




Gray-Box Software Integrity Checking via Side-Channels

Hong Liu and Eugene Y. Vasserman^(✉) 

Department of Computer Science, Kansas State University,
1701D Platt Street, Manhattan, KS, USA
{hongl, eyv}@ksu.edu

Abstract. Enforcing software integrity is a challenge in embedded systems which cannot employ modern protection mechanisms. In this paper, we explore feasibility of software integrity checking from measuring passive electromagnetic emissions of FPGA-implemented SoCs. We show that clock-cycle-accurate side-channel models can be built by utilizing gray-box analysis and regression techniques. The generality and effectiveness of our methods are shown by three different SoCs, profiled and tested on different chips of the same model. Our technique is non-invasive, and does not interrupt normal execution or change hardware/software configuration of the target device, making it particularly attractive for already-deployed systems.

Keywords: Security · Embedded systems · Side-channel analysis
Software attestation · Soft-core processors · FPGA

1 Introduction

Enforcing software integrity is a fundamental problem in system security: a device runs some software, and a verifier wants to know whether the device runs an unmodified version of the software, or a different piece of code, or original code but in an unintended execution state [13, 17, 43]. Enforcing software integrity for embedded systems, especially fielded/legacy ones, is extremely difficult. Software-based methods such as hypervisors [41], separation kernels [19], and control flow integrity checking [40, 43] detect/prevent tampering by utilizing hardware security features that provide some form of separation such as operation modes and memory protection. Remote attestation [10, 23, 29], secure boot [16], and watchdog coprocessors [42] rely on trusted hardware and memory access controls to execute attestation code, e.g., to verify memory content and examine signatures appearing on buses. However, many embedded systems do not have such sophisticated capabilities due to hardware cost, high power consumption, and/or the difficulty in updating fielded components. Further, an external verification mechanism may be required no matter which protection method is used, as some security assumptions may weaken with time, and verification of design-time assumptions is needed.

For systems composed of discrete components, sniffing bus traffic between constituent components may suffice to verify that the system acts as expected. For Systems-on-Chips (SoCs), however, internal activities cannot be observed directly, so we utilize “side-channel” emissions to infer them. The idea is to discover the relationships between the internal states of a target device and side-channel information so that when given a new side-channel measurement, it is possible to determine whether the internal state is an expected one or not. Unexpected state may be a sign of incorrect execution or malware. Researchers have tried using external power measurements to detect abnormal behavior at the level of functions or code segments [18, 38, 45], but attackers can write compact malware (as small as one instruction in size), code-reuse malware, as well as malware that has minimal impact on system-wide side-channel measurements [49]. In such situations, software attestation [9, 36, 54, 62] can be used to detect modification of software down to single instructions. These methods utilize the timing side-channel and do not rely on specialized hardware or sophisticated processors, but do require interruption of normal execution of the target device. In addition, systems must explicitly support software attestation, making retrofitting difficult for legacy and deployed systems.

In this work we explore the possibility of using passive and non-invasive side-channel measurements for software integrity checking on SoCs. Our approach infers internal runtime state of an embedded system at a granularity sufficient to detect compact and side-channel-aware malware, without modifying the target device. This is extremely beneficial since the verifier is external to the device under test and is the only possible effective verifier when all security mechanisms (if any) internal to the device have failed.

Instead of analyzing numerous hardware platforms one by one, we choose field-programmable gate arrays (FPGAs) as the target SoC device. It has unique features compared to other platforms [20], but also poses unique challenges, namely the inability to perform full white-box analysis, as the detailed design and parameters of the base array and the configuration circuits are unknown to developers. The use of IP cores further obscures the electrical characteristics of a device. We therefore work with only partial knowledge, and so term our approach “gray-box” side-channel analysis.

In this work, we make the following contributions:

- We demonstrate the feasibility of using passive electromagnetic (EM) emissions of FPGA-based SoCs for software integrity checking. Our method is scalable, low-cost, and easily applicable to deployed systems.
- We show how to build cycle-accurate models of passive EM radiation of FPGA-based SoCs without access to detailed design specifications and how to efficiently and effectively use regression for EM profiling, even for systems with large instruction sets and variable instruction cycles.
- We experimentally validate the generality of our approach on three different FPGA-based SoCs – two based on a soft processor IP core (namely NIOS II), and one on the OpenMSP430 open processor core, and further show that

profiling is robust to manufacturing variations by testing on a different FPGA chip of the same model with the chip for profiling.

- We provide bounds on the (very low) probability that an EM-aware adversary can successfully modify code without being detected.

2 Related Work

Passive side-channel emissions of various embedded systems have long been studied to optimize power usage and perform EMI/EMC analysis [58]. Side-channel models can be built at different levels given different degrees of knowledge on system configurations. At the lowest level, power consumption of FPGAs has been modeled at the transistor level in order to build power-efficient FPGA architectures and power-aware CAD tools [8, 25, 31, 47, 50]. Dynamic power consumption is in general modeled as the aggregation of power consumed by each node inside an FPGA whose load and parasitic capacitances are charged and discharged at signal transitions, as well as short-circuit power that occurs in CMOS inverters [8, 25, 50]. For FPGA-implemented SoCs in which a complex processor is involved, low-level analysis becomes impractical, especially given reliance on detailed design information, so researchers built side-channel models from empirical measurements of real boards. Senn et al. [53] measured system-level power consumption of the NIOS II core and Zipf et al. [63] performed a hybrid functional- and instruction-level power analysis of LEON2, another soft-core processor. However, these estimation models profiled the average side-channel emissions of embedded systems rather than trying to infer system state (which program is running and its runtime state) from side-channel measurements.

In the cryptographic hardware domain, passive side-channel emissions of FPGA-implemented cryptographic routines are used to extract secret materials (e.g., keys) [7, 33, 34, 60]. Such analyses concentrate on a few leakage points of the keys, with cryptographic algorithms often considered to be public (although the implementation details may be unknown), or irrelevant. Secret keys may be exposed regardless of cryptographic algorithms used [11], so the work on cryptographic hardware does not present a comprehensive picture of dynamic side-channel emissions of an entire embedded system.

Work on side-channel analysis of general programs used passive system-wide power measurements to detect anomalous behaviors and/or malware [18, 27, 28, 38, 45, 61]. These methods, however, assume malware (code) to be sufficiently long and not written to conceal its side-channel profiles. To use side-channels for rigorous integrity checking, we must consider compact and side-channel-aware malware. Software attestation [9, 36, 54, 62] utilizes the timing side-channel and is capable of detecting malware at such precision. However, the device must support such attestation, and carrying out the process requires interruption of the device execution, a particular drawback for legacy and actively-used systems.

Researchers have tried to build side-channel models of the instruction set for certain smart cards and microcontrollers [22, 26, 27, 46, 51, 55, 56, 59], and to build side-channel-based disassemblers by recognizing instruction operations from passive measurements. However, this focused on recognizing instruction operations

(e.g., `MOVLW`) instead of the entire instructions including operands (e.g., `MOVLW 0xAA`) and content of operands. In [37], researchers found that passive power measurement of some microcontroller was dominated by a small number of data-dependent relationships, implying that recognizing instruction operations solely from power consumption is unlikely to be reliable.

For integrity checking of the FPGA platforms specifically, previous research on reading SRAM [57] and detection of FPGA trojans [30] shows the fundamental possibility of verifying FPGA configuration logic. The methods however require invasive measurements and specialized equipment, and do not scale well for complex systems. It is not known how to efficiently verify the software of an FPGA-based embedded system in practice.

3 Problem Definition

We define the general software integrity checking problem as follows, shown in Fig. 1. There are two parties: the prover P (a device running the target application software S), and the verifier V (who would like to determine whether P runs S or a modified version S'). V is a trusted entity who knows the initial hardware and software configuration of P . P and V communicate over an explicit channel C and/or a side-channel E . V bases its judgment on evidence that P provides directly through C (e.g., using signatures) or indirectly over E (e.g., by timing or EM radiation).

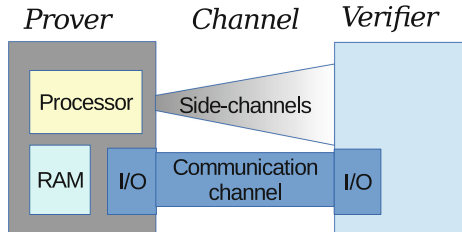


Fig. 1. General software integrity checking problem with explicit communication channels and side-channels

This model can be instantiated in different ways. In a microcontroller-based embedded system, for example, S is naturally the software of the microcontroller, and P is the microcontroller chip and the printed circuit board (PCB). In an FPGA-based embedded system, however, both the hardware and the software are programmable: FPGA configuration logic describes both the embedded system hardware (processor, memory, I/O, etc.) as well as the application software running on the system. Since reconfiguring an FPGA causes considerable difference in observations (such as loss of main clock in EM emissions), it is straightforward to discover tampering with FPGA configuration logic. We focus on detecting tampering with the application software which is modifiable

on-the-fly in memory. S is therefore defined as the application software, and P incorporates the PCB, the FPGA chip, and the FPGA configuration logic describing the hardware of the system.

3.1 Threat Model

Attacker. We assume that the attacker is unable to modify or inject faults into the PCB and the FPGA chip of the target embedded system. This is enforceable in practice using physical tamper-resistance or tamper-proof techniques [35]. Moreover, the attacker is not an insider of the FPGA or IP core manufacturers, and cannot tamper the FPGA IC design, IP cores, or the CAD tools, on which we rely to establish the ground truth. We further assume that the attacker is unable to modify the FPGA hardware configuration, i.e., cannot reconfigure the hardware of the FPGA. This can be achieved by authenticating the configuration bit-stream [20], or by removing configuration peripherals before deployment, or by observing the EM emissions as mentioned in last paragraph. However, we assume that the attacker *is* able to modify the application software S , e.g., modifying the RAM of the device through buffer overflows, data-based attacks, etc. We also assume that the attacker is side-channel-aware – actively attempts to evade detection via side-channel emissions by crafting the modified software S' in a fashion that minimizes side-channel deviations from S . Nonetheless, the attacker cannot *invasively* profile the side-channels of the target device.

Verifier. We assume a verifier of very limited capability for applicability of our approach. We assume that the verifier knows the configuration of a target device and is able to profile the side-channel characteristics only on a *different* device of the same model with the target device. The verifier can only perform non-invasive measurements on the target device, which is important for this methodology to be easily applied to deployed devices.

We emphasize that the verifier is completely external to the target device, and *cannot modify the device hardware or software* to change the nature or magnitude of side-channel emissions, so that the verifier has the advantage of invisibility to attackers. The verifier can only passively measure the target device with measurement equipment incurring minimal impact on EMC. For example, the verifier may remove the shielding enclosure for measurements, but may not remove the noise decoupling circuits.

4 Experimental Setup

For a representative legacy and deployed system, we choose a general-purpose development board for the Altera Cyclone III FPGA EP3C5E144C8 as the target device. The FPGA chip is designed for low-cost, small-scale applications. We choose the EM side-channel, which is much more convenient to measure than power consumption. Preliminary test on the chip shows that it is not EM-shielded, which eases our experiment. We implement one SoC on the FPGA at

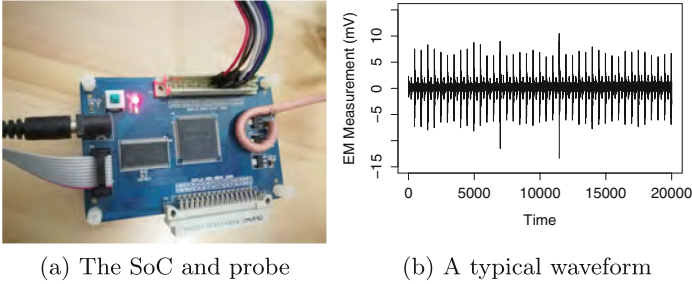


Fig. 2. Experimental setup and an example of a recorded result

a time in the way that the only observable I/O is a parallel peripheral I/O (the memory chip on the development board is not used). Two different chips (i.e., boards) of the same model are used, one for profiling and another for testing.

We find that several positions on the board emit strong EM signals of similar waveform. We have tried both far-field and near-field measurements of EM radiation, and obtained the best result from near-field measurements with a shielded loop probe similar to the EMC probe in [48]. The probe measures the global radiation of the FPGA chip. The resulting setup, with the probe position near one of the power regulators, is shown in Fig. 2a.

The output is amplified by a 20 dB amplifier with bandwidth from 1 kHz to 1 GHz, and then is sampled by a PicoScope 5244B oscilloscope, which has a 200 MHz bandwidth and a maximally 500 MS/s sample rate for each channel. We use the 20 MHz integrated hardware filter of the oscilloscope to avoid aliasing. The processor core clock frequency therefore should be set far lower – we set it to 1 MHz. Our results should be repeatable at higher frequencies using more costly oscilloscopes supporting higher bandwidths. The position and orientation of the probe is then adjusted to gain signals of the maximal signal-to-noise ratio (SNR). Probe location is re-adjusted for each SoC. The resulting SNR of the EM traces, computed by the ratio of the variance of the signal and noise, is around 15 dB. A typical single-captured waveform is shown in Fig. 2b.

We have intentionally used low-cost signal acquisition and analysis equipment in order to show that verification of low-end legacy systems can be accomplished with only modest resources. The most costly component in our experiment setup is the off-the-shelf USB oscilloscope, at about \$2,000; putting the total system cost at under \$2,100. Combining all components into a single “software integrity measurement device” and manufacturing at scale is likely to further reduce costs.

5 The Test Code and SoC Test Targets

We evaluate our approach on three SoCs, implemented in turn on our FPGA test-bed: a NIOS II-based system capable of running an operating system; a simpler NIOS II-based system with a more constrained resource configuration; and an OpenMSP430-based system that is also operating system-capable.

5.1 System A: NIOS II-Based SoC

Our first experiment is on a NIOS II-based SoC. NIOS II is a general-purpose 32-bit RISC soft processor core from Altera [4]. We chose the NIOS II/e Quartus II 13.1 web edition (the latest edition for the target FPGA). NIOS II/e is designed for simple control logic applications and/or inexpensive systems. It supports over a hundred instruction operations, executed in a variable number of clock cycles, ranging from six to 38. (Our experiments show it is actually seven to 39, contrary to specifications.) The HDL source is not available, and the processor offers only limited configurability. It does not support different operating modes or memory protection, so modern security mechanisms that rely on processor-enforced separation cannot be used.

The NIOS II-based system is composed of the core, 40 kB M9K RAM, a timer, and a 16-bit parallel I/O connected using the Avalon bus. The system can run the FreeRTOS operating system [3] and several application tasks. The entire FPGA-implemented SoC consists of the NIOS II-based system, a small control unit that supplies clock and reset signals to the core, and a phase-locked loop (PLL). Programs are loaded and executed directly in RAM, forming a complete SoC, i.e., with no bus interfaces outside the chip except parallel I/O. We remove the JTAG interface of the processor, as it is unlikely to be present in a deployed device. The 1 MHz clock is obtained by using a PLL core connected to an external 25 MHz clock source. We do not make any effort to enhance the side-channel emissions when generating the system, so the experiment measures the typical EM radiation of a NIOS II-based system.

5.2 System B: Resource-Constrained NIOS II-Based SoC

The second SoC is also NIOS II-based, but simpler, to represent a “bare-bones” system that does not have enough resources to run an operating system. It has only a 16 kB M9K RAM for program and data memory, and an 8-bit parallel I/O. No timers are present. Otherwise is identical to system A.

5.3 OpenMSP430-Based SoC (System C)

The third system is based on OpenMSP430, a 16-bit open-source MSP430 family-compatible processor [5]. It supports 27 core instructions and seven addressing modes. Any valid combination of source and destination addressing modes is possible in an instruction, unlike NIOS II, which uses explicit load and store operations. Instructions of OpenMSP430 can be byte or word operations, whereas NIOS II supports only 32-bit word operations for instructions other than load and store. The number of clock cycles required for an instruction is variable (from one to six), depending both on the instruction type and addressing mode.

The SoC consists of the processor, 32 kB M9K RAM for program memory and 4 kB for data memory, a timer, and a 16-bit parallel I/O. Otherwise configuration is the same for all three systems. Programs are compiled using MSP430-GCC, then binaries are converted to an FPGA RAM initialization file, and loaded and executed directly in RAM, forming a complete SoC.

5.4 Test Code

Ideally, we should exercise all the possible internal states of the target SoCs to build side-channel models (i.e., template analysis). However, this is impractical since our preliminary tests show that the EM radiation depends on instruction operations, operands, and the content of the registers, memories, etc. (see Sect. 8). We can only build side-channel models from a very limited number of programs and data configurations, compared with the entire state space of the system. The validity of the resulting model is tested both by the reasonableness of its form and by the predictive power for side-channel emissions of new programs and data. Our test code is an integration of the FreeRTOS operating system port for each core (except system B) and re-implementation (to fit into available memory) of a part of the CoreMark benchmark suite [1]: integer matrix multiplication, floating-point multiplication, greatest common divisor, quick-sort of vector data, list find, list quick-sort, string hash, and a finite state machine, as well as random assembly code we generate for each core that avoids memory access and bypasses the native compilers: one composed of logic/arithmetic instructions, and one composed of only five types of logic/arithmetic instructions. The program binaries execute in a similar number of clock cycles. We do not model the EM radiation of I/Os, or code that change system-level behaviors, e.g., the timer intervals.

6 Modeling Side-Channels

Our preliminary tests show that EM emissions of different instruction operations largely overlap. Profiling target EM emissions by using general classifiers is therefore difficult [22, 26, 51, 56]. Furthermore, knowing only the instruction operations does not guarantee integrity since an attacker may write malware by varying only the operands and content of registers/memory, while keeping the operations the same. Previous research [37] has shown that the power consumption of a microcontroller can be accurately described as a linear model of a few internal data-dependent activities, e.g., Hamming weight (HW) of instructions and Hamming distance (HD) between operand and result – the contribution of different operations is negligible. We perform a similar experiment using EM measurements (same probe), and find that previous linear model of power consumption still holds for EM radiation. The only difference is the values of regression coefficients and the omission of a near-DC component, which is linear to the HW of instructions in the power model. This strongly suggests that we may be able to build similar regression models for the FPGA-based SoCs.

We assume the EM sample Y_t at time t can be modeled as a function of internal states $\mathbf{x}_t = (x_{1t}, \dots, x_{pt})$ at t :

$$Y_t = f(\mathbf{x}_t) + N_t$$

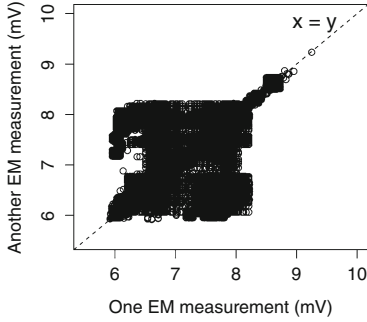
where N_t encloses remaining components in the EM radiation including noise and time-dependent components, Y_t and N_t are necessarily random variables. x_{it} ($i = 1, \dots, p$) are called the predictor variables and Y_t the response variable.

Since both processors execute instructions in a variable number of clock cycles, depending on operands and bus traffic, we build side-channel models for each clock cycle rather than for each instruction cycle. The sample rate of the oscilloscope is 500 MS/s, meaning at least 500 regression models *can* be built. In practice, however, most information is found at clock rising edges. Y_t is therefore the peak amplitude at rising edges of clock t . A sum of 26 points near the peak gives slightly better results than using the single peak value. We denoise the traces for use in regression by averaging over only 100 EM traces. Selection of the predictor variables still poses a challenge.

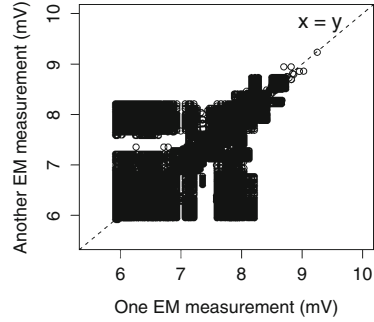
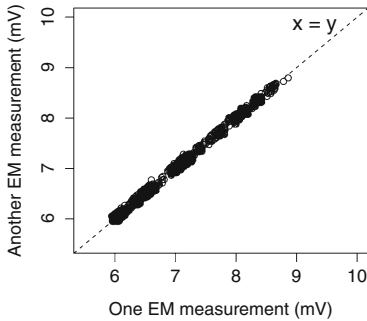
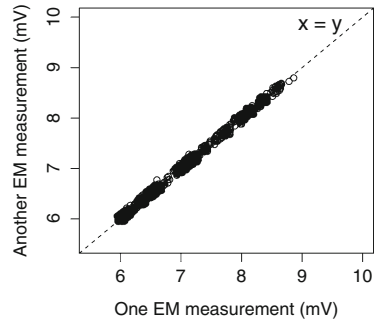
Black-box Model-building. Switches of internal signals and voltage differences of neighboring signals are a promising initial choice for \mathbf{x}_t , supported by research on power consumption of FPGAs and general circuits [21, 25, 31, 47, 50]. Because the design details of NIOS II are not available, we initially treat the system as a black box and attempt to reason about internal activities directly from the instruction set documents, as in [37]. However, the EM samples correlate poorly with predictions. This is not surprising, as the target SoC systems are much more complex than the PIC microcontroller in previous work. In particular, unlike the PIC chip, there is no dominant power-consuming memory interface. We instead turn to the simulation models of the processor cores.

For system A, register-transfer level (RTL) simulation gives, for example, runtime values of thousands of signals, including 35 bus signals; gate-level post-fit simulation gives runtime values of at least 8259 1-bit internal signals. For system B, and system C, there are over 5800 and 12606 1-bit gate-level signals, respectively. EM radiation must be related with these signals in some form. However, directly estimating $f(\cdot)$ does not work due to the sheer number of signals, and also due to the multicollinearity among the signals (many signals are highly correlated with each other, and thus only one signal in a correlated set may be a useful predictor). Some signals are even identical – a simulation artifact. More variables are identical when considering switches of signals (transitions from 0 to 1 or vice versa). However, removing duplicate variables does not eliminate multicollinearity, showing that more complex correlations exist among the signals and signal switches.

As a first step in selecting representative signals for \mathbf{x}_t , we test whether the EM radiation has similar amplitudes when a subset of internal signals stay the same while others vary. If so, we need only to retain the subset of signals for model building. Figure 3a shows pairs of EM measurements (peak amplitudes) when bus signals are identical while other signals vary. The x-axis is one EM measurement, and y-axis is another EM measurement. Figure 3b shows pairs of EM measurements when signal transitions (0-1 and 1-0 are regarded as different transitions) of bus signals are identical. These two figures mean that, *no matter what the form of $f(\cdot)$ is*, the EM radiation is not determined only by the bus signals. This is in contrast with the PIC chip, whose EM radiation is dominated by the Hamming distance of bus signals. Therefore, we must include additional variables in \mathbf{x}_t . Because it is impractical to try arbitrary subsets of signals, we have to turn to the gate-level signals, as RTL signals are optimized out in final



(a) Identical bus signals

(b) Identical *transitions* of bus signals(c) Identical transitions of selected *gate-level* signals(d) Identical *Hamming distance* of selected *gate-level* signals**Fig. 3.** Measurements of system A when a subset of internal states are identical

layout of the SoC system. However there are thousands of gate-level signals. To select the most representative ones, we utilize the vendor-provided power estimation tool PowerPlay [6], therefore taking the gray box modeling approach, since PowerPlay encodes partial knowledge of the FPGA design.

PowerPlay is a tool for developers to estimate power consumption of an FPGA system to allow selection of power supply and heat dissipation scheme. Total thermal power estimates are claimed to have $\pm 20\%$ accuracy to silicon. However, since PowerPlay only reports comparatively rough estimates of accumulative power consumption, it cannot be directly used to solve our problem, which requires side-channel models for at least each instruction. PowerPlay can, however, generate a set of signal names for use in a gate-level simulation (which is in turn used for power estimation). It is reasonable to assume that these signals contribute more significantly to power consumption (thus causing more EM radiation). For system A, 1778 (out of 8259) gate-level signals are selected by PowerPlay, a huge reduction in variables requiring post-processing.

We again test whether EM radiation is similar when the 1778 signals are identical while others vary. Figure 3c shows pairs of EM measurements when signal transitions are identical for the 1778 variables, and Fig. 3d shows pairs of EM measurements when the HDs (which do not distinguish between 0-1 and 1-0) are identical. We do not find enough pairs whose absolute values of the 1778 variables are identical. Nevertheless, Fig. 3d already illustrates that it is reasonable to select HD of the 1778 variables as \mathbf{x}_t , regardless the real form of $f(\cdot)$ (Note that the set of points in Fig. 3c is a subset of those in Fig. 3d). After modeling is finished, we retry using the original 8259 gate-level signals, and find that indeed modeling with the 1778 signals gives better results. For systems B and C the number of selected signals is 1280 and 2715, respectively.

However, multicollinearity still exists among the selected variables. Since further dividing the selected variables based on SoC structure does not lead to improvement and exhaustive search is computationally impractical, we turn to statistical techniques. There are several which can deal with predictors that have multicollinearity: ridge regression, partial least-square regression (PLS), principal component regression (PCR), and stepwise regression. For the selected variables, we find that all regression techniques produce similarly good regression models in terms of the coefficients of determination R^2 , MSE , and F -tests in the model building step. To test validity of the regression techniques, we perform model validation to measure model reasonableness and predictive power.

7 Validation

We perform five-fold cross-validation to test the ability of the regression model in predicting EM radiation. Among the test programs (see Sect. 5.4), half are used for modeling and the other half for testing. One FPGA chip is used for building the EM model and a different FPGA chip of the same model is used for testing the model. This stricter five-fold (compared to the common seven-fold or even ten-fold) validation scheme is used because it is impractical to perform exhaustive exploration and associated physical measurements, so we are forced to use a limited set of programs for side-channel profiling and derive a model which accurately predicts the experimental results from all other possible programs. The goal is to evaluate the validity of using above variable selection approach and regression techniques for side-channel profiling, rather than to obtain a specific “best” model. Since some of the combinations of modeling/testing code may yield better results, cross-validation eliminates this problem by repeating the modeling and testing procedure using different programs for modeling and testing each time. We exhaustively compute all 252 possible combinations. We assume $f(\cdot)$ can be approximated as a first-order linear function for now. Pearson’s r and Spearman’s ρ are used to evaluate the quality of our models – the larger the correlation, the greater the predictive power. Pearson’s r is effective because we observe that slightly moving the probe will only cause the amplitude of EM radiation to change linearly. We still compute Spearman’s ρ , which can reveal nonlinear relationships between measurements and models (r and

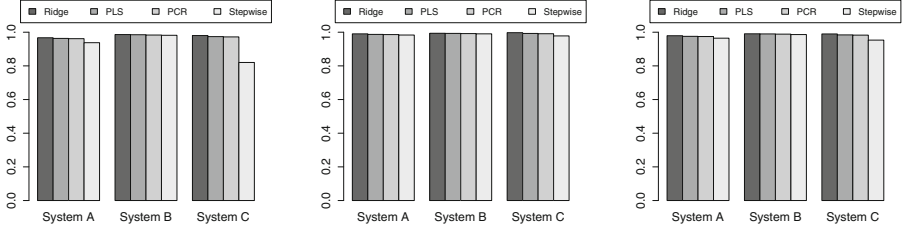
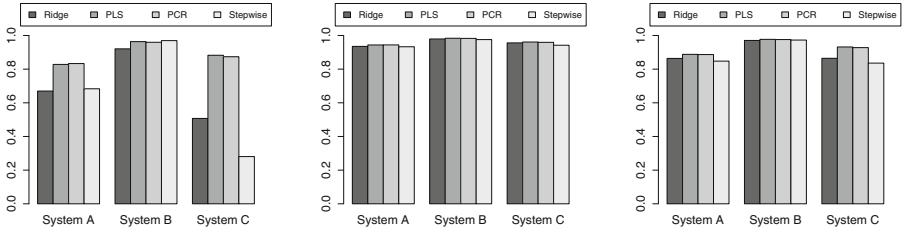
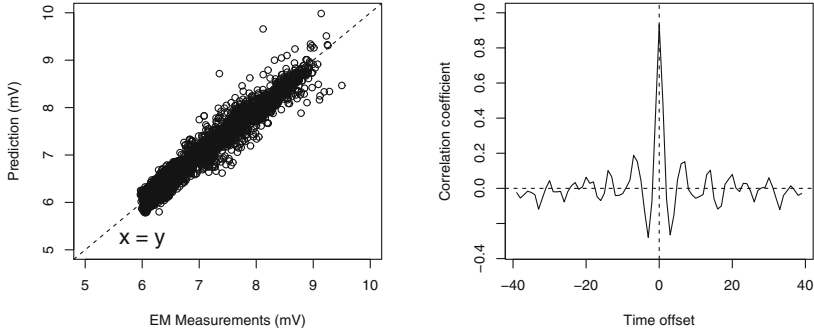
(a) Minimum r of profiling (b) Maximum r of profiling (c) Mean r of profiling(d) Minimum r of testing (e) Maximum r of testing (f) Mean r of testing

Fig. 4. Pearson’s r for model validation, with the profiling (modeling) results in the top row ((a)–(c)) and the testing (on another chip) results in the bottom row ((d)–(f)); from the darkest bar to the lightest are ridge regression, PLS, PCR, and stepwise regression.

ρ are equivalent when relationships are linear). Pearson’s r is shown in Fig. 4. Spearman’s ρ results are almost identical, validating our choice, and omitted due to space constraints. In addition, regressing \mathbf{x}_t always has the highest correlation coefficients when \mathbf{x}_t and Y_t are aligned in time (omitted due to space constraints). When profiling and testing using the same FPGA chip, all the r and ρ values are slightly better (omitted due to space constraints), as expected.¹

The results show that (with few exceptions) linear regression models can predict EM radiation of new programs with satisfactory accuracy, especially for system B. Adding the absolute values of the signals (i.e., Hamming weights) to \mathbf{x}_t does not improve model performance. PLS and PCR outperform other techniques and are stable in all cases (with no unacceptably low-performing outliers $r < 0.80$). PLS and PCR have been used in various domains such as chemistry and biology, where, similar to our situation, one observation is associated with many variables [24, 44]. PLS and PCR have nearly identical performance,

¹ The parameters of each regression technique are selected to achieve best results for a few pre-selected random modeling/testing combinations and then fixed for all the others. Note that although for a particular combination the best parameter varies, it does not change our conclusions.



(a) Model prediction (y-axis) versus measurements (x-axis) (b) Correlation of model prediction and measurement with sliding time window

Fig. 5. Model prediction and measurements for the best PLS model of system A.

which is interesting since unlike PLS, PCR does not take the response Y_t into account when selecting the predictors. Principal component analysis (PCA) has been used in side-channel analysis as a preprocessing step of pattern matching or classification to reduce dimension and to denoise the sampled traces in *time* [12, 22]. We instead use PCA to eliminate multicollinearity for regression. Note that there is no noise in the predictors \mathbf{x}_t , which are not random variables.

The effectiveness of the regression models is further shown in Fig. 5. The x-axis of Fig. 5a is the actual measurement for a testing program, and the y-axis is the model prediction (of the best PLS model of system A). Although some outliers exist, most measurements and predictions fall along the line of $x = y$. Figure 5b shows the Pearson's r between the actual measurement of a testing program and the model prediction which has an offset in time from the actual measurement. The x-axis of Fig. 5b is the time offset, and the y-axis is r . A sharp peak occurs when the model prediction and real measurement have no time offset, showing the soundness and validity of the model.

Second, we examine the resulting models for the reasonableness of their coefficients. We observe that PLS and PCR result in similar regression coefficients for each system. Several selected signals are clock signals that switch at each clock cycle. These signals do not provide information on internal states, and should only contribute to the constant in the model. Only ridge regression assigns non-zero coefficients to these signals. This is due to the procedure of ridge regression and has caused it to perform worse. PowerPlay reports that the M9K component consumes the majority ($\sim 60\%$) of the core dynamic power, but only a portion of M9K signals have larger regression coefficients in our resulting models. We have regressed separately with the M9K signals including memory and registers banks, as well as other signals reported in PowerPlay as consuming more power, yet the resulting models are *not* better than original models, especially in cross-validation.

8 Applying Results to Software Integrity Checking

To enforce code integrity, we must guarantee that tampering with the internal states of the system can be detected from side-channel measurements. Given the regression model, we predict EM radiation of new programs by using only gate-level simulation. We can therefore determine whether tampering with the original code will be reflected in the emission or not.

We first consider a conventional attacker, who is unable to analyze the side-channel of the system or unaware of the existence of a side-channel-based integrity checking mechanism. Integrity checking is a hypothetical test of whether a given measurement is from tampered code/data (undesired state) or not. The performance of this integrity mechanism is quantified by (1) the false positive rate (when a normal system state is flagged as tampering), and (2) the false negative rate (when tampering is performed, but is not flagged).

Table 1. False positive rates (%) for a(n) (aligned) single-captured EM trace

Threshold	0.90			0.85		
Number of cycles	7	14	21	7	14	21
System A	14.2	12.9	9.68	8.73	5.92	3.31
System B	13.1	10.3	8.57	8.52	6.14	3.08
System C	16.7	14.7	13.1	9.14	5.53	3.80

Table 2. False negative rates (%) for an aligned single-captured EM trace

Threshold	0.90			0.85		
Number of cycles	7	14	21	7	14	21
System A	20.6	4.65	1.75	30.5	9.83	4.01
System B	5.74	0.99	0.59	10.12	1.98	1.01
System C	0.81	0.18	0.13	1.54	0.22	0.14

Table 3. False negative rates (%) for an arbitrary single-captured EM trace

Threshold	0.90			0.85		
Number of cycles	7	14	21	7	14	21
System A	3.41	0.70	0.24	5.65	1.51	0.56
System B	1.43	0.16	0.09	2.93	0.38	0.16
System C	0.67	0.11	0.10	1.33	0.14	0.10

Recall that the SNR of our experiment is 15 dB. Taking both environmental noise and regression residual into account, we obtain from the best PLS models and EM measurements that for system A, 85.8% of the Pearson’s r between

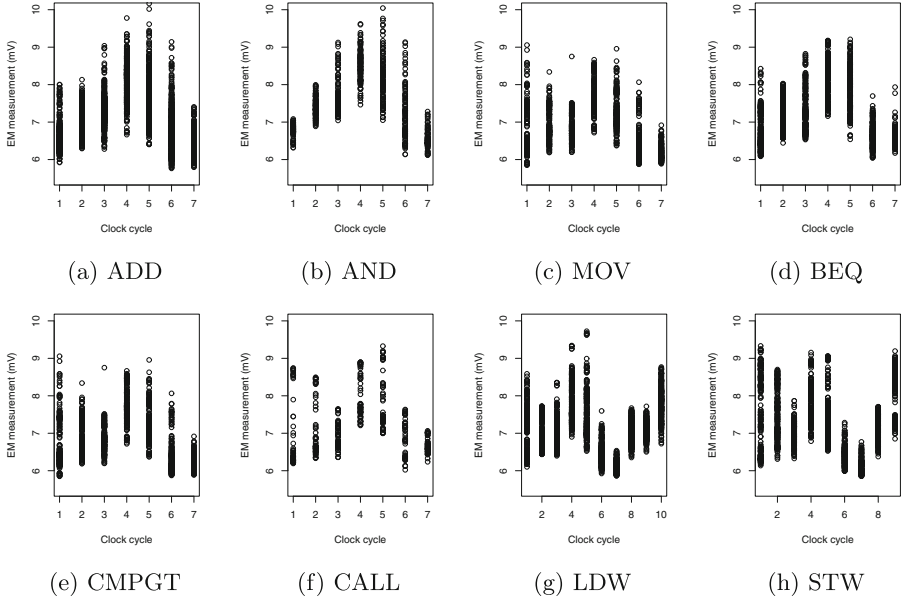


Fig. 6. EM measurements of instructions grouped by operations for system A.

seven-cycle **single-captured** traces (on the testing chip) with the model prediction (from the profiling chip) is greater than 0.90 to execute at most one instruction. We can design the integrity checking mechanism by fixing the threshold 0.90, and then compute the false positive rate of any 7-/14-/21-cycle trace for each system from real measurements. Table 1 lists the false positive rates when the threshold is fixed to 0.90 and 0.85. Seven cycles are chosen because the *actual* number of clock cycles per instruction for NIOS II is from seven to 39. While the actual number of clock cycles per instruction for OpenMSP430 ranges from one to six, we use the same intervals for comparison purpose.

The false negative rate is computed from the percentage of 7-/14-/21-cycle execution traces of different code and/or data on the testing chip, but having r greater than the threshold with the model prediction of executing target code with desired data. Table 2 lists the false negative rates when the EM traces are aligned with starts of execution, applicable to the case in which tampered code/data executes in the same number of clock cycles with the target code. Table 3 lists the false negative rates for arbitrary EM traces that are aligned or misaligned with the original one.

The results show that the probability of random malware evading the integrity checking is very low. Even compact malware (very few instructions) can be detected reliably. Both false positive and false negative rates decrease rapidly as the number of clock cycles increases. When the number of cycles is fixed, there is a tradeoff between false positives and false negatives: a lower threshold will reduce false positives at a cost of higher false negatives. Note that

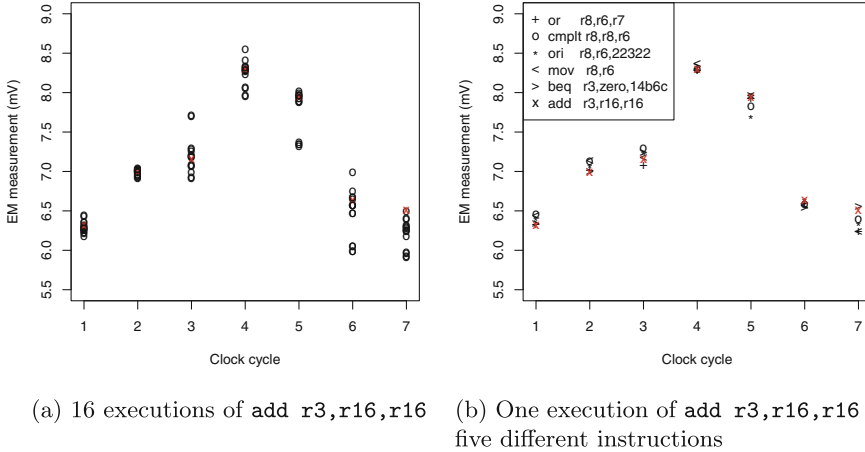


Fig. 7. EM measurements of `add r3,r16,r16` for system A; red cross indicates the same instance of executing `add r3,r16,r16`. (Color figure online)

the threshold and number of cycles can be computed to achieve desired false positive and false negative rates. Overall, the side-channel-based integrity check is effective to detect a conventional attacker.

Next, we consider a side-channel-aware attacker who actively tries to compute alternative code that has near indistinguishable EM radiation from the original code. To rewrite malware with semantically equivalent code but having similar EM measurements with the normal program, the attacker needs to know the reverse mapping from EM radiation to runtime state including instructions and data. The success rate of preventing the attacker from doing so relies on the hardness to obtain such mapping. We first analyze the side-channel of instructions classified by operations (e.g., `add`, `call`), as done in previous research [14, 22, 26, 51, 56]. We find that significant variation exists among instructions of the same operations, and EM measurements of different operations are not discriminatory. Figure 6 shows the EM measurements grouped by operations. Figure 7a shows an example in which even when executing the same instruction (`add r3,r16,r16`), the EM radiation varies significantly. On the other hand, EM measurements of executing different instructions may have nearly the same value. Figure 7b shows an example in which the EM trace of one execution of `add r3,r16,r16` has nearly the same value with those of five different instructions of different operations. This can be quantified by class (i.e., operation) separability by using within-class and between-class scatter matrices:

$$J = \frac{\text{trace}(S_m)}{\text{trace}(S_w)}, S_m = S_w + S_b$$

$$S_w = \sum_{i=1}^M P_i \Sigma_i, S_b = \sum_{i=1}^M P_i (\mu_i - \mu_0)(\mu_i - \mu_0)^T$$

where each operation is a class, P_i is the a priori probability of operation i , M is the total number of operations, μ_i is the mean of operation i , Σ_i is the covariance of i , and $\mu_0 = \sum_{i=1}^M P_i \mu_i$ is the global mean vector. We compute the statistics of 53 common operations. The resulting J is 1.23, very close to one, which means that EM radiation, when grouped by operations, is not well clustered and is nearly indistinguishable from each other. For system C, the number of clock cycles per instruction varies from one to six, and depends on the addressing modes of the source and destination operands. Table 4 shows class separability for different clock cycles. The resulting J is still close to one.

Table 4. Class separability J for system C.

One cycle	3.81	Four cycles	1.71
Two cycles	16.63	Five cycles	1.54
Three cycles	7.51	Six cycles	1.37

As shown in Sect. 6, the EM model is a function of thousands of selected gate-level signal switches. The operations, bus signals, and M9K signals, which can be easily deduced from code, only contribute to a small portion of EM variance. Even if exactly the same instruction is executed, different runtime state of other signals will cause significantly different EM radiation. The attacker has to rewrite malware based on the many thousands gate-level signals which cannot be manipulated arbitrarily, but rather through the programming model of the processor. Furthermore, the gate-level signals are synthesized and optimized results of the processor core, whose relationship with the assembly code is unknown. Without knowing the design of the processor, as in the case of NIOS II, or without the ability to deal with processor complexity, the attacker will have to exhaustively search for alternative malware code that has similar EM radiation. In addition, each combination of operation and operands will result in a different internal state at each clock cycle. As the length of EM measurements increases linearly, the complexity of searching increases exponentially, effectively making the attack impractical. Detailed information on experiment setup, data, two ports of test code, and results are available at [2].

9 Discussion and Future Work

We have quantified the effectiveness and generality of using low-cost acquisition equipment to verify runtime states of three FPGA-based SoCs passively and non-invasively, against both conventional and side-channel-aware attackers. Profiling and testing use different chips (boards) of the same model. We show that by using our variable selection procedure and regression techniques, it is possible to model EM radiation of complex and gray-box processor-core-based SoC systems with high accuracy at clock-cycle granularity. Linear regression has also been used to

break cryptographic hardware from side-channel leakage [15, 32, 39, 52]. Note that attacking cryptographic hardware is chiefly concerned with special time points that leak key materials during multiple executions of the same cryptographic routine. In contrast, for integrity checking, we must detect one-time execution of malware from a single measurement and must consider instruction operations, operands, register and memory content.

Since we build side-channel models from simulation results, it can be inferred that directly applying the method for integrity checking requires the system to be deterministic. For example, no context switching should happen when measuring a target program. To what extent our approach can be applied for integrity checking of non-deterministic systems is left for future work.

Acknowledgments. This work was supported in part by NSF grant 1253930.

References

1. CoreMark. <http://www.eembc.org/coremark/>
2. Experiment Setup and Data. <http://cis.ksu.edu/~hongl/fpga/>
3. FreeRTOS. <http://www.freertos.org/>
4. NIOS II Processor Reference Handbook. https://www.altera.com/en_US/pdfs/literature/hb/nios2/n2cpu_nii5v1_01.pdf
5. The OpenMSP430 Project. <http://opencores.org/download,openmsp430>
6. PowerPlay Early Power Estimator. https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_epe.pdf
7. Aciğmez, O., Koç, C.K., Seifert, J.-P.: On the power of simple branch prediction analysis. In: ASIACCS (2007)
8. Anderson, J.H., Najm, F.N.: Power estimation techniques for FPGAs. *IEEE VLSI* **12**(10), 1015–1027 (2004)
9. Armknecht, F., Sadeghi, A.-R., Schulz, S., Wachsmann, C.: A security framework for the analysis and design of software attestation. In: CCS (2013)
10. Asokan, N., Brassler, F., Ibrahim, A., Sadeghi, A.-R., Schunter, M., Tsudik, G., Wachsmann, C.: SEDA: scalable embedded device attestation. In: CCS (2015)
11. Baek, Y.-J., Gratzer, V., Kim, S.-H., Naccache, D.: Extracting unknown keys from unknown algorithms encrypting unknown fixed messages and returning no results. In: Sadeghi, A.R., Naccache, D. (eds.) *Towards Hardware-Intrinsic Security: Foundations and Practice*, pp. 189–197. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14452-3_8
12. Batina, L., Hogenboom, J., van Woudenberg, J.G.J.: Getting more from PCA: first results of using principal component analysis for extensive power analysis. In: Dunkelmann, O. (ed.) *CT-RSA 2012. LNCS*, vol. 7178, pp. 383–397. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27954-6_24
13. Bletsch, T., Jiang, X., Freeh, V.W., Liang, Z.: Jump-oriented programming: a new class of code-reuse attack. In: ASIACCS (2011)
14. Bohy, L., Neve, M., Samyde, D., Quisquater, J.-J.: Principal and independent component analysis for crypto-systems with hardware unmasked units. In: *e-Smart* (2003)
15. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) *CHES 2004. LNCS*, vol. 3156, pp. 16–29. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_2

16. Butterworth, J., Kallenberg, C., Kovah, X., Herzog, A.: BIOS chronomancy: fixing the core root of trust for measurement. In: CCS (2013)
17. Checkoway, S., Feldman, A.J., Kantor, B., Halderman, J.A., Felten, E.W., Shacham, H.: Can DREs provide long-lasting security? The case of return-oriented programming and the AVC advantage. In: EVT/WOTE (2009)
18. Clark, S.S., Ransford, B., Rahmati, A., Guineau, S., Sorber, J., Fu, K., Xu, W.: WattsUpDoc: power side channels to nonintrusively discover untargeted malware on embedded medical devices. In: HealthTech (2013)
19. Dam, M., Guanciale, R., Khakpour, N., Nemati, H., Schwarz, O.: Formal verification of information flow security for a simple ARM-based separation kernel. In: CCS (2013)
20. Drimer, S.: Volatile FPGA design security - a survey. http://www.cl.cam.ac.uk/~sd410/papers/fpga_security.pdf
21. Duan, C., Cordero, V., Khatri, S.P.: Efficient on-chip crosstalk avoidance CODEC design. IEEE VLSI **17**(4), 551–560 (2009)
22. Eisenbarth, T., Paar, C., Weghenkel, B.: Building a side channel based disassembler. In: Gavrilova, M.L., Tan, C.J.K., Moreno, E.D. (eds.) Transactions on Computational Science X. LNCS, vol. 6340, pp. 78–99. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17499-5_4
23. Francillon, A., Nguyen, Q., Rasmussen, K.B., Tsudik, G.: A minimalist approach to remote attestation. In: DATE (2014)
24. Frank, L.E., Friedman, J.H.: A statistical view of some chemometrics regression tools. Technometrics **35**(2), 109–135 (1993)
25. Goeders, J.B., Wilton, S.J.E.: VersaPower: power estimation for diverse FPGA architectures. In: ICFPT (2012)
26. Goldack, M.: Side-channel based reverse engineering for microcontrollers. Master's thesis, Ruhr-Universität Bochum, Germany (2008)
27. Gonzalez, C.R.A.: Power fingerprinting for integrity assessment of embedded systems. Ph.D. thesis, Virginia Polytechnic Institute and State University (2011)
28. Gonzalez, C.R.A., Reed, J.H.: Power fingerprinting in SDR & CR integrity assessment. In: MILCOM (2009)
29. Gu, L., Ding, X., Deng, R.H., Xie, B., Mei, H.: Remote attestation on program execution. In: STC (2008)
30. Jin, Y., Kupp, N., Makris, Y.: Experiences in hardware Trojan design and implementation. In: HOST (2009)
31. Kadric, E., Lakata, D., DeHon, A.: Impact of memory architecture on FPGA energy consumption. In: FPGA (2015)
32. Kasper, M., Schindler, W., Stottinger, M.: A stochastic method for security evaluation of cryptographic FPGA implementations. In: FPT (2010)
33. Kocher, P., Jaffe, J., Jun, B., Rohatgi, P.: Introduction to differential power analysis. J. Cryptogr. Eng. **1**(1), 5–27 (2011)
34. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
35. Kömmerling, O., Kuhn, M.G.: Design principles for tamper-resistant smartcard processors. In: USENIX Smartcard (1999)
36. Li, Y., McCune, J.M., Perrig, A.: VIPER: verifying the Integrity of PERipherals' firmware. In: CCS (2011)

37. Liu, H., Li, H., Vasserman, E.Y.: Practicality of using side-channel analysis for software integrity checking of embedded systems. In: Thuraisingham, B., Wang, X.F., Yegneswaran, V. (eds.) *SecureComm 2015*. LNCS, vol. 164, pp. 277–293. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-28865-9_15
38. Liu, Y., Wei, L., Zhou, Z., Zhang, K., Xu, W., Xu, Q.: On code execution tracking via power side-channel. In: *CCS (2016)*
39. Lomné, V., Prouff, E., Roche, T.: Behind the scene of side channel attacks. In: Sako, K., Sarkar, P. (eds.) *ASIACRYPT 2013*. LNCS, vol. 8269, pp. 506–525. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-42033-7_26
40. Davi, L., Sadeghi, A.R., Lehmann, D., Monrose, F.: Stitching the gadgets: on the ineffectiveness of coarse-grained control-flow integrity protection. In: *SEC (2014)*
41. Dam, M., Guanciale, R., Nemati, H.: Machine code verification of a tiny ARM hypervisor. In: *TrustED (2013)*
42. Mahmood, A., McCluskey, E.: Concurrent error detection using watchdog processors - a survey. *Trans. Comput.* **37**(2), 160–174 (1988)
43. Mohan, V., Larsen, P., Brunthaler, S., Hamlen, K.W., Franz, M.: Opaque control-flow integrity. In: *NDSS (2015)*
44. Montgomery, D.C., Peck, E.A., Vining, G.G.: *Introduction to Linear Regression Analysis*, 5th edn. Wiley, Hoboken (2012)
45. Moreno, C., Fischmeister, S., Hasan, M.A.: Non-intrusive program tracing and debugging of deployed embedded systems through side-channel analysis. In: *LCTES (2013)*
46. Msgna, M., Markantonakis, K., Naccache, D., Mayes, K.: Verifying software integrity in embedded systems: a side channel approach. In: Prouff, E. (ed.) *COSADE 2014*. LNCS, vol. 8622, pp. 261–280. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10175-0_18
47. Muralimanohar, N., Balasubramonian, R., Jouppi, N.P.: *CACTI 6.0: A Tool to Model Large Caches (2009)*
48. Ott, H.W.: *Electromagnetic Compatibility Engineering*. Wiley, Hoboken (2009)
49. Perrig, A., van Doorn, L.: Refutation of “On the difficulty of software-based attestation of embedded devices” (2010). <http://www.netsec.ethz.ch/publications/papers/perrig-ccs-refutation.pdf>
50. Poon, K.K.W., Wilton, S.J.E., Yan, A.: A detailed power model for field-programmable gate arrays. *ACM TODAES* **10**(2), 279–302 (2005)
51. Quisquater, J.-J., Samyde, D.: Automatic Code Recognition for Smart Cards Using a Kohonen Neural Network (2002)
52. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) *CHES 2005*. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005). https://doi.org/10.1007/11545262_3
53. Senn, L., Senn, E., Samoyeau, C.: Modelling the power and energy consumption of NIOS II softcores on FPGA. In: *Cluster Computing Workshops (2012)*
54. Seshadri, A., Perrig, A., Doorn, L.V., Khosla, P.: SWATT: SoftWare-based ATTestation for embedded devices. In: *IEEE S&P (2004)*
55. Strobel, D., Bache, F., Oswald, D., Schellenberg, F., Paar, C.: Scandalee: a side-channel-based disassembler using local electromagnetic emanations. In: *DATE (2015)*
56. Strobel, D., Oswald, D., Richter, B., Schellenberg, F., Paar, C.: Microcontrollers as (in)security devices for pervasive computing applications. *Proc. IEEE* **102**(8), 1157–1173 (2014)

57. Sugawara, T., Suzuki, D., Saeki, M., Shiozaki, M., Fujino, T.: On measurable side-channel leaks inside ASIC design primitives. *J. Cryptogr. Eng.* **4**(1), 59–73 (2014)
58. Tiwari, V., Malik, S., Wolfe, A., Lee, M.T.-C.: Instruction level power analysis and optimization of software. In: *VLSI Design* (1996)
59. Vermoen, D., Witteman, M., Gaydadjiev, G.N.: Reverse engineering Java card applets using power analysis. In: Sauveron, D., Markantonakis, K., Bilas, A., Quisquater, J.-J. (eds.) *WISTP 2007*. LNCS, vol. 4462, pp. 138–149. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72354-7_12
60. Whitnall, C., Oswald, E.: Robust profiling for DPA-style attacks. In: Güneysu, T., Handschuh, H. (eds.) *CHES 2015*. LNCS, vol. 9293, pp. 3–21. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_1
61. Yang, Y., Su, L., Khan, M., Lemay, M., Abdelzaher, T., Han, J.: Power-based diagnosis of node silence in remote high-end sensing systems. *ACM Trans. Sens. Netw.* **11**(2), 33 (2014)
62. Zhang, F., Wang, H., Leach, K., Stavrou, A.: A framework to secure peripherals at runtime. In: Kutylowski, M., Vaidya, J. (eds.) *ESORICS 2014*. LNCS, vol. 8712, pp. 219–238. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11203-9_13
63. Zipf, P., Hinkelmann, H., Deng, L., Glesner, M., Blume, H., Noll, T.G.: A power estimation model for an FPGA-based softcore processor. In: *FPL* (2007)