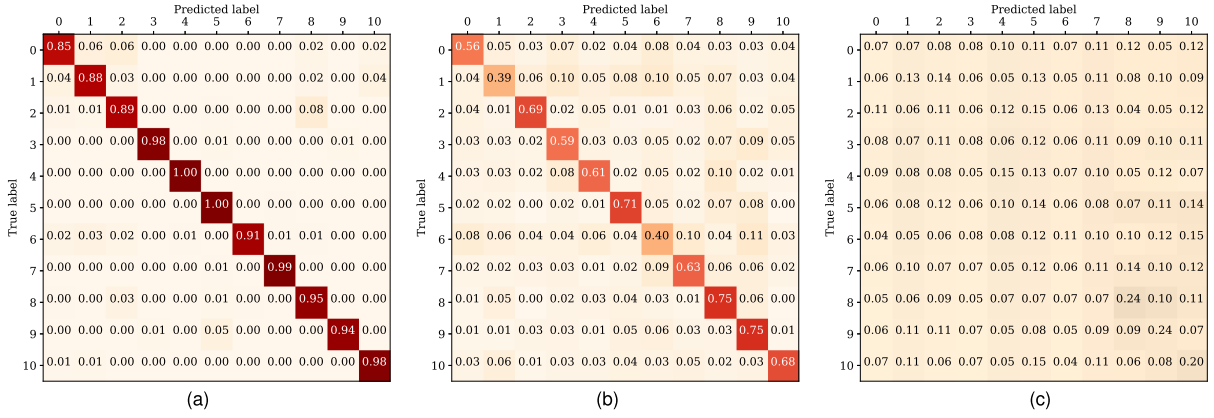

FIGURE 4. Confusion matrices for detecting the running application from power signals. (a) Random inputs (avg. accuracy 94%). (b) Maya Constant (avg. accuracy 62%). (c) Maya GS (avg. accuracy 14%).

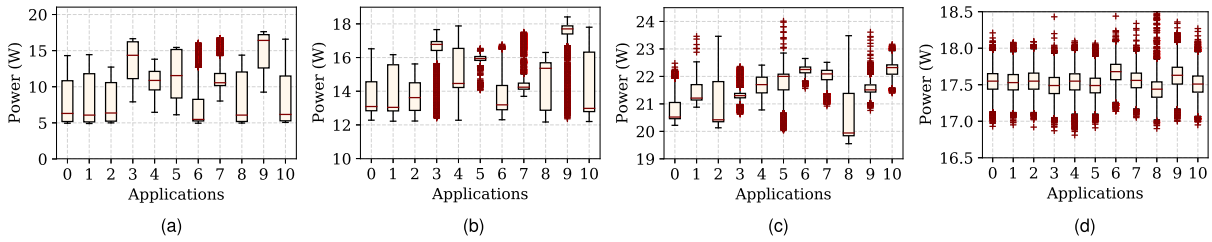

FIGURE 5. Summary statistics of the average of 1,000 signals. The *Y*-axis of each chart is drawn to a different scale. (a) Noisy baseline. (b) Random inputs. (c) Maya constant. (d) Maya GS.

Figure 4 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. To defend 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 the "Generating Effective Targets" section, ensuring constant power causes information leaks at application change points.

Finally, the attack on *Maya GS* has only 14% average accuracy. This is close to the 9% prediction accuracy of random chance. Overall, *Maya GS* achieves excellent obfuscation. The Gaussian sinusoid mask and the formal controller 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.

Signal Statistics and Analysis

For more insights, we analyze the signals produced by the defenses of Table 4 using signal summary statistics and change-point analysis.

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 out. 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 5 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 25th to 75th 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* [see Figure 5(a)], the value distribution is distinct for each application and acts like a fingerprint. In *Random inputs* [see Figure 5(b)], the boxes shrink in size, but the relative difference remains the same. With *Maya constant* [see Figure 5(c)], 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* [see Figure 5(d)] 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.

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[b] to identify the phases found in the reshaped signals. We present the highlights of this analysis using the *blackscholes* application.

With *Noisy baseline* [see Figure 6(a)], four phases of the application are clearly seen: 1) sequential, 2) parallel, 3) sequential, and 4) idleness after the

[b]https://www.mathworks.com/help/signal/ref/findchangepts.html

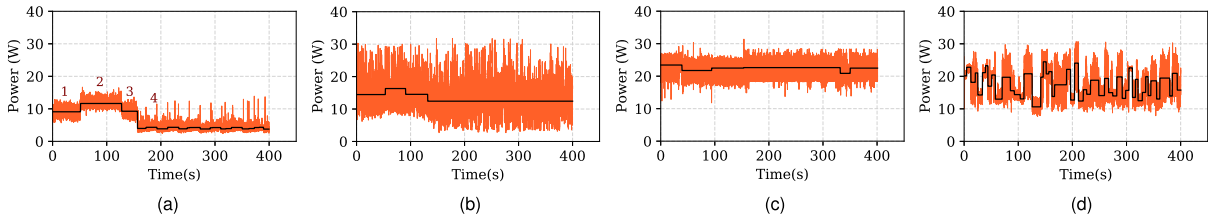

FIGURE 6. Change-point detection in blackscholes using traces over time. Chart (a) shows all four phases being detected. (a) Noisy baseline. (b) Random inputs. (c) Maya constant. (d) Maya GS.

application ends. The difference between the phases is not too large, and there is some noise because of the interference with idle and balloon activity. However, the algorithm detects the four major phases.

With *Random inputs* [see Figure 6(b)], 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* [see Figure 6(c)], the power profile is mostly around 25 W because the mask is held constant at that value. However, the algorithm can still recover 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* [see Figure 6(d)], 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.

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, 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 microsecond.

Running the controller involves computing (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 $x(T)$ (1). It can be shown that running the controller needs $\approx$200 fixed-point operations, which complete within 1 μs. The controller needs less than 1 kb 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 millisecond or more.

Impact on Application Power and Performance

We run the PARSEC and SPLASH-2x applications on Sys1 with *Maya GS* and *Baseline* (see the "Defenses" section). *Baseline* runs applications at the highest available frequency without inserting idle or balloon threads. We measure power and execution time. Compared to the high-performance *Baseline*, the average power consumed by the applications with *Maya GS* is 30% lower. The power is lower because *Maya GS* uses idle threads and sometimes low DVFS values. On the other hand, the average execution time is 47% higher, because of the idle and balloon threads, and the low DVFS values. The total energy consumed by *Maya GS* is approximately the same as that of *Baseline*.

We believe that the slowdown of *Maya GS* relative to *Baseline* is acceptable given the high level of security that it provides without needing any hardware support at all.

PROMISING RESEARCH DIRECTIONS

Two promising research directions are to decrease Maya's performance and energy overhead, and to widen Maya's applicability to other side channels.

Lowering Maya's Performance and Energy Overhead

The performance and energy overhead of Maya may be prohibitive for systems, such as those involving

battery-powered and Internet of Things devices. Among the methods to reduce this overhead, one approach is to activate Maya only during sensitive sections of applications, such as when executing login routines. This approach is similar to how power governors can be invoked in Linux. Even though such selective activation reveals the onset of a sensitive phase, information during that phase is protected.

Another approach to reduce the application slowdown is to run the application and the 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 the software overhead of executing the power balloon threads.

MAYA NEEDS FEW RESOURCES TO OPERATE, MAKING IT ATTRACTIVE FOR FIRMWARE, SOFTWARE, OR EVEN HARDWARE IMPLEMENTATIONS.

A more complex solution to reduce overhead would be to utilize knowledge about an application to minimize the distortions. Currently, Maya assumes that information-carrying patterns may exist anywhere in an application. Therefore, it thoroughly reshapes the application's power activity. However, recent research has indicated that it is possible to use additional knowledge of an application—e.g., gathered with machine learning techniques—to identify particular locations where information-leaking patterns occur and also to identify any specific features those patterns have (e.g., Gu *et al.*'s work[11]). This knowledge can reduce the locations and magnitude of the distortions that Maya introduces. Consequently, the overall energy and performance impact may also come down. We consider the prospects of jointly using machine learning and Maya's formal control to be exciting new work in security.

Simultaneously Tackling Multiple Sources of Information Leakage

Computers have various sources of information leakage that attackers can exploit. Information may also be identified by correlations across patterns in multiple channels (e.g., memory traffic and power). Previous work[12] has shown that signal reshaping can stop information leaking through side channels, such as memory and network traffic measurements. A key challenge, however, is to *simultaneously* obfuscate information from multiple channels.

MAYA IS THE FIRST DEFENSE AGAINST POWER SIDE CHANNELS THAT IS APPLICATION TRANSPARENT AND READY TO BE DEPLOYED TO PROTECT EXISTING SYSTEMS.

Maya is a promising solution for multichannel obfuscation for several reasons. First, as this article shows, a formal control-based defense is simple and effective. Second, the controller has the capability to regulate multiple outputs at the same time, potentially mitigating multiple side channels *simultaneously*. Third, the Maya design needs few resources to operate, making it suitable for even hardware and firmware implementations, and to scale with more signals. Finally, the use of formal control simplifies verification and can be used to provide the guarantees of protection.

We believe that there is a significant research potential in identifying the best target masks for multichannel obfuscation and in designing multirate controllers that can obfuscate patterns at various timescales across different channels.

CONCLUSION

Maya is the first defense against power side channels that is application transparent and ready to be deployed to protect existing systems. It needs few resources to operate, and the knobs that it changes are supported by virtually every computer today.

In our conference paper, we show that Maya prevents the profiling of instructions from power signatures. Such capability prevents the recent PLATYPUS attack[2] from recovering cryptographic keys from RAPL energy counters.

Maya has already been used by a different research group to block a new power attack. Specifically, Shao *et al.*[3] described a covert-channel attack across a building's power network.

We hope that Maya will spur many researchers to use advanced formal control for security. We consider Maya-based signal reshaping to be a powerful defense mechanism when attackers can gather signals from a victim.

REFERENCES

1. P. Lifshits *et al.*, "Power to peep-all: Inference attacks by malicious batteries on mobile devices," *Proc. Privacy Enhancing Technol.*, vol. 2018, no. 4, pp. 141–158, 2018.
2. M. Lipp *et al.*, "PLATYPUS: Software-based power side-channel attacks on x86," in *Proc. IEEE Symp. Secur. Privacy*, 2021, pp. 355–371.
3. 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, pp. 1–39, May 2020.
4. I. Giechaskiel, K. Rasmussen, and J. Szefer, "C3APSULe: Cross-FPGA covert-channel attacks through power supply unit leakage," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 1728–1741.
5. S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and Design*. Hoboken, NJ, USA: Wiley, 2005.
6. R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun, "Thermal covert channels on multi-core platforms," in *Proc. USENIX Secur. Symp.*, 2015, pp. 865–880.
7. P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power," *J. Cryptographic Eng.*, vol. 1, no. 1, pp. 5–27, Apr. 2011.
8. 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.
9. R. P. Pothukuchi, S. Y. Pothukuchi, P. Voulgaris, and J. Torrellas, "Yukta: Multilayer resource controllers to maximize efficiency," in *Proc. Int. Symp. Comput. Archit.*, 2018, pp. 505–518.
10. R. P. Pothukuchi, A. Ansari, P. Voulgaris, and J. Torrellas, "Using multiple input, multiple output formal control to maximize resource efficiency in architectures," in *Proc. Int. Symp. Comput. Archit.*, Jun. 2016, pp. 658–670.
11. R. Gu, P. Wang, M. Zheng, H. Hu, and N. Yu, "Adversarial attack based countermeasures against deep learning side-channel attacks," 2020, *arXiv:2009.10568*.
12. Y. Zhou, S. Wagh, P. Mittal, and D. Wentzlaff, "Camouflage: Memory traffic shaping to mitigate timing attacks," in *Proc. Int. Symp. High Perform. Comput. Archit.*, 2017, pp. 337–348.

RAGHAVENDRA PRADYUMNA POTHUKUCHI is an associate research scientist and a CRA/NSF computing innovation fellow at Yale University, New Haven, CT, 06520, USA. His research interests include computer architecture and systems, and cognitive science, quantum computing, brain–computer interfaces formal control, energy efficiency, security, machine learning, and compilers. Pothukuchi received a Ph.D. degree from the University of Illinois at Urbana-Champaign, Urbana, IL, USA. He is a Member of IEEE. Contact him at raghav.pothukuchi@yale.edu.

SWETA YAMINI POTHUKUCHI works on simplifying the programming of large distributed-memory systems while providing high performance using multilayer code generation and optimization. Her research interests include compilers and parallel programming. Pothukuchi received a Ph.D. degree from the University of Illinois at Urbana-Champaign, Urbana, IL, USA. Contact her at sweta.yamini@gmail.com.

PETROS G. VOULGARIS is the chair, founding aerospace program director, and Victor LaMar Lockhart Professor in mechanical engineering at the University of Nevada, Reno, NV, 89557, USA. His research interests include the general area of robust and optimal control and coordination of autonomous systems. Voulgaris received a Ph.D. degree in aeronautics and astronautics from the Massachusetts Institute of Technology, Cambridge, MA, USA. He is a Fellow of IEEE. Contact him at pvoulgaris@unr.edu.

ALEXANDER SCHWING is an assistant professor at the University of Illinois, Urbana-Champaign, Urbana, IL, USA. His research interests are computer vision and machine learning, where he has worked on scene understanding, inference and learning algorithms, deep learning, image and language processing, and generative modeling. Schwing received a Ph.D. degree in computer science from ETH Zurich, Zurich, Switzerland. Contact him at aschwing@illinois.edu.

JOSEP TORRELLAS is the Saburo Muroga Professor of computer science at the University of Illinois, Urbana-Champaign, Urbana, IL, USA. His research interests include computer architectures for parallelism, energy efficiency, programmability, and security. Torrellas received a Ph.D. degree from Stanford University, Stanford, CA, USA. He is a Fellow of IEEE. Contact him at torrella@illinois.edu.