Leveraging Side-channel Information for Disassembly and Security

JUNGMIN PARK, Florida Institute for Cybersecurity Research, University of Florida, USA
FAHIM RAHMAN, Florida Institute for Cybersecurity Research, University of Florida, USA
APOSTOL VASSILEV, National Institute of Standards and Technology, USA
DOMENIC FORTE, Florida Institute for Cybersecurity Research, University of Florida, USA
MARK TEHRANIPOOR, Florida Institute for Cybersecurity Research, University of Florida, USA

With the rise of Internet of Things (IoT), devices such as smartphones, embedded medical devices, smart home appliances as well as traditional computing platforms such as personal computers and servers have been increasingly targeted with a variety of cyber attacks. Due to limited hardware resources for embedded devices and difficulty in wide-coverage and on-time software updates, software-only cyber defense techniques, such as traditional anti-virus and malware detectors, do not offer a silver-bullet solution. Hardware-based security monitoring and protection techniques, therefore, have gained significant attention. Monitoring devices using side channel leakage information, e.g. power supply variation and electromagnetic (EM) radiation, is a promising avenue that promotes multiple directions in security and trust applications. In this paper, we provide a taxonomy of hardware-based monitoring techniques against different cyber and hardware attacks, highlight the potentials and unique challenges, and display how power-based side-channel instruction-level monitoring can offer suitable solutions to prevailing embedded device security issues. Further, we delineate approaches for future research directions.[1]

CCS Concepts: • **Security and privacy** → **Side-channel analysis and countermeasures**;

ACM Reference Format:
Jungmin Park, Fahim Rahman, Apostol Vassilev, Domenic Forte, and Mark Tehranipoor. 2019. Leveraging Side-channel Information for Disassembly and Security. *ACM J. Emerg. Technol. Comput. Syst.* 0, 0, Article 0 (2019), 21 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the advent of the Internet of Things (IoT), embedded devices, and networked high performance computation platforms and data centers, various cyber attacks such as malware, ransomware, distributed denial-of-service (DDoS), etc., have become a significant concern in the present world.

[1]DISCLAIMER: This paper is not subject to copyright in the United States. Commercial products are identified in order to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the identified products are necessarily the best available for the purpose.

Authors' addresses: Jungmin Park, Florida Institute for Cybersecurity Research, University of Florida, Gainesville, FL, 32611, USA, jungminpark@ufl.edu; Fahim Rahman, Florida Institute for Cybersecurity Research, University of Florida, Gainesville, FL, 32611, USA, fahim034@ufl.edu; Apostol Vassilev, National Institute of Standards and Technology, Gaithersburg, MD, USA, apostol.vassilev@nist.gov; Domenic Forte, Florida Institute for Cybersecurity Research, University of Florida, Gainesville, FL, 32611, USA, dforte@ece.ufl.edu; Mark Tehranipoor, Florida Institute for Cybersecurity Research, University of Florida, Gainesville, FL, 32611, USA, tehranipoor@ece.ufl.edu.

1550-4832/2019/0-ART0 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

All network-connected devices – from high performance PCs and cloud servers to low-cost and lightweight IoT and embedded devices – are susceptible to such cyber and hardware attacks. Since low-cost loT devices have limited resources, such as low processing, memory capability and energy, deploying sophisticated defense mechanisms is extremely difficult and financially infeasible making them attractive targets to adversaries [72]. The attacks and vulnerabilities are expected to be even more with an estimated 26 billion connected devices by the end of 2020 [49]. Many IoT devices can be infected by a botnet malware and be used as 'zombies' for distributed denial-of-service (DDoS) attacks [4]. An adversary can extract private data such as credit card numbers and passwords to log into sensitive portals hosted by these devices, or obtain unauthorized control to critical infrastructure such as power plants by malware such as Stuxnet [5]. Furthermore, a malware infection through Internet or physical access can cause malfunction of medical devices and smart cars, as well as personal computers and cloud devices. It is with no doubt that such successful cyber attacks can lead to serious economic loss, infrastructural damage, or injury to humans [55].

It is apparent that thwarting the threats and vulnerabilities against cyber attacks requires lasting attention. In particular, runtime monitoring of computing devices from all domains is highly necessary to detect malware, unauthorized access, and illegitimate controls and applications. Such monitoring techniques can be either *software-based* and *hardware-based*. The *software-based* method mostly performs control-flow integrity (CFI) assessment [10] which can monitor unexpected changes by a malicious code by analyzing the runtime control-flow graph (CFG). For instance, in order to enforce the software-based CFI, machine-code instructions (or instrumentations) for an indirect function call and a corresponding function return can be rewritten in a way that unique IDs are assigned for the source and the destination functions, and validity of the IDs are checked for the integrity verification [1]. However, such software-based methods have disadvantages of performance degradation (e.g., CFI in [1] and program shepherding in [38] have 45 % and 100 % performance overhead for the SPEC2000 benchmark program `crafty` [34], respectively) and unavailability to devices with resource-constrained architecture. Further, an attacker can potentially evade such countermeasures. For example, while non-executable data (NXD) and non-writable code (NWC) of software-based CFI can be protected by page-based access control (e.g., via write-xor-execute, $W \oplus E$), an attacker can disable it with a syscall command to `mprotect()/VirtualProtect()` [18]. Traditional signature-based software monitors for standard computing devices, such as common anti-virus or malware detection software, do not provide sufficient protections as they face difficulties in detecting zero-day threats and the embedded device may not have sufficient resources (e.g., memory to store and update known malware signatures) to support such schemes [30].

On the other hand, *hardware-based* methods usually use embedded and/or independent trusted hardware to observe the behavior of a program running on the device under monitor. Hardware-based CFI architecture integrates hardware monitors into processor's pipeline stages or hardware debug interface such as the Joint Test Action Group (JTAG) or scan chain is used to validate CFI at runtime. Hardware-based detection methods require smaller overhead for resource and latency compared to the software-based counterparts. However, such techniques heavily rely on machine learning (ML) techniques that need extensive training and validation and, therefore, may require additional hardware supports [56, 66, 70]. Based on the existing limitations, it is evident that neither the prevailing software-only or hardware-only techniques can provide a complete defense against the numerous threats and attacks.

Recently, researchers have paid more attention to hardware-based methods that leverage side-channel leakages such as power consumption and EM [14, 19, 48, 50, 51, 68]. Such side-channel leakages can be used for revealing secret data residing inside the device, e.g., private key used for encryption. This is traditionally known as side-channel attacks (SCA). However, side-channel information can also be leveraged for analyzing the status of a computing system at runtime. One crucial way to do so is to *disassemble* the runtime code, i.e., to translate side-channel information into assembly codes consisting of an instruction and operands (such as the source register or the destination register) in a timely sequence. It can be used for verification of programs running on the device. In this paper, we refer to this as a side-channel disassembler (SCD). Using a SCD, the control flow of the target device can be tracked in the coarse- or fine-grained granularity, both in software and hardware domains, without any performance degradation of the target device. A SCD is multipurpose as it allows to enforce decoupled monitoring of targeted devices. For example, it can analyze the runtime status of the device and can detect potential malware and security breaches which is of concern to many. A Defense Advance Research Program Administration (DARPA) program called Leveraging the Analog Domain for Security (LADS) [16] that is similar to this concept attempts to achieve security and protection using different side channels that are analog in nature. Additionally, a SCD can potentially perform hardware-firmware attestation and firmware reverse engineering, even against firmware that is protected by encryption and anti-tamper technologies. Such a SCD-based reverse engineering may be considered as a potential threat for Intellectual Property (IP) theft, whereas the same technique can be used for protection through firmware/software verification and authentication. One way to maintain the integrity of the underlying firmware is to verify whether the firmware is modified while running on the device by monitoring the side-channel information. Since disassembly techniques do not require additional hardware to be embedded in the original processor architecture, legacy devices without internal hardware monitors, such as performance counters or JTAG, can be greatly benefited. Such devices, therefore, can be protected by attaching an external side-channel monitor capable to collect necessary power or EM signature [50].

To date, existing SCD techniques have mostly been implemented on low-performance microcontrollers due to obvious technical limitations, such as noise-free data acquisition, additional hardware (e.g, oscilloscope with high sampling rates and high bandwidths, high-gain amplifiers, or filters) cost, complex data processing, etc., that get even more pronounced for high performance processors used in personal computers and smartphones. For example, noise free data collection from high-performance multi-core processors is still a challenge and existing SCD techniques are not always readily scalable for complex systems. One may also argue that it is not economically feasible to employ expensive and bulk instruments for collecting side-channel leakages for low-cost IoT and embedded devices using simpler microcontrollers or processing units. As one can see, resolving the prevailing challenges requires a unified and holistic effort from the research community. We firmly believe that by overcoming the challenges, this technique can offer a comprehensive solution to present day cyber-threats in all domains of electronic devices.

In this work, we focus on analyzing hardware-based monitoring techniques leveraging power side-channels for IoT and embedded devices and highlight potential applications and prevailing challenges for supporting high-performance computing devices as well. We first present a taxonomy of hardware-based monitoring methods to summarize and compare existing techniques based on different threat models. Next, we introduce the assembly-level instruction monitoring and disassembly technique for embedded devices using power side-channel leakage. We also provide potential applications such as malware detection and firmware reverse engineering using the disassembly technique. Finally, we outline the unique challenges in this field and propose high-level approaches for future research directions.

The rest of the paper is organized as follows. Section 2 discusses the adversarial threat models focusing on different attacks and attackers' capabilities. Section 3 presents the taxonomy of hardware-based monitors and discusses existing side-channel monitors. Section 4 discusses potential applications leveraging the proposed technique. Section 5 provides challenging problems in this field and the future research directions. Finally, we conclude in Section 6.

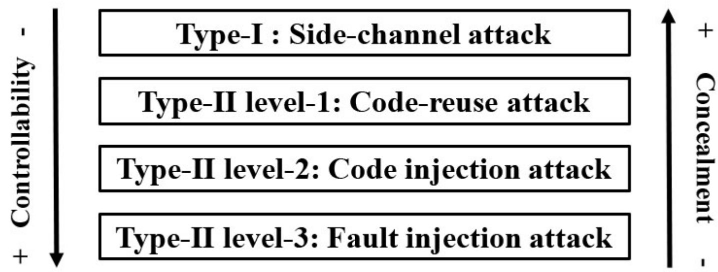

Fig. 1. Controllability and concealment of adversary models.

2 ADVERSARIAL CAPABILITIES AND DEFENSE STRENGTH

Before analyzing different side-channel information based monitoring and security schemes in detail, it is imperative that we understand the underlying threats from the adversary, as well as different levels of defense that may be supported by various techniques against such threats. We note that the associated threats may have similar components across different types of devices such as IoT and embedded modules as well as high-performance computing platforms.

We define two different adversarial models – *Type-I* and *Type-II* – based on the attackers' capabilities. We assume that *Type-I* attackers only have access to the device under attack for information gathering. They cannot manipulate the operation of the given device, i.e., they cannot control or modify any of the data memory and code memory by adversarial code injection or malware. We assume that the attackers can access only the data input/output ports and power pins of the device under attack, and the target device can be modeled as a black box if necessary. Traditional non-invasive side-channel attacks, such as Differential Power Analysis (DPA), Correlation Power Analysis (CPA), and profiling attacks, are likely to be performed by *Type-I* attackers.

On the other hand, *Type-II* attackers can launch active runtime attacks as they have the ability to control or modify data memory or code memory depending on the capabilities (controllability) available to manipulate the original control flow [18]. However, as one can understand, not all *Type-II* attackers have the same amount of capabilities and control over the device under attack. For example, we assume that *Type-II level-1* attackers can control only data memory which includes the stack and the heap, but they cannot modify the code memory. This means that the attackers cannot perform code injection or code tampering attacks. By modifying data memory and executing an indirect branch, attackers can redirect control flow of existing code with a malicious result in the code memory. Code-reuse attack (CRA), such as the return-to-libc or return-oriented programming, is among such *Type-II level-1* attacks [8]. Further, we assume that *Type-II level-2* attackers have control over both the data memory and the code memory. Such attackers can, therefore, inject malicious code or data structure, referred to as code injection attack [21]. finally, we assume that the *Type-II level-3* attackers can control all memory elements including registers and flip-flops. They can perform non-invasive fault injection attacks such as glitching attack [6], temperature fault attack [32], and CLKSCREW attack [67] as well as other lower level attacks. Additionally, the attackers can perform semi-invasive attacks on the device.

We note that as the attackers' control and capability over the device under attack increases, the concealment of the attack decreases since higher level attacks become more prominent and tend to show activities and properties that diverge enough for a legit user to identify easily (e.g., the device under attack may become unresponsive, malfunction, or show unusual network activity). Therefore, the quality and accuracy of the defense mechanism employed is highly related to the

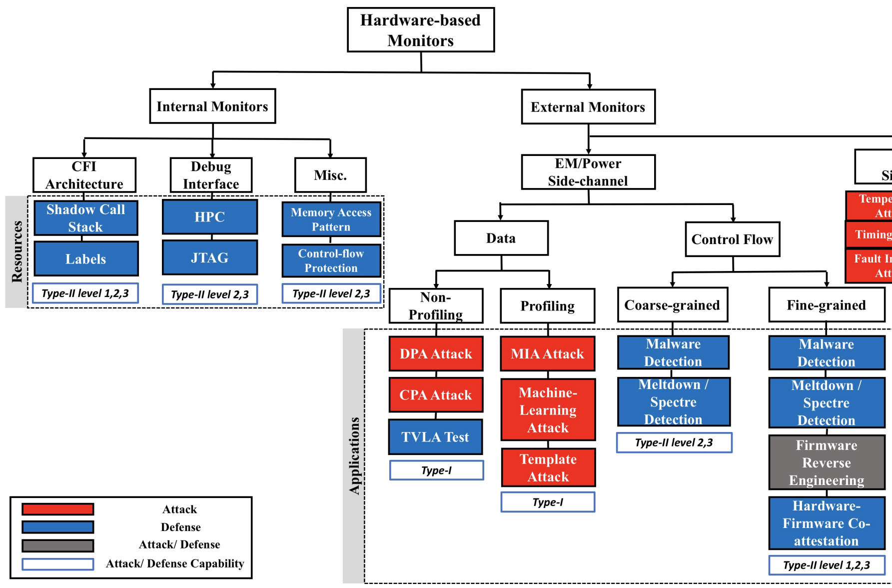

Fig. 2. Taxonomy of hardware-based monitors.

threat under consideration. A security monitoring and defense scheme based on side-channel information may offer a generic coarse-grain monitoring for defending against attacks that utilize non-/semi-invasive techniques such as fault injections (i.e., *Type-II level-3* attacks); however, it may not be suitable for detecting more subtle attacks that modify the original data/control flow to make divergent operations from the legit one (e.g. *Type-II level-1,2* attacks). Henceforth, a more powerful monitoring mechanism is required to detect more concealable threats. Fig. 1 shows the controllability and concealment of the adversary model. We note that the prevailing defense mechanisms, as mentioned in Section 3, are often geared towards selective threat models and fail to offer comprehensive protections against cross-layer threats from all levels and types.

3 TAXONOMY AND EXISTING HARDWARE-BASED MONITORS

Side-channel information, i.e., information that do not directly refer to the functional outcome of the device but may potentially exhibit the activity of the device, can be obtained from different sources such as supply power, EM radiation, temperature, or by utilizing different sensors, registers, and communication channels. Such information capturing monitors can be generally categorized into *internal* monitors, e.g., performance monitoring units (PMUs) with hardware performance counters (HPCs), and *external* monitors, e.g., EM probe and monitors, depending on whether they are integrated into, or external to, the original hardware design. Internal monitors are classified by used resources to estimate the activity of the device and external monitors are classified by the objective such as extracting data and tracking control flow. As shown in Fig. 2, each monitor has a range of attack or defense levels. For the sake of simplicity, we mostly focus on the external side channel information such as power. Details of such monitors are as follows.

3.1 Internal Monitors

The internal hardware-based monitoring method exploits various embedded hardware resources including CFI architecture, debug interface (e.g., scan chain, JTAG, performance monitoring units, etc.) that is common in many modern SoCs or memory access monitors.

3.1.1 CFI Architecture. A CFI architecture such as shadow call stack (SCS) with a buffer can be used to detect tampering with the return address during a function call by comparing it from two different and independent stacks where the address copies are stored [13]. Such a technique can prevent *Type-II level-1* attacks such as code-reuse attacks (e.g., return-oriented programming (ROP) and jump-oriented programming (JOP)) as well as *Type-II level-2 and 3* attacks. Additionally, legit program return addresses can be labeled 'valid' and stored in an isolated memory for future runtime comparison [2]. However, such techniques are potentially vulnerable to control-flow bending (CFB) attacks [17, 18]. Another major drawback is that the secure on-chip memory, i.e. the shadow stack or label state memory where the addresses are stored for integrity comparison, may not be readily available for lightweight IoT devices.

3.1.2 Debug Interface. Hardware debug interfaces can monitor and detect several *Type-II level-2 and 3* attacks. For example, an interface following IEEE-ISTO NEXUS 5001 standards [20] can be used to observe branch target address at runtime to monitor any mismatch from the targets stored in the branch destination table caused by potential malware [27]. However, such a technique usually requires an additional unit to collect and process data from the debug interface. Additionally, performance monitoring units (PMUs) using HPCs can be used for micro-architectural event monitoring for potential anomaly detection [66, 70]. Alam *et al.* [3] proposed machine learning based real-time detection mechanism to deal with security against micro-architectural side-channel attacks including cache-based attacks and branch-prediction-based attacks, which can identify abnormalities in the number of micro-architectural events while those side-channel attacks are being executed. PMUs offer a fine-grain filtering for individual executions and provide a faster response than the software-only counterparts. Also, being an integrated part of the hardware, such monitors operate transparently to any program running on the processor. Being oblivious of the program that is running, HPCs capture true activity information, and, therefore, it is very hard for the adversary to control HPCs for evading the malicious footprint generated due to any external malicious software. However, additional hardware supports as well as extensive training for machine learning classification are required.

3.1.3 Memory Access Monitors. Analyzing memory access patterns using hardware monitors can offer defense against *Type-II level-2 and 3* attacks. In such attacks, an infected program can request suspicious memory access for undercover attacks such as rowhammer attack on DRAM [37]. Yoon *et al.* [75] utilized profiled memory behavior via Memory Heat Map (MHM) collected by an on-chip hardware module called Memometer for malware detection. Xu *et al.* [73] utilized virtual memory access patterns for identifying potential anomalies. In both cases, ML techniques were used to differentiate between malicious and benign programs.

These internal monitoring methods often require additional resources such as a control mechanism to collect and process data from internal monitors as well as heavily rely on machine learning techniques due to limited available information distinguishable features. In addition, real-time detecting algorithms using internal resources may degrade the performance of the target device. Further, legacy devices do not usually contain such internal hardware monitors. Therefore, CFI assessment and monitoring techniques using internal embedded hardware is not readily attainable for legacy and resource constrained devices.

3.1.4 Control-Flow Protection. Werner *et al.* [71] proposed a sponge-based control-flow protection technique which supports the confidentiality of software IP and its authentic execution on IoT devices. Firmware is stored in a sponge function based authenticated encryption scheme [7] in the memory and each instruction is decrypted after the fetch pipeline stage such that correct instructions can be decoded and executed. Since the encryption depends on the previous instruction states and the current instruction, (i.e., control flow), control flow deviation by code-reuse, code-injection and fault-injection results in randomized instructions by incorrect decryption. The randomized instructions can thwart attacker's control. Also, this method prevenst from firmware IP theft due to firmware encryption. However, since side-channel leakage is not considered in such cases, it does not have robustness against side-channel attacks.

3.2 External Side-channel Monitors

The external monitoring methods generally use side-channel leakages such as power consumption, EM radiation, temperature [33], or timing [53] with the measurement and monitoring units being independent from the target device. The objective of the side-channel monitor is to extract private data or estimate control flow (e.g., instruction sequences) at runtime. Side-channel analysis techniques to extract secret data usually involve adversarial intentions, e.g. stealing private encryption keys, etc., to control and exploit the devices and network. Such data extraction attacks can further be classified into *non-profiling* and *profiling* attacks depending on whether a signature profiling is required. Common *non-profiling* attacks are differential power analysis (DPA) [40], and correlation power analysis (CPA) [9] attacks. On the other hand, template attacks [11], mutual information analysis (MIA) [23], and various machine-learning based attacks [31, 57] correspond to *profiling* attacks that analyze and classify the side-channel signature into certain domains for confident extraction of underlying information. Side-channel monitoring methods for data extraction have been used by *Type-I* adversaries and well-studied for the last few decades [60, 62, 77]. From a defense point of view, this monitor can be used to evaluate side-channel leakage of embedded devices by performing side-channel attacks as well as leakage assessment test such as test vector leakage assessment (TVLA) t-test [24].

Another objective of side-channel monitoring can be to validate the *control flow integrity* (CFI). This defensive technique against various attacks can further be classified into *coarse-grained* and *fine-grained* CFI methods based on the granularity of monitored activities. If the CFI design is based on a periodic activity (e.g., loop) in the program [61], a *coarse-grained* estimation of per-iteration execution time using side-channel leakage can be statistically compared to a benign program to ensure the legitimacy of the runtime control flow. In [61], repetitive program activity such as loops is analyzed by the spectrum of EM side-channel signals with spikes at specific frequencies corresponding to the iteration time of the loop. Based on the spectral profiling of a benign program, it is possible to recognize the spectrum of malicious programs. This method, therefore, can be applied for malware detection with repetitive features [51]. A more precise CFI policy is based on instruction-level granularity, which is referred to as a *fine-grained* CFI method. The *fine-grained* CFI monitor can be utilized for reverse engineering of instruction code, also known as an instruction-level disassembler, as well as malware and anomaly detection. As one can see, different hardware monitors (e.g., power-based monitors vs. EM monitors) may lead to different implementations and analysis techniques, nonetheless the basic target applications (attack or defense) remain the same irrespective of the monitor itself.

3.2.1 Side-channel-based Coarse-grained CFI Methods. Clark *et al.* [14] proposed a malware detection technique, called WattsUpDoc, on an embedded medical device and a supervisory control and data acquisition (SCADA) device via power side-channel. WattsUpDoc collects system-wide power consumption data at runtime and identifies anomalous activity using supervised ML algorithms using traces of both the normal and abnormal activities. Since the medical and SCADA devices have a small number of functional states (e.g., idle, booting, shutdown and compound tasks in case of a pharmaceutical compounder), the normal behavior can be characterized at the functional-level granularity and WattsUpDoc can detect abnormal behavior caused by known malware with at least 94 % accuracy and by unknown malware with at least 85 % accuracy. Although this technique does not necessarily perform standard CFI assessment, it is able to distinguish intrusive behaviors caused by potential malware.

Nazari *et al.* [51] proposed a technique called EDDIE that can detect anomalies caused by code-injection attacks in program execution using EM side-channel. In this approach, the authors implement a loop-oriented execution where the control flow graph (CFG) of the program represents the flow from a loop-level state to other loop-level states. The loop-level states at runtime can be estimated by EM spectrum resulting from short-time Fourier transform (STFT) of collected EM signals [61]. By comparing monitored control flow to the reference (malware-free) control flow using thestatistical *Kolmogorov-Smirnov* (K-S) test, EDDIE can detect malware injected into 10 benchmarks from MiBench [28] with at least 92 % accuracy.

While these coarse-grained monitoring techniques can detect malware from *Type-II level-2* and *Type-II level-3* adversary models, they cannot detect the lower-level malware such as sophisticated code-reuse attacks. Therefore, fine-grained CFI monitoring methods are required to identify more subtle changes in the control flow caused by potential malware.

3.2.2 Side-channel-based Fine-grained CFI Methods. In order to identify malicious instruction code that can extract a secret key or redirect the control flow to existing code with a malicious result (e.g., code-reuse attack), an instruction-level side channel monitor, also called a side-channel disassembler (SCD), can be used for fine-grained monitoring and analysis. A SCD can be designed in such a way that the instructions (code) tracked using side-channel, such as power consumption or EM radiation, can be statistically compared to the reference control flow with *instruction-level granularity* to detect any anomaly if it exists. In addition, a SCD can be used for reverse engineering of software or firmware running in embedded devices since its granularity can be tuned to individual instructions. Reverse engineering protected firmware or software is very difficult since the software is stored in the secure memory [29, 65]. In order to prevent software IP piracy, code and data are encrypted and then stored in the tamper-resistant memory. Despite of the difficulty of reverse engineering, a SCD can recognize the behavior of decrypted code and potentially detect software IP piracy. For example, a company may want to know whether its software IP is cloned by competitors or not. Since the side-channel dissembler can recognize the behavior of decrypted code, it can be utilized to detect software IP piracy. In some cases, one may need to get access to firmware in a legacy system. Although firmware is encrypted, side-channel analysis would be a useful tool to reverse-engineer the firmware and understand the functionality of the system. The only other alternative is to invasively extract the firmware (e.g., probing), which is risky and could destroy the legacy device (very few may be available).

Researchers have demonstrated various side-channel leakage-based disassembly techniques each slightly different from one another due to the target devices and applications. Vermoen *et al.* [68] introduced Java Card reverse engineering methodology that can recognize 10 different bytecodes with at least 90 % accuracy. It correlates a measured power trace during operation of the smart card at 4 MHz with an averaged power template of each bytecode and then classifies the measured power into the bytecode with the maximum correlation.

Eisenbarth *et al.* [19] proposed reverse engineering of the program executed on a PIC16F687 microcontroller at 1 MHz clock frequency. Statistical techniques such as Bayesian classifiers are used to construct classification templates from the known power consumption traces. It achieves a recognition rate of 70.1 % on 35 test instructions and 50.8 % on real code by applying a priori statistical model such as a hidden Markov model (HMM).

Msgna *et al.* [50] accomplished 100 % recognition rate on a chosen set of 39 instructions in an ATMega163-based smart card running at a clock frequency of 4 MHz. They classify the power traces by applying $k(=1)$-nearest neighbors (kNN) algorithm in combination with principal component analysis (PCA). The 100 % recognition rate, however, has not been reproduced by Strobel's experiments when the Msgna's approach was applied to different a microcontroller, PIC16F687 (less than 43 % for k=10) [63].

The SCD proposed by Strobel *at el.* [63] has a recognition rate of 96.24 % on test code and 87.69 % on real code on a PIC16F687 using localized multiple EM channels (antennas) with a decapsulated package without the priori statistical model (e.g. Markov chain). Polychotomous linear discriminant analysis (LDA) is used for the dimensionality reduction and the kNN machine learning algorithm classifies collected EM leakages with the reduced dimensionality into 33 instruction classes.

Liu *et al.* [46] proposed code execution tracking on a STC89C52 microcontroller, an implementation of Intel's 8051 architecture, at 11 MHz clock frequency using power side-channel. An HMM is applied, and in order to model good observation symbols, signal extraction with a filter to remove low SNR frequency components and PCA dimensional reduction is performed. The emission probability in the HMM is estimated by multivariate Gaussian distribution. Instructions of 9 benchmark programs are recognized with 99.94 % accuracy and less modification of original code (e.g., `NOP` instruction is replaced with an `ADD A,0x00`) can be detected.

McCann *et al.* [48] proposed an instruction-level power estimator (IPE) on ARM Cortex-M0 using linear regression to spot even subtle leakage in implementations. It is an inverse function of the SCD, i.e., if a SCD is defined as a function, $y = f(x)$, a IPE is represented as $x = f^{-1}(y)$, where x is a power trace and y is an instruction. It allows a programmer to estimate power side-channel leakage during execution of a program without real measurement. The IPE can be used in order to detect vulnerable instructions which can reveal secret information via power side-channel leakage. In addition, for malware detection, the IPE can build a fine-grained power signature of a benign application for a malware-free signature reference.

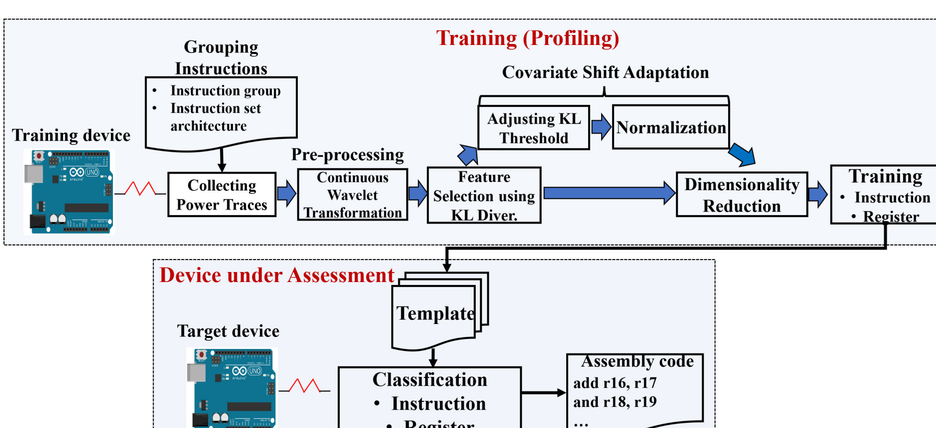

Fig. 3. Process flow for our disassembler [54]

Most of the power side-channel based fine-grained CFI assessment techniques follow the similar basic steps of data collection, preprocessing for noise reduction, and using various machine-learning techniques to identify the underlying control flow or dissimilarities if any. Existing solutions suffer from the following shortcomings: (1) the small number of instruction classes to recognize makes applicability of existing disassemblers limited. The existing methods are not able to recognize operands such as address of registers, making the reverse-engineering incomplete. (2) Most of the target devices are running at low clock frequency. Disassembling these devices are easier than those with the higher clock-frequency since the higher the frequency, the more difficult signal acquisition would be and consequently more noise to handle during analysis [22]. In a similar effort, below we present an instruction-level power-based SCD [54]. Our technique can dissect a runtime program to extract individual instructions, i.e. both the opcode and operands, efficiently with an accuracy of 99.03 %. We assume that there is no dependency between instructions. Under this assumption, some SCDs, e.g., presented by Eignebarth et al. and Liu et al., are unable to utilize the control flow information of a given program to be disassembled for a higher accuracy, and it is impossible to reverse-engineer unknown firmware in IoT devices. However, our SCD can track code execution of both known and unknown programs since we assume that every instruction can be executed independently.

Our SCD obtains all instruction templates from an original device (e.g., IoT home security system, smart thermostat, etc.) and utilizes machine learning algorithms to uniquely identify instructions executed on the device. The feature selection using Kullback-Leibler (KL) divergence and the dimensional reduction using PCA in the time-frequency domain are proposed to increase the identification accuracy. Moreover, a hierarchical classification framework is proposed to reduce the computational complexity associated with large instruction sets. In addition, covariate shifts caused by different environmental measurements and device-to-device variations are minimized by our covariate shift adaptation technique. This technique is demonstrated on an ATMega328P [35] keeping low-cost and lightweight IoT applications in mind. We would like to emphasize that this approach can be generalized to devices of similar or higher complexity. Experimental results[2] demonstrate that our disassembler can recognize test instructions including register names with a success rate no lower than 99.03 % with quadratic discriminant analysis (QDA). Fig. 3 shows the overall workflow for our SCD. We follow the below basic steps to disassemble runtime instructions:

Step 1. Power traces for instructions are collected from a training device.
Step 2. Time-varying power traces are mapped into the time-frequency domain by continuous wavelet transform.
Step 3. Feature selection and normalization are performed using Kullback-Leibler (KL) divergence with covariate shift adaptation of which details are presented in Section 5.2.
Step 4. Feature dimensionality reduction (for efficient data analysis) is performed using principal component analysis (PCA).
Step 5. Traces with reduced features are trained by ML classifiers to generate reference templates (i.e., creates decision boundaries).
Step 6. Power traces collected from a target device (i.e., device under assessment) are classified based on the templates, and then the disassembler generates the reverse-engineered assembly code running on the target device.

Our SCD has advantages as follows: It can identify operands such as the address of source registers or destination registers as well as opcode via a three-phase hierarchical process –

[2]In this experiments, 2500 power traces per class are used for the training and 500 power traces per class are collected for the testing. The accuracy is the ratio of the number of correctly classified traces to the total number of test traces.

Table 1. Comparison of existing side-channel hardware monitors.

Hardware monitor	Target devices	Clock	# of classes	Accuracy	Control flow	Dimensionality reduction	Classifier	Side-Channel	Application
WattsUpDoc [14]	x86 CPU AMD Athlon 64	664 MHz –	2 (norm. and abnorm.)	94 %/ 88.5 % 99 % / 84.9 %	coarse-grained (functional)	Mutual information	3-NN, Perceptron Random forest	Power	Malware detection
EDDIE [51]	ARM Cortex A8	–	2 (norm. and abnorm.)	92 %	coarse-grained (loop)	Spectral profiling	K-S test	EM	Malware detection
Vermoen *et al.* [68]	Smart Card	4 MHz	10 Instructions	92 %	fine-grained	–	Correlation Coefficient	Power	Reverse-engineering
Eisenbarth *et al.* [19]	PIC16F687	1 MHz	33 Instructions	70.1 %	fine-grained (HMM)	PCA, LDA	Multivariate Gaussian	Power	Reverse-engineering
Msgna *et al.* [50]	ATMega163	4 MHz	39 Instructions	100 %	fine-grained	PCA	kNN	Power	Reverse-engineering
Strobel *at el.* [63]	PIC16F687	4 MHz	33 Instructions	96.24 %	fine-grained	Polychotomous LDA	kNN	Multiple EM	Reverse-engineering
Liu *et al.* [46]	STC89C52	4 MHz	–	99.94 %	fine-grained (HMM)	PCA	Multivariate Gaussian	Power	Reverse. Malware.
McCann *et al.* [48]	ARM Cortex-M0	8 MHz	Emulating leakage	–	fine-grained	–	Linear regression	Power	Leakage evaluation
Our method [54]	ATMega328	16 MHz	112 Insts. 64 Regs.	99.03 %	fine-grained	PCA	LDA, QDA SVM, Naive	Power	Reverse. Malware.

i) identifying the instruction group[3] that a collected power trace $\boldsymbol{l}$ belongs to,
ii) identifying a particular instruction (opcode) within the identified group from the previous step, and
iii) identifying the associated operands, i.e., source and destination registers (`Rs` and `Rd`, respectively) if any.

Hence, this classification capability has potential to detect sophisticated malware and various types of other attacks (see Section 4).

Table 1 shows a comparison of existing side-channel hardware monitors in terms of the target device, the clock frequency, the number of classes, the accuracy, the granularity of control flow, the dimensionality reduction, the classifier, type of the side-channel leakage used and the target application. We see that coarse-grained techniques can sustain a relatively good amount of hardware complexity and can be implemented on low-level commodity processors. However, the fine-grained techniques that target instruction-level disassembly are mostly implemented on lightweight microcontrollers. An obvious reason behind it is that the granularity needed for instruction-level disassembly is extremely finer and the noise sensitivity affected by the complex pipeline and instruction set architecture (ISA) plays a big role in properly identifying the instructions from leakage information (Details on these challenges are presented in Section 5). However, for lightweight IoT devices, the complexity of the processing unit, i.e. microcontroller unit (MCU), is much less than that of the high-end commodity processors making the former a suitable choice for low-cost and resource-constrained applications.

4 POTENTIAL APPLICATIONS

As shown in Fig. 2, a fine-grain CFI assessment technique can be used for defense against several possible threats, as well as for adversarial attacks. In this section, we discuss some potential application cases where the user can leverage our proposed power side-channel-based instruction-level disassembler – for malware detection, firmware reverse engineering, hardware-firmware co-attestation, and detecting Meltdown and Spectre attacks, as shown in Table 2.

[3] A total of 112 instructions out of 131 instructions except for residual control, multiplication, and residual branch instructions can be recognized by the proposed disassembler. For ease of disassembly, these 112 instructions are classified into 8 groups based on corresponding operands.

4.1 Malware Detection

Due to various reasons, such as lightweight architecture, resource-constrained design, and inadequate security, IoT and embedded devices are prone to various malware infections [4, 14, 45, 73]. Here, an adversary can insert malicious code that can leak secret information, provide unauthorized control, and/or infect other IoT devices connected to the network. The inserted malware may yet be undetectable as it may remain in a stealthy mode and not hamper the original functionality of the device unless triggered. However, the activated malware changes the activity of the infected device, with respect to the legitimate behavior, via interrupts and unauthorized routine calls, and forces the embedded device to perform malicious activities. For example, to prevent the first-order side-channel attack, the original key of the Advanced Encryption Standard (AES) encryption is masked with a random number [59]. However, if the random number is maliciously turned into a fixed value, such as all binary zeros or ones, the masking method is useless and the first-order side-channel attack is, therefore, possible. The adversary can perform this attack via malware. For example, we consider the malware that infects the original AES encryption code in the device to modify the instruction `xor r16, r17` into `xor r16, r0`, where an original 8-bit subkey, a 8-bit random number, and a zero number are stored in `r16`, `r17`, and `r0`, respectively. That is, the original key is still stored in `r16` after executing the instruction and a following non-linear operation (Sbox) with the unmasking key generates significant side-channel leakage.

Note that the control flow by the example malware is the same as the reference one. Hence, it is extremely difficult to identify the modification via coarse-grained monitors. However, such sophisticated malware can be confidently detected by an accurate disassembler (e.g., like the one we discussed in Section 3) due to its capability to detect the change of the source register via the instruction disassembly. To employ this disassembly technique to detect potential malware, one needs to collect the runtime power signature from the device and check the integrity of the program running on the board. If it shows any discrepancy, in terms of opcodes or operands in the monitored assembly code, a flag is raised for potential malware infection. Therefore, the disassembly technique can detect malicious activities from the hardware at runtime, even though malware control flow has similarity with that of goodware. The summarized action steps for malware detection are shown in Table 2.

4.2 Firmware Reverse Engineering

An adversary can choose to perform firmware piracy by reverse engineering the code for potential financial benefits, unauthorized controls, and creating backdoors, as it allows him to deploy unauthentic or counterfeit devices with cloned (pirated) firmware in addition to counterfeit and malicious software and updates. In addition, an adversary can introduce subtle modifications to the original functionality by exploiting the firmware code vulnerabilities that may lead to severe damage to the system [42].

As one can see, an instruction-level SCD (like one we summarized in Section 3) can leverage power signature to reverse-engineer the firmware residing on an authentic lightweight device given that the SCD technique can potentially identify both the opcode and operands for a given device and ISA. To perform the attack, as shown in Table 2, the adversary needs to collect the power signature during runtime. If the device is designed to run some add-on software, the signature can be collected during the boot process to separate the firmware signature from the noise generated by other programs. Given the firmware complexity and a satisfactory amount of power side-channel data from the target device, the extracted instructions can be sequentially placed to generate the cloned control flow and firmware image. For reverse engineering accuracy, we assume that the target device model and instruction set architecture are known to the attacker and the adversarial model for instruction profiling from the power leakage information is sufficiently equivalent to that of the target device. Also, by employing the covariate shift adaptation technique discussed in Section 5.2, the adversary can extract distinct and non-varying features from the adversarial power signature model and focus only on selective features making the reverse engineering attack more efficient.

Further, reverse engineering of firmware or software for piracy or copyright analysis is common in industry. For example, a company may want to know whether its software IP is cloned by competitors. Even though the firmware in a competitor's devices is encrypted in the tamper-resistant memory, an instruction-level SCD can recognize the behavior of decrypted code. That is, a security engineer can use the instruction-level SCD to perform reverse engineering of software running on the competitor's device for verification of software piracy.

4.3 Hardware-Firmware Co-attestation

To ensure the integrity of an IoT network, all associated devices and firmware residing in them need to be authentic (not counterfeit), and malware-free. Further, to avoid any adversarial impersonation [15], e.g., as in the case of relay attacks, a device and its firmware can be bound together to be considered as a unified identity. The proposed fine-grained SCD method can offer a hardware-firmware co-attestation technique for ensuring the authenticity of both the device and firmware or detecting counterfeit devices and firmware. The idea behind it is that every hardware device running the same authentic firmware generates a similar but unique power signature due to manufacturing process variation, runtime conditions, and process data and workload. It should be noted that the generated in-field power signature is often too noisy to be uniquely identified by the attester using only regular template matching techniques. A well-designed SCD (similar to the one described in Section 3) can be potentially implemented to extract distinct and non-varying features. For this, one needs to identify the features that are much less susceptible to noise and possible covariate shift. If noise reduction and covariate shift adaption (discussed in Section 5.1 and 5.2) are well applied, the detection error due to environmental noise can be reduced.

To perform a hardware-firmware co-attestation, the original equipment manufacturer (OEM) is required to collect and store the power signature of the authentic device with the legitimate firmware at the beginning of the operational lifetime. During in-field operation, test signatures can be collected and verified against the initially obtained data. If any of the elements of the system (i.e. either the hardware device or the firmware) is compromised, the power signature will not remain the same and the unified attestation will no longer be valid. A further analysis of the signature to dissect the program into sequential instructions can lead to identifying whether the firmware is compromised (through unrecognized instruction/control flow) or the hardware is under attack, as summarized in Table 2. This approach can be further extended for developing a system-level mutual authentication technique [26] utilizing additional hardware-based IDs and obfuscated firmware.

4.4 Detecting Meltdown and Spectre Attacks

Two major hardware flaws in modern CPUs, called Meltdown and Spectre, were revealed in January 2018 [43]. These two bugs allow an attacker to access sensitive data stored in the memory without any log records. It impacts almost every CPU such as Intel, AMD, and ARM processors built in the past 10 years meaning that a huge number of computers, smartphones, and cloud servers currently in use are significantly vulnerable to these two security concerns. Although the software patch for Meltdown, called KAISER [25], is currently available, it still has limitations: The software patch leaves a small amount of privileged memory exposed in the user space. If the hardware exploits, namely out-of-order executions and speculative branch predictions, used by the two attacks need to be addressed, the performance may fall down by 30 %. Since these flaws are rooted in the hardware

Table 2. Potential Applications for fine-grained Instruction-level Disassembly

Application	Type	Assumption	Action summary
Malware Detection	Defense Mechanism	Malware violates legitimate CFI by adding/modifying instructions/registers (source, destination).	1. Collect power signatures for runtime instruction disassembly. 2. Do in-field CFI assessment by performing instruction-level disassembly. 3. Compare against the golden program control flow. 4. Flag suspicious instructions due to malware.
Firmware Reverse Engineering	Adversarial Threat / Defense Mechanism	Power signature model is comparable to that of the target devices and instruction sets.	1. Collect power signatures from boot process. 2. Match power templates for known hardware models and instruction sets. 3. Perform instruction-level disassembly. 4. Do consecutive instruction placement to obtain reverse engineered firmware.
Hardware-Firmware Co-attestation	Defense Mechanism	Certain non-varying features are extractable even with the presence of noise.	1. Collect power signatures from multiple target devices at *time zero* (golden data). 2. Extract and store distinct and non-varying features (solving covariate shift problem). 3. Collect in-field runtime signatures at *time t*. 4. Extract runtime features and compare with that from step 2. 5. Verify hardware-software authenticity.
Meltdown/ Spectre Detection	Defense Mechanism	Meltdown and Spectre attacks execute iterative memory access instructions which violate legitimate CFI.	1. Collect power signatures from the monitored CPU. 2. Do in-field CFI assessment by identifying iterative loop modules. 3. Determine if the identified loop is normal operations compared to the benign control flow. 4. Flag attack instructions and then terminate the application.

itself, the fundamental solution is to replace the vulnerable modules with updated (redesigned) hardware. However, it is extremely expensive, time-consuming, and practically infeasible to upgrade all vulnerable hardware. Thus, detecting and preventing Meltdown and Spectre attacks is necessary for keeping lowest possible cost and performance degradation in mind.

The fine-grained SCD framework has potential to detect both Meltdown and Spectre attacks, before the completion of attacks, given that it is adapted and optimized for commodity processors. If the attacks are detected, termination of the infected application and refreshing memory prevents an attacker from obtaining confidential information. Spectre attack [39] exploits speculatively executed indirect branch instructions which should not have been executed during a correct program execution, with following transient instructions which transmit secret data via microarchitectural covert channels (e.g., cache timing side-channel). The branch predictor directs the control flow to the transient instructions which request an access to the private data that is temporarily stored in the cache until the process redirects to normal control flow reverting the previous state before execution of the indirect branch instruction. Using cache timing attack (e.g., `Flush+Reload` attack [74]), the dump of data can be extracted. In order to detect the Spectre attack, two loops for the setup and cache timing attack should be identified by a SCD. The setup loop consists of iterative indirect branch instructions that mistrain the branch predictor so that it will later make an erroneous speculative prediction. The loop for the cache timing attack also consists the same instructions to request access to the secret data. Since these two loops are a deviation from the normal control flow, they can be detected easily by a fine-grained CFI technique such as our proposed SCD as well as by a course-grained CFI such as EDDIE [51].

The Meltdown attack [44] exploits the out-of-order execution of transient instructions stored in the reorder buffer for raising an exception caused by illegal memory access. The transient instructions to access inaccessible pages such as kernel pages are still executed in the small window time between the illegal memory access and the raising of the exception. An attack can extract the dump of inaccessible memory using cache timing attack such as `Flush+Reload` attack. Since our SCD, as well as course-grained CFIs, can identify the cache timing attack, Meltdown can be detected as well.

4.5 Miscellaneous Applications

A side-channel based instruction disassembler and its variants can offer several additional applications for IoT as well as a traditional computing domain. A key application resides in *IP/IC fingerprinting and watermarking*. Similar to hardware-firmware co-attestation technique, a SCD can utilize the power traces to extract distinct features that essentially could be used as an active fingerprint or passive watermark to the hardware device or the firmware IP under consideration [47]. A similar approach can also be explored for digital rights management (DRM) for the software/application running on an embedded device.

5 LIMITATIONS AND FUTURE RESEARCH

In this section, we discuss the open issues and challenging problems of existing side-channel monitors and address high-level approaches for future research directions.

5.1 Increased Complexity

Following the advancement trend, it is expected that the hardware used for IoT and an embedded applications will get more powerful and complex over the time, making it possible to run more sophisticated programs and with higher data collection and processing capabilities. For instance, embedded system in a smart-home collects data from many sensors and processes it continuously to make a critical decision, such as applying emergency alarms and activating water sprinkler in case of a fire, based on gathered information. However, the collected data can contain an error due to failing sensors or injected malicious code leading to a potential inaccurate decision. It requires that the system should have verification methods to decide whether the data is correct or not. If the data validation is achieved by only software, the control flow of the software generally becomes so significantly complicated that fine-grained CFI methods become infeasible. Furthermore, since an advanced electrical device requires a high performance computing unit to support the complex processing, it may contain deep pipelining, multiple cores, and a large ISA. For such cases, the side-channel templates corresponding to the control flow states at the granularity of instruction level would grow to tremendous complexity. Therefore, a fine-grained CFI method using only side-channel leakage may become infeasible to detect malicious codes.

The fine-grained CFI method with internal hardware monitors and sensors such as hardware performance counters or debug interfaces may become beneficial in such cases. For example, the fine-grained control flow graph with the granularity of instruction level can be replaced with hierarchical control flow graphs that have module-level states consisting of additional substates corresponding to instructions. As a hybrid approach, the higher-level control flow integrity can be validated using built-in hardware monitors such as performance counters and the lower-level control flow integrity in each module-level state can be validated by the fine-grained CFI monitor simultaneously to provide the accuracy in the face of increasing complexity.

5.2 Addressing Covariate Shift Problem

In real life, an embedded device undergoes different operating conditions (e.g., power supply and temperature variation) as well as runs different programs with numerous instruction combinations. The collected power traces for disassembly from a real device in the field, therefore, may be significantly different than that of an experimental device where the data is collected in a controlled environment with known programs and instructions. This can lead to a poor recognition of instructions from an in-field device using an experimentally trained classifier due to the *covariate shift* problem. This problem arises due to the difference in the probability distribution of training data (from experimental device) and testing data (from in-field device) such that $Pr_{te}(\boldsymbol{x}) \neq Pr_{tr}(\boldsymbol{x})$ even if the conditional probability of classes given training data is the same as the conditional probability of classes given testing data ($\Pr[C|\boldsymbol{x}_{tr}] = \Pr[C|\boldsymbol{x}_{te}]$) [64]. This problem also occurs in power measurement at different times or across devices and may come in a form of simple DC offset, significant magnitude and phase changes, or random noise [12].

5.2.1 Covariate Shift Adaptation. Keeping the covariate shift problem in mind, a more rigorous sample acquisition can be done to highlight distinct features. For example, in case of our SCD in Section 3.2.2, the collected dataset is extended from 2500 traces to 5700 traces to estimate non-varying feature points against the training programs with the following covariate shift adaptation; the KL threshold for within-class divergence calculation can be adjusted to a lower limit for a finer characterization. Additionally, distinct and not-varying feature points between two different classes are normalized in order to reduce the range of shifted space. Park *et al.* [54] showed that the successful recognition rate of classification between ADC and AND instructions when the covariate shift adaptation method is applied can be increased by 73.5 %.

5.2.2 Covariate Shift Caused by Different Devices. The covariate shift problem also occurs in measured powers from different devices that are the same model as the trained device. It exhibits similar challenges to that caused by different programs. Based on the template from a trained device, the measurements from other devices can be adjusted upon testing and validation. In short, covariate shift problems caused by both different programs and devices can be minimized by expanding sample space and searching not-varying feature points with normalization.

However, the requirement of increased sample space to adapt the covariate shift creates additional complexity in terms of sample acquisition, data processing, and obtaining fine-tuned signatures. Further, it requires an extensive amount of validation and adjustment from a large number of devices which subsequently makes the process costly and time-consuming. Additionally, the extraction of finer features requires high-end acquisition hardware for collecting noise-less fine-grained data. It eventually makes the current adaptation scheme somewhat infeasible for low-cost applications.

5.2.3 Aging-induced Shift. Similar to the covariate shift and noise, aging-induced shifting and SNR variation introduces additional challenges for data acquisition, model building, and verification. In addition, gate/circuit-level countermeasures against traditional power side-channel attacks [76] also suffer from aging. The predictive aging models [52, 69] can potentially be utilized to find statistical correlation, if any, for the complete system and reduce the shift in the side channel profile during post-processing.

5.3 Noise Reduction

Signal-to-noise (SNR) of side-channel leakage affects the accuracy of fine-grained CFI monitors significantly. Collected power or EM signals include noise from measurement instruments, environmental components, temperature variation, and so on. In order for the fine-grained CFI monitor to estimate op-codes and operands in an assembly code on a complex Systems on Chip (SoC) processor, each power consumption trace/profile corresponding to the op-code and operands should be extracted from a raw (original) power trace that is measured using an oscilloscope. That is, pure side-channel signals without noise should be preprocessed for high accuracy before classification or estimation.

Table 3. Challenging problems and future research directions.

Challenging Problem	Description	Research Direction
Increased Complexity	Sophisticated software has complicated CFG.	Hierarchical or Hybrid Fine-grained CFI
Covariate Shift	In-field devices produce different side-channel signatures than training devices.	Distinct and Not-varying Feature Selection
Noise Reduction	Most side-channel leakage is affected by noise. Low SNR results in low accuracy.	BSS Signal Processing
Data Acquisition	A high volume of training data is required for high accuracy or complicated processors.	High-performance Acquisition Platform
Physical Access	Physical one-spot access has limitation to simultaneously monitor multiple IoT devices.	RF Side-channel Generator

Blind source separation (BSS) such as independent component analysis [41] or singular spectrum analysis [58], i.e., the decoupling of unknown signals that have been mixed in an unknown way, can be exploited to simultaneously extract independent signals with reduced noise from the leakage. Each independent signal is used to estimate the opcode or operands. In addition, since such a signal does not depend on devices and temperature, the covariate shift problem in a non-stationary environment can be solved.

5.4 Data Acquisition and Measurement

A higher volume of data for training (or profiling) is required for high accuracy. In addition, the number of classes depending on instruction set architecture, the depth of the pipelining, and the number of CPU cores (e.g., # of classes = # of instruction $\times$ # of depth $\times$ # of cores) affects the volume of the training data. This results in an increased cost and delay as collecting side-channel leakage from state-of-art microcontrollers with measurement instruments (e.g., oscilloscope) is quite time-consuming. For a fast acquisition of side-channel leakage, the bandwidth speed between the target device and the control PC and between the measurement instrument and the control PC needs to be improved. For example, PCI-express based measurement instruments such as NI PXI platform [36] support automatic and high-performance measurement setup.

5.5 Limitation of Physical Access

To measure power or EM radiation, the target device has to be physically accessed or at least accessed within its near field. This physical one-spot access has limitation to simultaneously monitor multiple IoT devices connected to a network such as a smart home. Remote and parallel measurement methods are required in order to observe multiple IoT devices simultaneously and reduce economical cost (e.g., it is expensive for a high-performance instrument measures a side-channel leakage of a low-cost device).

For this open issue, a dedicated analog device [45] to generate an radio frequency (RF) signal including the side-channel signal as well as sending data may be a good candidate. The side-channel signal from the collectively accumulated signal/data is extracted at the monitor and based on the side-channel, the state of IoT devices can be estimated. Since the monitor can receive RF signals from multiple IoT devices remotely, it can monitor multiple IoT devices simultaneously. Table 3 shows the summary of challenging problems and future research directions.

6 CONCLUSION

With extensive concerns about the security of modern computing devices, it is imperative that hardware-based monitors be developed and deployed to thwart various cyber attacks. Our analysis shows that the existing hardware-based monitors, especially focusing on side-channel leakage-based control flow and instruction checking, require further improvement. In this regard, we illustrate a power-based side-channel instruction-level disassembler. A few simple case studies show the potential applications of the proposed disassembler. Finally, the challenging problems of existing side-channel CFI methods and high-level solutions are highlighted.

REFERENCES

[1] Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. 2005a. Control-flow Integrity. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS '05)*. ACM, New York, NY, USA, 340–353. DOI: http://dx.doi.org/10.1145/1102120.1102165

[2] Martín Abadi, Mihai Budiu, Ulfar Erlingsson, and Jay Ligatti. 2005b. Control-flow integrity. In *Proceedings of the 12th ACM conference on Computer and communications security*. ACM, 340–353.

[3] Manaar Alam, Sarani Bhattacharya, Debdeep Mukhopadhyay, and Sourangshu Bhattacharya. 2017. Performance Counters to Rescue: A Machine Learning based safeguard against Micro-architectural Side-Channel-Attacks. Cryptology ePrint Archive, Report 2017/564. (2017). https://eprint.iacr.org/2017/564.

[4] Waqas Amir. 2016. Hackers are increasingly targeting IoT Devices with Mirai DDoS Malware. https://www.hackread.com/iot-devices-with-mirai-ddos-malware/. (Oct. 2016).

[5] Nate Anderson. 2012. Confirmed: US and Israel created Stuxnet, lost control of it. https://arstechnica.com/tech-policy/2012/06/confirmed-us-israel-created-stuxnet-lost-control-of-it/. (June 2012).

[6] Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. 2011. An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011*. 105–114. DOI: http://dx.doi.org/10.1109/FDTC.2011.9

[7] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. 2011. Duplexing the sponge: single-pass authenticated encryption and other applications. Cryptology ePrint Archive, Report 2011/499. (2011). https://eprint.iacr.org/2011/499.

[8] Tyler Bletsch. 2011. *Code-reuse Attacks: New Frontiers and Defenses*. Ph.D. Dissertation. Advisor(s) Freeh, Vince and Jiang, Xuxian. AAI3463747.

[9] Eric Brier, Christophe Clavier, and Francis Olivier. 2004. *Correlation Power Analysis with a Leakage Model*. Springer Berlin Heidelberg, Berlin, Heidelberg, 16–29. DOI: http://dx.doi.org/10.1007/978-3-540-28632-5_2

[10] Nathan Burow, Scott A. Carr, Joseph Nash, Per Larsen, Michael Franz, Stefan Brunthaler, and Mathias Payer. 2017. Control-Flow Integrity: Precision, Security, and Performance. *ACM Comput. Surv.* 50, 1, Article 16 (April 2017), 33 pages. DOI: http://dx.doi.org/10.1145/3054924

[11] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. 2003. *Template Attacks*. Springer Berlin Heidelberg, Berlin, Heidelberg, 13–28. DOI: http://dx.doi.org/10.1007/3-540-36400-5_3

[12] Omar Choudary and Markus G. Kuhn. 2014. *Template Attacks on Different Devices*. Springer International Publishing, Cham, 179–198.

[13] Nick Christoulakis, George Christou, Elias Athanasopoulos, and Sotiris Ioannidis. 2016. HCFI: Hardware-enforced Control-Flow Integrity. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy (CODASPY '16)*. ACM, New York, NY, USA, 38–49. DOI: http://dx.doi.org/10.1145/2857705.2857722

[14] Shane S. Clark, Benjamin Ransford, Amir Rahmati, Shane Guineau, Jacob Sorber, Wenyuan Xu, and Kevin Fu. 2013. WattsUpDoc: Power Side Channels to Nonintrusively Discover Untargeted Malware on Embedded Medical Devices. In *Presented as part of the 2013 USENIX Workshop on Health Information Technologies*. USENIX, Washington, D.C. https://www.usenix.org/conference/healthtech13/workshop-program/presentation/Clark

[15] Boris Danev, Heinrich Luecken, Srdjan Capkun, and Karim El Defrawy. 2010. Attacks on physical-layer identification. In *Proceedings of the third ACM conference on Wireless network security*. ACM, 89–98.

[16] DARPA. 2015. Leveraging the Analog Domain for Security (LADS). (2015). http://www.darpa.mil/program/leveraging-the-analog-domain-for-security

[17] Lucas Davi, Patrick Koeberl, and Ahmad-Reza Sadeghi. 2014. Hardware-Assisted Fine-Grained Control-Flow Integrity: Towards Efficient Protection of Embedded Systems Against Software Exploitation. In *Proceedings of the 51st Annual Design Automation Conference (DAC '14)*. ACM, New York, NY, USA, Article 133, 6 pages. DOI: http://dx.doi.org/10.1145/2593069.2596656

[18] Ruan de Clercq and Ingrid Verbauwhede. 2017. A survey of Hardware-based Control Flow Integrity (CFI). *CoRR*

abs/1706.07257 (2017). http://arxiv.org/abs/1706.07257
[19] Thomas Eisenbarth, Christof Paar, and BjÃ rn Weghenkel. 2010. Building a Side Channel Based Disassembler. In *Transactions on Computational Science X*, Marina L. Gavrilova, C.J. Kenneth Tan, and Edward David Moreno (Eds.). Lecture Notes in Computer Science, Vol. 6340. Springer Berlin Heidelberg, 78–99.
[20] The Nexus 5001 Forum. 2003. Standard for a Global Embedded Processor Debug Interface Version 2.0. *IEEE-Industry Standards and Technology Organization (IEEE-ISTO)* (2003).
[21] Aurélien Francillon and Claude Castelluccia. 2008. Code injection attacks on harvard-architecture devices. In *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 15–26.
[22] Jake Longo Galea, Elke De Mulder, Daniel Page, and Michael Tunstall. 2015. SoC it to EM: electromagnetic side-channel attacks on a complex system-on-chip. *IACR Cryptology ePrint Archive* 2015 (2015), 561.
[23] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. 2008. *Mutual Information Analysis*. Springer Berlin Heidelberg, Berlin, Heidelberg, 426–442. DOI: http://dx.doi.org/10.1007/978-3-540-85053-3_27
[24] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. 2011. A Testing Methodology for Side-Channel Resistance Validation. In *NIST non-invasive attack testing workshop*. 158–172.
[25] Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, Clémentine Maurice, and Stefan Mangard. 2017. *KASLR is Dead: Long Live KASLR*. Springer International Publishing, Cham, 161–176. DOI: http://dx.doi.org/10.1007/978-3-319-62105-0_11
[26] Ujjwal Guin, Swarup Bhunia, Domenic Forte, and Mark M. Tehranipoor. 2017. SMA: A System-Level Mutual Authentication for Protecting Electronic Hardware and Firmware. *IEEE Transactions on Dependable and Secure Computing* 14, 3 (2017), 265–278. DOI: http://dx.doi.org/doi.ieeecomputersociety.org/10.1109/TDSC.2016.2615609
[27] Z. Guo, R. Bhakta, and I. G. Harris. 2014. Control-flow checking for intrusion detection via a real-time debug interface. In *2014 International Conference on Smart Computing Workshops*. 87–92. DOI: http://dx.doi.org/10.1109/SMARTCOMP-W.2014.7046672
[28] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. 2001. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop (WWC '01)*. IEEE Computer Society, Washington, DC, USA, 3–14. DOI: http://dx.doi.org/10.1109/WWC.2001.15
[29] Michael Henson and Stephen Taylor. 2014. Memory Encryption: A Survey of Existing Techniques. *ACM Comput. Surv.* 46, 4, Article 53 (March 2014), 26 pages. DOI: http://dx.doi.org/10.1145/2566673
[30] H. Holm. 2014. Signature Based Intrusion Detection for Zero-Day Attacks: (Not) A Closed Chapter?. In *2014 47th Hawaii International Conference on System Sciences*. 4895–4904. DOI: http://dx.doi.org/10.1109/HICSS.2014.600
[31] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. 2011. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering* 1, 4 (27 Oct 2011), 293. DOI: http://dx.doi.org/10.1007/s13389-011-0023-x
[32] Michael Hutter and Jörn-Marc Schmidt. 2013. The Temperature Side Channel and Heating Fault Attacks. In *CARDIS (Lecture Notes in Computer Science)*, Vol. 8419. Springer, 219–235.
[33] Michael Hutter and Jörn-Marc Schmidt. 2014. The Temperature Side Channel and Heating Fault Attacks. *IACR Cryptology ePrint Archive* 2014 (2014), 190. http://eprint.iacr.org/2014/190
[34] Robert Hyatt. 1999. Crafty (186) SPEC CPU2000 Benchmark. (1999). https://www.spec.org/cpu2000/CINT2000/186.crafty/docs/186.crafty.html Accessed: 2017-12-14.
[35] Atmel Inc. 2016. AVR Instruction set manual. (2016). http://www.atmel.com/images/Atmel-0856-AVR-Instruction-Set-Manual.pdf
[36] National Instruments. 2018. PXI Platform. (2018). http://www.ni.com/pxi/ Accessed: 2018-3-3.
[37] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ACM SIGARCH Computer Architecture News*, Vol. 42. IEEE Press, 361–372.
[38] Vladimir Kiriansky, Derek Bruening, and Saman P. Amarasinghe. 2002. Secure Execution via Program Shepherding. In *Proceedings of the 11th USENIX Security Symposium*. USENIX Association, Berkeley, CA, USA, 191–206. http://dl.acm.org/citation.cfm?id=647253.720293
[39] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre Attacks: Exploiting Speculative Execution. *ArXiv e-prints* (Jan. 2018).
[40] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. Springer-Verlag, 388–397.
[41] E. Kofidis. 2016. Blind Source Separation: Fundamentals and Recent Advances (A Tutorial Overview Presented at SBrT-2001). *ArXiv e-prints* (March 2016).
[42] Charalambos Konstantinou and Michail Maniatakos. 2015. Impact of firmware modification attacks on power systems field devices. In *Smart Grid Communications (SmartGridComm), 2015 IEEE International Conference on*. IEEE, 283–288.

[43] Selena Larson. 2018. Major chip flaws affect billions of devices. http://money.cnn.com/2018/01/03/technology/computer-chip-flaw-security/index.html?iid=hp-toplead-dom. (Jan. 2018).

[44] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. *ArXiv e-prints* (Jan. 2018).

[45] Y. Liu, Y. Jin, and Y. Makris. 2013. Hardware Trojans in wireless cryptographic ICs: Silicon demonstration detection method evaluation. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 399–404. DOI: http://dx.doi.org/10.1109/ICCAD.2013.6691149

[46] Yannan Liu, Lingxiao Wei, Zhe Zhou, Kehuan Zhang, Wenyuan Xu, and Qiang Xu. 2016. On Code Execution Tracking via Power Side-Channel. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 1019–1031. DOI: http://dx.doi.org/10.1145/2976749.2978299

[47] Cédric Marchand, Lilian Bossuet, and Edward Jung. 2014. IP watermark verification based on power consumption analysis. In *System-on-Chip Conference (SOCC), 2014 27th IEEE International*. IEEE, 330–335.

[48] David McCann, Carolyn Whitnall, and Elisabeth Oswald. 2016. ELMO: Emulating Leaks for the ARM Cortex-M0 without Access to a Side Channel Lab. Cryptology ePrint Archive, Report 2016/517. (2016). http://eprint.iacr.org/2016/517.

[49] Peter Middleton, Peter Kjeldsen, and Jim Tully. 2013. Forecast: The internet of things, worldwide, 2013. *Gartner Research* (2013).

[50] Mehari Msgna, Konstantinos Markantonakis, and Keith Mayes. 2014. *Precise Instruction-Level Side Channel Profiling of Embedded Processors*. Lecture Notes in Computer Science, Vol. 8434. Springer, 129–143. DOI: http://dx.doi.org/10.1007/978-3-319-06320-1_11

[51] Alireza Nazari, Nader Sehatbakhsh, Monjur Alam, Alenka Zajic, and Milos Prvulovic. 2017. EDDIE: EM-Based Detection of Deviations in Program Execution. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. ACM, New York, NY, USA, 333–346. DOI: http://dx.doi.org/10.1145/3079856.3080223

[52] Fabian Oboril and Mehdi Baradaran Tahoori. 2012. ExtraTime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level. In *IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2012, Boston, MA, USA, June 25-28, 2012*. 1–12. DOI: http://dx.doi.org/10.1109/DSN.2012.6263957

[53] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache Attacks and Countermeasures: The Case of AES. In *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology (CT-RSA'06)*. Springer-Verlag, Berlin, Heidelberg, 1–20. DOI: http://dx.doi.org/10.1007/11605805_1

[54] Jungmin Park, Xiaolin Xu, Yier Jin, Domenic Forte, and Mark Tehranipoor. 2018. Power-based Side-channel Instruction-level Disassembler. In *Proceedings of the 55th Annual Design Automation Conference (DAC '18)*. ACM, New York, NY, USA.

[55] Siman Parker. 2017. Understanding The Physical Damage of Cyber Attacks. https://www.infosecurity-magazine.com/opinions/physical-damage-cyber-attacks/. (Oct. 2017).

[56] Nisarg Patel, Avesta Sasan, and Houman Homayoun. 2017. Analyzing Hardware Based Malware Detectors. In *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 25.

[57] S. Picek, A. Heuser, A. Jovic, S. A. Ludwig, S. Guilley, D. Jakobovic, and N. Mentens. 2017. Side-channel analysis and machine learning: A practical perspective. In *2017 International Joint Conference on Neural Networks (IJCNN)*. 4095–4102. DOI: http://dx.doi.org/10.1109/IJCNN.2017.7966373

[58] Santos Merino Del Pozo and François-Xavier Standaert. 2016. Blind Source Separation from Single Measurements using Singular Spectrum Analysis. Cryptology ePrint Archive, Report 2016/314. (2016). https://eprint.iacr.org/2016/314.

[59] Emmanuel Prouff and Matthieu Rivain. 2007. *A Generic Method for Secure SBox Implementation*. Springer Berlin Heidelberg, Berlin, Heidelberg, 227–244. DOI: http://dx.doi.org/10.1007/978-3-540-77535-5_17

[60] Tobias Schneider and Amir Moradi. 2015. *Leakage Assessment Methodology*. Springer Berlin Heidelberg, Berlin, Heidelberg, 495–513. DOI: http://dx.doi.org/10.1007/978-3-662-48324-4_25

[61] N. Sehatbakhsh, A. Nazari, A. Zajic, and M. Prvulovic. 2016. Spectral profiling: Observer-effect-free profiling by monitoring EM emanations. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–11. DOI: http://dx.doi.org/10.1109/MICRO.2016.7783762

[62] Raphael Spreitzer, Veelasha Moonsamy, Thomas Korak, and Stefan Mangard. 2016. SoK: Systematic Classification of Side-Channel Attacks on Mobile Devices. *CoRR* abs/1611.03748 (2016).

[63] Daehyun Strobel, Florian Bache, David Oswald, Falk Schellenberg, and Christof Paar. 2015. Scandalee: a side-channel-based disassembler using local electromagnetic emanations. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015*. 139–144. http://dl.acm.org/citation.cfm?id=2755784

[64] Masashi Sugiyama and Motoaki Kawanabe. 2012. *Machine Learning in Non-Stationary Environments: Introduction to Covariate Shift Adaptation*. The MIT Press.

[65] G. Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. 2003. AEGIS: Architecture for Tamper-evident and Tamper-resistant Processing. In *Proceedings of the 17th Annual International Conference on*

Supercomputing (ICS '03). ACM, New York, NY, USA, 160–171. DOI: http://dx.doi.org/10.1145/782814.782838

[66] Adrian Tang, Simha Sethumadhavan, and Salvatore J. Stolfo. 2014. *Unsupervised Anomaly-Based Malware Detection Using Hardware Features*. Springer International Publishing, Cham, 109–129. DOI: http://dx.doi.org/10.1007/978-3-319-11379-1_6

[67] Adrian Tang, Simha Sethumadhavan, and Salvatore J. Stolfo. 2017. CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management. In *USENIX Security Symposium*.

[68] Dennis Vermoen, Marc Witteman, and Georgi N. Gaydadjiev. 2007. Reverse Engineering Java Card Applets Using Power Analysis. In *Proceedings of the 1st IFIP TC6 /WG8.8 /WG11.2 International Conference on Information Security Theory and Practices: Smart Cards, Mobile and Ubiquitous Computing Systems (WISTP'07)*. Springer-Verlag, Berlin, Heidelberg, 138–149. http://dl.acm.org/citation.cfm?id=1763190.1763207

[69] W. Wang, V. Reddy, Bo Yang, V. Balakrishnan, S. Krishnan, and Yu Cao. 2008. Statistical prediction of circuit aging under process variations. In *2008 IEEE Custom Integrated Circuits Conference*. 13–16. DOI: http://dx.doi.org/10.1109/CICC.2008.4672007

[70] X. Wang, C. Konstantinou, M. Maniatakos, R. Karri, S. Lee, P. Robison, P. Stergiou, and S. Kim. 2016. Malicious Firmware Detection with Hardware Performance Counters. *IEEE Transactions on Multi-Scale Computing Systems* 2, 3 (July 2016), 160–173. DOI: http://dx.doi.org/10.1109/TMSCS.2016.2569467

[71] Mario Werner, Thomas Unterluggauer, David Schaffenrath, and Stefan Mangard. 2018. Sponge-Based Control-Flow Protection for IoT Devices. *CoRR* abs/1802.06691 (2018). http://arxiv.org/abs/1802.06691

[72] Jacob Wurm, Yier Jin, Yang Liu, Shiyan Hu, Kenneth Heffner, Fahim Rahman, and Mark Tehranipoor. 2016. Introduction to Cyber-Physical System Security: A Cross-Layer Perspective. *IEEE Transactions on Multi-Scale Computing Systems* (2016).

[73] Zhixing Xu, Sayak Ray, Pramod Subramanyan, and Sharad Malik. 2017. Malware Detection Using Machine Learning Based Analysis of Virtual Memory Access Patterns. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE '17)*. European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 169–174. http://dl.acm.org/citation.cfm?id=3130379.3130417

[74] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-channel Attack. In *Proceedings of the 23rd USENIX Conference on Security Symposium (SEC'14)*. USENIX Association, Berkeley, CA, USA, 719–732. http://dl.acm.org/citation.cfm?id=2671225.2671271

[75] Man-Ki Yoon, Sibin Mohan, Jaesik Choi, and Lui Sha. 2015. Memory Heat Map: Anomaly Detection in Real-time Embedded Systems Using Memory Behavior. In *Proceedings of the 52Nd Annual Design Automation Conference (DAC '15)*. ACM, New York, NY, USA, Article 35, 6 pages. DOI: http://dx.doi.org/10.1145/2744769.2744869

[76] Lu Zhang, Luis Vega Gutierrez, and Michael Bedford Taylor. 2016. Power Side Channels in Security ICs: Hardware Countermeasures. *CoRR* abs/1605.00681 (2016). http://arxiv.org/abs/1605.00681

[77] YongBin Zhou and DengGuo Feng. 2005. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing. (2005). http://eprint.iacr.org/2005/388 zyb@is.iscas.ac.cn 13083 received 27 Oct 2005.