Practicality of Using Side-Channel Analysis for Software Integrity Checking of Embedded Systems

Hong Liu, Hongmin Li, and Eugene Y. Vasserman(✉)

Department of Computing and Information Sciences,
Kansas State University, Manhattan, KS 66506, USA
{hongl,hongminli,eyv}@ksu.edu

Abstract. We explore practicality of using power consumption as a non-destructive non-interrupting method to check integrity of software in a microcontroller. We explore whether or not instructions can lead to consistently distinguishable side-channel information, and if so, how the side-channel characteristics differ. Our experiments show that data dependencies rather than instruction operation dependencies are dominant, and can be utilized to provide practical side-channel-based methods for software integrity checking. For a subset of the instruction set, we further show that the discovered data dependencies can guarantee transformation of a given input into a unique output, so that any tampering with the program by a side-channel-aware attacker can either be detected from power measurements, or lead to the same unique set of input and output.

Keywords: Side-channels · Power consumption · Software integrity · Security · Embedded systems

1 Introduction

Checking software integrity is a fundamental problem of system security. Given a device under test (DUT), a verifier tries to determines whether it runs the desired code or not. Developers traditionally focus on realizing functionality, while ignoring the fact that an attacker can change the behavior of the DUT by overwriting its program and/or data remotely [15,17,18,23] or locally [7,11,13,25].

Many approaches have been proposed to try to enforce that a device runs the original code. The approaches can be classified by where the verifier resides. An *internal* approach resides in the same device with the target software. Hypervisors [30,45], mandatory access control [1,42], and control flow integrity [14,19], are internal software-based approaches that aim to prevent "anomalous behavior" of programs that share the same hardware with the verifier. Watchdog coprocessors [33,38] and TPM [4,9] are internal hardware-based approaches that examine hardware status such as "signatures" of code that appear on buses or statistics of software and firmware to prevent deviations from the original design.

B. Thuraisingham et al. (Eds.): SecureComm 2015, LNICST 164, pp. 277–293, 2015.
DOI: 10.1007/978-3-319-28865-9_15

The verifier can also be outside of the DUT, leading to *external* verification. Software attestation [27,34,44] and remote attestation [21] are approaches in which a verifier external to the DUT asks the DUT to provide evidence of integrity from time to time and checks it against prior knowledge of hardware and software configuration and/or shared secrets.

Another promising external approach is to check evidence of integrity from side-channels. Unlike attestation, which communicates with the DUT explicitly and actively, this approach tries to identify tampering by analyzing passive information leakage from the DUT, such as timing of network traffic, power consumption, electromagnetic (EM) emissions, light emissions, vibrations, etc. [6,24,35–37]. These channels are "side" because they are unavoidable byproducts of implementing the desired functionality on a physical device. A side-channel approach has advantages over other approaches in that

1. It does not interfere with the normal execution of the DUT – the DUT does not even know about the existence of the verifier;
2. since the DUT does not have a verifier implemented, an attacker who successfully penetrates into the DUT still does not know about the existence or the implementation of the verifier;
3. verification instrumentation and algorithms can be easily updated;
4. it works with legacy devices that cannot implement modern integrity checking techniques;
5. it works with attacks against CAD tools which may tamper the debugging and programming traffic and therefore fail all internal protection mechanisms.

Previous research has been successful in using side-channels to check IC integrity [6,24,35]. By comparing side-channel information of the DUT to that of the "golden samples", researchers are able to find minimal differences that indicate tampering of the design. A great number of embedded systems, however, are based on general-purpose microcontrollers/microprocessors. Detailed hardware information about the microcontrollers (μCs) are in general not accessible to system developers. It is therefore hard to obtain "golden samples" for side-channel analysis (SCA).

Using side-channel information for integrity checking of μCs without detailed design information poses a great challenge. Given a set of samples of side-channel emissions, we need to extract instruction-level information about the running device. The sample is an aggregation of power consumption cost by reading memory, executing instructions, accessing peripherals, and noise. In the worst case the tampered code only gets executed once during sampling. The verifier therefore does not have the advantage of reducing noise in samples by averaging thousands of execution traces, as in DPA [26,28].

Previous attempts on instruction-level SCA have been focused on reverse engineering of instruction operations [16,20,39] by using either the power consumption or the EM side-channel, and have achieved different degrees of success. One recent work [31] proposed using instruction-level power consumption SCA

for software integrity checking, yet was found not repeatable on a different (but simpler) μC [39].

Current trends in SCA demand more and more advanced acquisition equipment such as broadband high-sensitivity oscilloscopes, Picosecond Imaging Circuit Analysis [37], micro magnetic-field probes [40], etc. Occasions that need SCA-based checking for legacy or low-cost μCs are not always able to afford such equipment. Another major obstacle is noise both from the ambient environment and from the DUT. As shown by research on breaking cryptographic embedded systems [5], power consumption is mainly due to bus traffic as opposed to the smaller currents within a CPU.

In this work, we propose practical methods and results for power-based software integrity checking. Our contributions are:

- We point out pitfalls in previous work that an attacker will always try to replace instructions with those that have similar side-channel characteristics, and thus turns any ($< 100\%$) recognition rate on random code into near-0 on crafted code.
- We propose a systematic approach for SCA profiling which enables us to design experiments and analyze the effects of runtime status on power consumption efficiently.
- We show mechanisms that determine side-channel characteristics. The results have direct implications on using simple (versus differential) SCA for software integrity checking of embedded systems in practice.
- For a subset of the instruction set, we show that the data dependencies we have discovered are enough to guarantee unique transformations of input and output. So, the verifier can ensure that even if the program is altered by a side-channel-aware attacker, as long as the side-channel measurements are the same, the program still computes the same value.

2 Related Work and Pitfalls

Research on SCA is mostly focused on breaking cryptographic hardware, including general-propose μCs, FPGAs, and ASICs. The goal is to extract secret keys by analyzing several thousands of executions of cryptographic routines [22,26,28,35]. Cryptographic routines are in general publicly available. In our case, in contrast, only a single trace of side-channel emission is available, and we also need to derive runtime instructions from side-channel measurements. Techniques in breaking cryptographic hardware are therefore not directly applicable to SCA for software integrity checking.

SCA for IC integrity relies on full knowledge of the IC design. By scanning emissions of the IC for enough time, it is possible to detect untriggered trojan circuits [6,24,36,37]. For software-integrity checking of μCs, detailed knowledge of the IC design is not available. It is therefore not possible either to use simulation tools or to infer power consumption from the architecture design.

At the system level, SCA has been used to provide preliminary detection of abnormal behaviors such as malware and anomalous reboots. Yang et al. [43]

used external power measurements to distinguish between several categories of failures in remote high-end sensing systems. WattsUpDoc [12] applies machine learning to detect untargeted malware by monitoring system-wide AC (wall outlet) power consumption of medical devices and SCADA systems that run variants of the Windows operating system. WattsUpDoc specifically excludes malware that is designed to evade power analysis. While an aggressive malware may be visible at the system level through abnormal power consumption (e.g., by draining too much energy), a stealthy malware will hide itself in the noise introduced from multiple components that are running in parallel in a big system. Real malware detection requires instruction-level integrity checking techniques.

Previous work on instruction-level SCA uses random data input, PCA+LDA and template analysis [16,20,31,39]. In particular, [39] claims a relevant recognition rate of 96.24% on test data and 87.69% on real code by using multi-position localized EM emissions and semi-invasive access to the chip. In [31], a 100% classification rate was reported by using power measurements, However, neither [39] nor us succeeded in repeating the authors' results on a different (but simpler) μC.

There are two major shortcomings in all of the previous work. First, previous work has been focused on recovering the instruction operations. Data, including operands and values of registers are regarded as noise. Operands and other runtime status such as PC are therefore not known.

The more significant drawback of previous research, however, is that little is known about the reason for failure of recognition. Given any non-100% average recognition rate, an attacker will naturally try to write malware utilizing only the misclassified instructions to any extent possible, and therefore turns a high recognition rate on random code into 0 on crafted code, in a way similar to [29]. This problem is fatal both for reverse engineering and for software integrity checking. It is more demanding to discover whether instructions can lead to distinguishable side-channel information or not and, if so, how the side-channel characteristics differ.

3 A Systematic Approach for Instruction-level Side-channel Analysis

We use PIC16F687 as our DUT, because most previous research in this area has been performed on this IC [16,20,31,39]. We assume that the attacker is able to modify the software of the DUT, for instance by reprogramming the device or inserting trojans into the CAD software. The attacker is able to profile the side-channel emissions of the DUT and to modify the software in a fashion that minimizes side-channel deviations. The attacker is however unable to modify the hardware, including the IC design and the PCB on which the DUT is mounted.

PIC16F687 is a 8-bit RISC μC in Harvard architecture. It has a 14-bit program bus, which is connected to the program flash, and a 8-bit data bus, which is connected to RAM, EEPROM, PORTs, ADC, etc. The instruction set has 35 operations, all executed in single instruction cycle, except branches. The processor has a two-stage pipeline, therefore unconditional and conditional branches

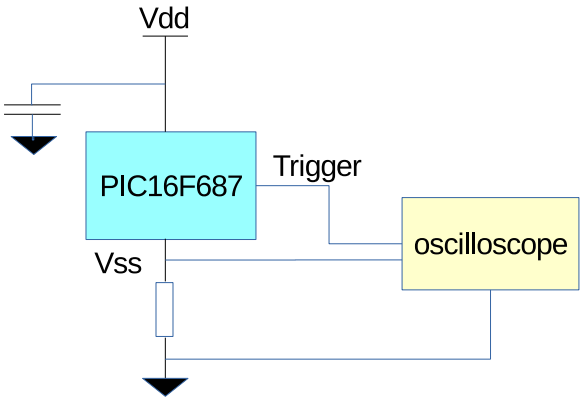

Fig. 1. Measurement setup

take two instruction cycles if a branch is taken. Each instruction execution is overlapped with the next instruction fetch. The working register is one of the two operands of the ALU. There is a 128-byte register file including general purpose registers and special function registers (SFRs).

Because there are so many factors that may affect power consumption, an ad hoc experiment will soon become unmanageable. We develop a systematic approach for instruction-level side-channel analysis:

- Build semantic models of the instruction set, using known architecture information.
- Generate random testing code that is long enough to execute each instruction operation many times.
- Calculate runtime status according to semantic models for each instruction.
- Cross-validate power consumption and the semantic models with respect to instruction operations and runtime status.

We use random instructions rather than real code in order to evenly sample the code space, avoiding overfitting to any specific code base. Potential high-order side-channel characteristics that exist only among some particular instruction pairs/blocks will be averaged out when using random code. While we may lose some information for particular instruction blocks, we retain side-channel properties that are applicable to arbitrary programs. It is therefore not necessary to reanalyze every new piece of software made to run on the target IC, as we are able to develop general protection mechanisms. See Section 5 for additional details.

Semantic models. Building semantic models of an instruction set includes elaborating the detailed operations, such as fetch, decode, and data read/write, that happen during an instruction execution. Because the known architecture information is not complete, our semantic models are only assumptions, which can

be cross-validated with the side-channel measurements. This has numerous benefits. First, this is necessary for predicting branches during code generation. Second, analyzing the measurements with respect to the runtime status reveals effects of data versus those of processing. Third, waveforms can also be checked against the predicted runtime status in order to guarantee that the chip functions correctly. This is necessary because using a large shunt resistor (see below), introduces common impedance coupling and narrows the voltage drops between V_{DD} and V_{SS}, whereas a large enough shunt resistor eliminates amplification circuits which may introduce additional noise. Sanity checking of the waveform against the predicted status helps in choosing the right resistor value besides the bandwidth consideration.

Based on the limited architecture information described in the PIC16F687 datasheet [2,3], we deduce that potential data that may appear on buses, and therefore are likely to cause the major power consumption, include values of the program counter (PC), the operands and opcode of instructions, the working register, the selected file register, and the `STATUS` SFR. Then we generate random code traces and calculate bus traffic from instruction semantics.

Power traces are collected following the standard setup for power-based SCA, as shown in Figure 1. The ground pin of the DUT is connected to a 82Ω shunt resistor. Voltage drop across the shunt resistor is captured by the PicoScope 5444B 200MHz USB oscilloscope. The ground pin, instead of the power supply pin, is used due to limitations of the oscilloscope. To mitigate the low-pass filtering effects of the chip itself [28,32], we set the frequency to 125 kHz. The sample rate is 31.3 MS/s. Higher frequency settings suffers more from the low-pass filtering effects and do not work with the oscilloscope. The setup is low-cost and reflects a worse-case scenario from the verifier's perspectives.

To build side-channel models, the verifier needs to have access to the device. For integrity checking, it is reasonable to assume that the verifier has access to the exact DUT, thus to ignore small variations among chips of the same device model resulted from the process technology. In all the following experiments, tests are performed on the same device that is used to build the models.

3.1 Recognizing Operations Versus Recognizing Execution Instances

For our 2K-memory μC, we generate 1435 instructions, which are randomly selected from 29 instruction operations (excluding `CALL`, `RETFIE`, `RETLW`, `RETURN`, `SLEEP`, and `CLRWDT`). Operands are also random.[1] `CALL`, `RETFIE`, `RETLW`, and `RETURN` are manually inserted in multiple places so that the program can execute normally. 1020 power traces are collected, among which 50% are used for modeling and 50% are used for testing. A typical waveform is shown in Figure 2. PIC16F687 has an instruction cycle of four clock cycles, denoted as Q1 to Q4. The waveform exhibits sharp peaks at clock rising/falling edges, showing that

[1] To have enough samples per operation, file register access is limited to 12 general-purpose registers and the `STATUS` SFR.

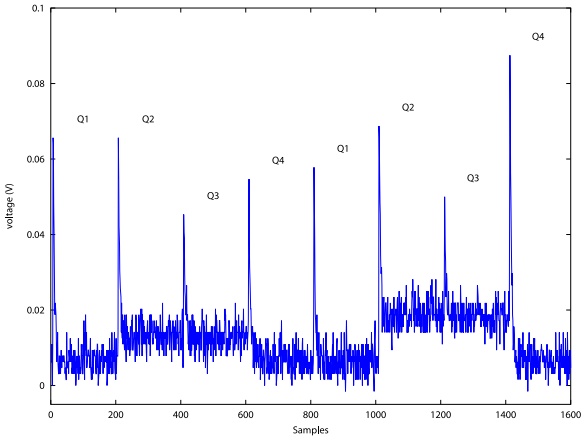

Fig. 2. Sample waveform of executing "`MOVLW 0x69`" and "`ADDWF 0x40,F`"

the low-pass effects are not prominent in our experiment setup. Samples are time-aligned according to peak values.

We first build a model with respect to instruction operations, as in previous research [16,20,31,39]. Given a single trace of power samples of four clock cycles, the verifier tries to recognize one out of 33 instruction operations, a typical pattern recognition/classification problem. We apply various classifiers, including naive Bayes, kNN, SVM, Multilayer Perceptron, etc., together with/without feature selection by PCA, mutual information, and LDA. The best recognition rate is obtained by using template analysis [10,16]. The power consumption is approximated as multi-variate Gaussian signals, which yields very good results in recognition/classification and separability analysis. One template is built for each instruction operation ω_i. When selecting l samples in one instruction cycle for modeling, the templates are l-dimensional Gaussian distributions with parameters estimated from power consumption observations when executing ω_i.

$$p(\boldsymbol{x}|\omega_i) = \frac{1}{(2\pi)^{l/2}|\boldsymbol{\Sigma}_i|^{1/2}} \exp\Big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\Big)$$

$$\boldsymbol{\mu}_i = \frac{1}{N_i}\sum_{j=1}^{N_i} \boldsymbol{x}_{ij}$$

$$\boldsymbol{\Sigma}_i = \frac{1}{N_i - 1}\sum_{j=1}^{N_i} (\boldsymbol{x}_{ij}-\boldsymbol{\mu}_i)(\boldsymbol{x}_{ij}-\boldsymbol{\mu}_i)^T$$

where $\boldsymbol{x}_{ij}$ is an l-dimensional observation of executing operation ω_i in the modeling data, N_i is the number of such observations in the modeling data. When given a new observation $\boldsymbol{x}$, the instruction operation is estimated by applying the Bayes rule, which is the ω_i that gives the maximum a posteriori probability.

$$\hat{\omega} = \underset{\omega_i}{\operatorname{argmax}}\, p(\omega_i|\boldsymbol{x}) = \underset{\omega_i}{\operatorname{argmax}}\, p(\boldsymbol{x}|\omega_i)P(\omega_i)$$

For integrity checking, the a priori distribution $P(\omega_i)$ is meaningless, since the verifier is unlikely to know with which instruction the attacker may use to replace the original code. We therefore assume the a priori distribution is uniform, thus reduce Bayes rule to the maximum likelihood criterion.

$$\hat{\omega} = \underset{\omega_i}{\operatorname{argmax}}\, p(\boldsymbol{x}|\omega_i)$$

One template is built for each operation. For file-register operations, each template is built for writing to the file/working register. In total, 47 templates are built. The resulting average recognition rate is 45.6%, which is comparable to unoptimized results of [10,16] and the single-location result of [39]. While some operations still have acceptable recognition rates, such as `CLRW` (99.0% recognition rate), `GOTO` (97.8%), and `COMF f,F` (95.7%), other operations, such as `CLRF`, `DECFSZ f,W` and `IORWF f,F`, are almost always misclassified.

To explore the sources of recognition errors, we perform the same template analysis but now build one template for each instance of instruction execution. The models thus incorporate power consumption caused by execution with different operands and runtime status. For the same data, we build 1435 templates. Applying again the maximum likelihood criterion, the average recognition rate is surprisingly 99.90%, in contrast with 0.0678% for random guess.

3.2 Separability

The high recognition rate can be explained by the separability of templates. One measure of separability is the Bhattacharyya distance, which is related to the upper bound of the minimum attainable error of the Bayes classifier [41].

$$P_e \leq \epsilon_{CB} = \sqrt{P(\omega_i)P(\omega_j)} \int_{-\infty}^{\infty} \sqrt{p(\boldsymbol{x}|\omega_i)p(\boldsymbol{x}|\omega_j)}d\boldsymbol{x}$$

For multi-variate Gaussian,

$$\epsilon_{CB} = \sqrt{P(\omega_i)P(\omega_j)}\exp(-B_{ij})$$

$$B_{ij} = \frac{1}{8}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^T(\frac{\boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_j}{2})^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j) + \frac{1}{2}\ln\frac{|\frac{\boldsymbol{\Sigma}_i+\boldsymbol{\Sigma}_j}{2}|}{\sqrt{|\boldsymbol{\Sigma}_i||\boldsymbol{\Sigma}_j|}}$$

Building templates for instances of instruction execution, the 30 errors in 30,000 tests correspond to 4 out of 1,028,895 pairs that have the smallest Bhattacharyya distances (from 3.98 to 11.45), showing that the multi-variate Gaussian models are good approximators of the signals. In contrast, for templates of instruction operations, the Bhattacharyya distances of the majority of template pairs, especially logic and arithmetic operations, are near zero, corresponding to recognition rates near to those of random guess.

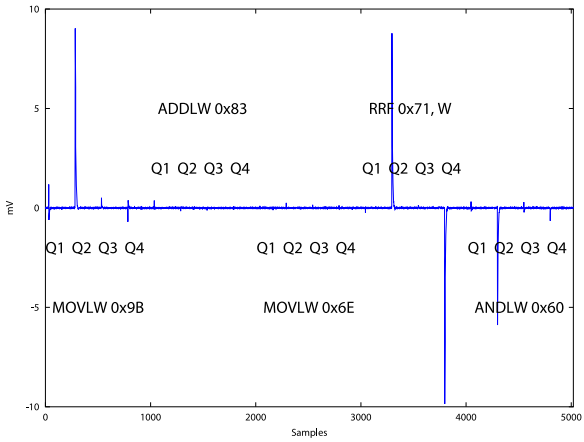

Fig. 3. Difference between two measurements. Although the same program is executing, the register values are different.

4 Data Effects

To discover the effects of runtime status such as bus traffic, we change testing programs by modifying initial values of registers and rerun the measurements. Because register values affect results of conditional branches, code near conditional branches is adjusted, so that only the instruction immediate after each conditional branch test is different while majority of instruction execution stays the same. The difference between the two resulting measurements is shown in Figure 3.

The measurements of "`MOVLW 0x9B`" have significant difference at the edge of Q2. After executing "`MOVLW 0x9B`", the measurements of "`ADDLW 0x83`" and "`MOVLW 0x6E`" are nearly identical. Executing "`RRF 0x71,W`" differs at Q2 and Q4, whereas executing "`ANDLW 0x60`" has significant difference at Q2 and slight difference at Q4. Q1 and Q3 are on the other hand almost the same at all time. This phenomenon coincides with the architecture description in [2]: for instruction execution, instruction is latched in Q1, data memory is read in Q2 (operand read), data is processed in Q3, and in Q4 data memory is written (destination write). After executing "`MOVLW 0x9B`", the working register and the `STATUS` register are the same [2], and the traffic on the data bus during operand read and destination write is therefore the same, which leads to the same side-channel measurements. The contents of the file register `0x71` are different, which results in different traffic on the data bus and accordingly different measurements at Q2 and Q4. The result of "`RRF 0x71,W`" is written to the working register, and thus causes further differences at Q2 and Q4 when executing "`ANDLW 0x60`". On the other hand, Q1 and Q3 do not show heavy data dependency, even though data is processed in Q3.

[2] The `STATUS` register is affected by previous code not shown.

Further analysis shows that there are strong linear relationships between runtime status and side-channel measurements. Let runtime status at time t be a vector of random variables $\boldsymbol{D}$, the power consumption at t be a random variable Y, the linear dependence between Y and $\boldsymbol{D}$ is formulated as

$$Y = \boldsymbol{a}^T \boldsymbol{D} + b$$

where $\boldsymbol{a}$ is a vector of weights, b encloses remaining components in the power consumption at time t including offsets, time-dependent components, and noise, and is assumed independent from other variables [8]. For two random variables $X = \boldsymbol{a}^T \boldsymbol{D}$ and Y, the Pearson correlation coefficient is a measure of linear dependence between X and Y:

$$r = \frac{cov(X, Y)}{\sigma_X \sigma_Y} = \frac{\sigma_X}{\sqrt{\sigma_X^2 + \sigma_b^2}}$$

r tends to ± 1 as σ_b^2 tends to 0. Spearman's rank correlation is the Pearson correlation between weakly-ordered values. The two correlations are identical for values which are monotonically related. Spearman's correlation is more sensitive to outliers.

Analysis shows that the Hamming distance (HD) of PC and (PC+1) influences the peaks in Q1, regardless of operations. This corresponds to the fact that the pipeline depth of the DUT is two: each instruction execution is overlapped with fetching the next instruction, and the PC is incremented in Q1 for instruction fetch.

The Hamming distance of values of operands influences the peaks in Q2. In Q2, different types of operations will load different types of operands. For bit-oriented file register operations and byte-oriented file register operations such as `CLRF`, `MOVWF`, `RRF`, `DECFSZ`, and `BTFSC`, the content of the file register is loaded, even if it will not be used in the computation in Q3 (as in `CLRF` and `MOVWF`). For literal operations, the literal is loaded. The power consumption is proportional to the Hamming distance of the value already on the bus (which is the result of previous instruction execution), and the data loaded for current instruction execution. For vector $\boldsymbol{D} = (HD(\text{old data on bus,new data on bus}))$, the regression coefficient vector $\boldsymbol{a}$ is $(2.87, 44.70)$, in mV. The Pearson correlation coefficient (r) is 0.971, and the Spearman's rank correlation coefficient (ρ) is 0.969.

The plateaus following the peaks in Q2 and Q3, are linear to the Hamming weight (HW) of next opcode, regardless of instruction operations. For vector $\boldsymbol{D} = (HW(\text{next opcode}))$, the coefficient vector is $(0.836, 14.41)$, in mV; $r = 1.000$ and $\rho = 0.991$.

The peaks in Q3 are linear to the Hamming weight of next opcode and the Hamming weight of current opcode: for vector $\boldsymbol{D} = (HW(\text{current opcode}), HW(\text{next opcode}))$, the coefficient vector is $(1.32, 0.828, 28.43)$, in mV; $r = 0.998$ and $\rho = 0.998$.

The peaks in Q4 is linear to the Hamming distance of values on the data bus and the Hamming weight of next opcode: For literal operations, vector is

(HD(literal, new value of working register), HW(next opcode)), coefficient vector is $(3.10, 2.19, 34.36)$, in mV; $r = 0.992$ and $\rho = 0.987$. For operations with the working register as destination, the vector is (HD(value of file register, new value of working register), HW(next opcode)), coefficient vector is $(3.00, 2.17, 35.29)$, $r = 0.998$ and $\rho = 0.997$. For operations with the file register as destination, the vector is (HD(old value of file register, new value of file register), HW(next opcode)), coefficient vector is $(3.60, 2.15, 36.22)$, $r = 0.998$ and $\rho = 0.998$. The Hamming distance of values of the `STATUS` SFR surprisingly does not significantly affect the power consumption in Q4, which is reflected by the fact that its regression coefficient is one order lower than those of other variables.

The relationships reveal several valuable sources of side-channel leakage that can be utilized for different verification purposes. First, they reveal that side-channel measurements have strong dependencies on data and weak dependencies on instruction operations. This explains why templates of logic and arithmetic operations have small Bhattacharyya distances: they have small differences in the Hamming weights of their opcode spaces and the distributions of operands and results (except for `COMF`, whose Q4 always has large power consumption since its (HD(old value of file register, new value of working/file register)) is always 8). While not helping in template analysis with respect to instruction operations, data dependencies in peaks of Q2 and Q4 help to match data values with operations. Second, the strong linear relationships also help to validate our semantic models. Third, the dependency in opcodes through Q2 to Q4 leaks information about the control flow. While not directly revealing the neighboring opcodes, this helps in identifying some instructions such as `NOP` (having the unique 0 opcode) and the `NOP` executed after each branch. Fourth, coefficient vectors of the order of mV per bit, given the dynamic range of the measurements of $(15, 70)$ mV, are resilient to noise in simple (versus differential) power analysis.

To increase the potential SNR of operation-related signals, we generate testing programs composed of code of the same Hamming weight. Except `GOTO` and instruction types that cannot have targeted Hamming weight (e.g. `NOP` and `CLRW`), all logic and arithmetic operations are included. Repeating the experiment, previous conclusions on data dependencies still hold. Q3 has nearly the same value, which can be shown by the small standard deviations (σ) among waveforms. For execution instances, the maximum σ, occurring at the peak of Q3, is 0.324 mV, in contrast with the maximum σ in previous experiments, which is 4.193. For instruction operations, the maximum σ is 0.121, in contrast with the maximum σ in previous experiments, which is 2.495. This implies that Q3 does not yield sufficient margins for classification. Applying various pattern recognition techniques, the best average recognition rate is 33.16% for instruction operations, obtained by SVM with polynomial kernel, five-fold cross-validation. The recognition rate is still much worse than that obtained by template analysis for instruction execution instances, which is 99.53%.

5 Side-channel Programming

Above experiments show that because of significant data dependency of power consumption, side-channel profiling according to instruction operations is unlikely to have high recognition rates or large margins. It is more suitable to use runtime data status for simple power analysis, especially in noisy environments. On the other hand, although there are very strong linear relationships between waveforms and data read in Q2 and destination write in Q4, it is the Hamming distance rather than the exact data that is involved. For a side-channel-aware attacker, it is easy to compute data pairs that have the same Hamming distance with previous data on the bus in Q2, go through different operations, and again have the same Hamming distance with the operands in Q4, thus evading side-channel-based checking. This is feasible even when considering the Hamming weight relationships through Q2 to Q4, since the opcode is quite compact.

The good news is that a change in data may have cascading effects: in order to tamper with data in one instruction, previous and next instructions must be modified accordingly. The developers of the μC can utilize aforementioned data dependencies to guarantee tamper detection. For a given instruction set, the developers can find a trace of side-channel measurements $\{Q2_i, Q3_i, Q4_i\}, i = 1, \ldots, n$ that for any programs, when given a set of initial register values, lead to a unique set of resulting values. The developers can just choose one program that transforms the input to the output. Any tampering with the program can then either be detected from side-channel measurements, or lead to the same unique set of resulting values.

For a simple example, let us confine the instruction set to include only the literal operations {`ADDLW`, `ANDLW`, `IORLW`, `XORLW`} that perform add/bit-and/bit-or/bit-XOR operations with the working register and a literal, and then write results to the working register. The runtime status that has strong linear relationships with the side-channel measurements includes (HD(old working register,literal)) in Q2, (HD(new working register,literal)) in Q4, and (HW(current opcode)) and (HW(next opcode)) through Q2 to Q4. It is possible to find that given the initial value of the working register `0x55`, execution of any four-instruction programs leads to the same resulting working register `0x1F`, given the runtime status constraints (HD(old working register,literal)) = $[3, 6, 3, 7]$ for Q2's of the four instructions respectively; (HW(current opcode)) = $[10, 10, 9, 11]$ for Q3's respectively (also for previous Q2, Q3, Q4); (HD(new working register,literal)) = $[5, 4, 2, 1]$ for Q4's respectively. There are two programs of four instructions that satisfy such side-channel constraints: {`ADDLW 0x7C`, `ANDLW 0x3F`, `XORLW 0xF1`, `ADDLW 0x3F`} and {`ADDLW 0x1F`, `ANDLW 0x9F`, `XORLW 0xF4`, `ADDLW 0x3F`}, but all lead to the same resulting working register `0x1F`. The developers can randomly pick one of the programs, say, {`ADDLW 0x7C`, `ANDLW 0x3F`, `XORLW 0xF1`, `ADDLW 0x3F`}. Even if an attacker is able to profile the side-channel characteristics of the μC, she can at best modify the code into {`ADDLW 0x1F`, `ANDLW 0x9F`, `XORLW 0xF4`, `ADDLW 0x3F`}, which results in exactly the same value and thus renders the attack meaningless, since other modifications will violate the side-channel constraints.

Data: $\forall q2 = [q2(1), \ldots, q2(n)], q3 = [q3(1), \ldots, q3(n)], q4 = [q4(1), \ldots, q4(n)], W_0, F$
Result: $w, num, op, opr, w1p, w1$

```
genPrgm(q2, q3, q4, W_0, F)
begin
    w(1) ← {W_0}
    for i = 1, ..., n do
        for w_0 ∈ w(i) do
            Opr ← {x | HD(x, w_0) = q2(i)}
            for f ∈ F do
                for x ∈ Opr do
                    y ← f(w_0, x)
                    if HW(x) + HW(opcode of f) = q3(i) and HD(x, y) = q4(i)
                    then
                        num(i) ← num(i) + 1
                        op(i, num(i)) ← f
                        opr(i, num(i)) ← x
                        w1p(i, num(i)) ← w_0
                        w1(i, num(i)) ← y
                        w(i + 1) ← w(i + 1) ∪ {y}
```

Algorithm 1. `genPrgm`: Compute programs that satisfy a given side-channel trace

Data: $\forall Q2 = \{q2_i, i = 1, \ldots, M_2\}, Q3 = \{q3_i, i = 1, \ldots, M_3\}, Q4 = \{q4_i, i = 1, \ldots, M_4\}, W_0, F$
Result: $ops, oprs, q2, q3, q4, W_1$

```
genSCP(Q2, Q3, Q4, W_0, F)
begin
    for q2 ∈ Q2 do
        for q3 ∈ Q3 do
            for q4 ∈ Q4 do
                [w, num, op, opr, w1p, w1] ←genPrgm(q2,q3,q4,W_0,F)
                if w(n + 1) is Singleton then
                    W_1 ← w(n + 1)
                    ops, oprs ← backtrace op and opr through w1 and w1p
                    break to top
```

Algorithm 2. `genSCP`: Compute combinations of programs and side-channel traces that produce unique output

This leads to the idea of "side-channel programming", in which software engineers utilize side-channel characteristics of existing hardware during development to guarantee tamper detection. General algorithms for finding such combinations of side channel constraints and programs of any length are shown in Algorithms 1 and 2, where W_0 is the initial value of the working register, F is a set of functions that simulate the operations in the instruction set, W_1 is the resulting value of the working register, and $(ops, oprs)$ compose programs that satisfy a n-long side-channel trace $\{q2, q3, q4\}$ and also output a unique W_1. For the above small instruction set, it takes just seconds to find side-channel traces that guarantee unique transformations of input and output on a commercial PC. As the instruction set increases, complexity increases. We leave it as future work to efficiently perform "side-channel programming" for the full instruction set.

6 Conclusion and Future Work

For simple power analysis, we explore whether or not instructions can lead to consistently distinguishable side-channel information, and if so, how the side-channel characteristics differ. By building semantic models of the instruction set and cross-validating with side-channel measurements, we show that data dependencies, rather than instruction operation dependencies, are dominant. We reveal strong linear relationships between runtime status and side-channel measurements, which enable "side-channel programming" that utilizes side-channel characteristics of existing hardware in software development to provide external verification of software integrity. We show how to generate combinations of side-channel constraints and programs of any length for a subset of the instruction set that guarantee a unique transformation of a given input, so that any tampering with the program by a side-channel-aware attacker can either be detected, or lead to the same unique set of input and output. Our future work involves side-channel programming for the full instruction set.

Side-channel characteristics are determined by the IC design of the DUT. The comparatively small instruction set and simple architecture of the μC under test greatly ease side-channel analysis so that the problem is tractable in reasonable period of time. As more complex IC designs are used in embedded systems, instruction sets support more operations executed in a variable number of instruction cycles, pipelines get deeper and more complex, and more components function in parallel. And therefore significantly more factors will need to be incorporated into the semantic models. It may not be possible to find side-channel characteristics for more complex ICs as succinct as discovered in this work for a simple IC. It is, however, fundamentally possible to derive side-channel characteristics as long as the IC operates deterministically. Our future work also involves applying the proposed approach to other microcontrollers/microprocessors.

Acknowledgments. This work was partially funded by NIH grant 1U01EB012470 and NSF grants CNS 1224007, CNS 1239543, and CNS 1253930.

References

1. grsecurity. http://grsecurity.net/
2. PIC16F631/677/685/687/689/690 data sheet. Microchip Technology Inc. (2008). http://ww1.microchip.com/downloads/en/DeviceDoc/41262E.pdf
3. PICmicro mid-range MCU family - reference manual. Microchip Technology Inc. (1997). http://ww1.microchip.com/downloads/en/DeviceDoc/31000a.pdf
4. Trusted computing group (TCG). TPM 2.0 library specification (2014). http://www.trustedcomputinggroup.org/resources/tpm_library_specification
5. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM side-channel(s). In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 29–45. Springer, Heidelberg (2003)
6. Agrawal, D., Baktir, S., Karakoyunlu, D., Rohatgi, P., Sunar, B.: Trojan detection using IC fingerprinting. In: Proceedings of the IEEE Symposium on Security and Privacy, S&P (2007)
7. Bletsch, T., Jiang, X., Freeh, V.W., Liang, Z.: Jump-oriented programming: a new class of code-reuse attack. In: Proceedings of the ACM Symposium on Information, Computer and Communications Security, ASIACCS (2011)
8. 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)
9. Butterworth, J., Kallenberg, C., Kovah, X., Herzog, A.: BIOS chronomancy: fixing the core root of trust for measurement. In: Proceedings of the ACM Conference on Computer and Communications Security, CCS (2013)
10. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)
11. 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: Proceedings of the Conference on Electronic Voting Technology/Workshop on Trustworthy Elections, EVT/WOTE (2009)
12. 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: Proceedings of the USENIX Conference on Safety, Security, Privacy and Interoperability of Health Information Technologies, HealthTech, p. 9. USENIX Association, Berkeley (2013)
13. Cui, A., Costello, M., Stolfo, S.: When firmware modifications attack: a case study of embedded exploitation. In: NDSS (2013)
14. Davi, L., Sadeghi, A.-R., Lehmann, D., Monrose, F.: Stitching the gadgets: on the ineffectiveness of coarse-grained control-flow integrity protection. In: Proceedings of the USENIX Security Symposium, SEC (2014)
15. Duflot, L., Perez, Y.-A., Morin, B.: What If you can't trust your network card? In: Sommer, R., Balzarotti, D., Maier, G. (eds.) RAID 2011. LNCS, vol. 6961, pp. 378–397. Springer, Heidelberg (2011)
16. 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)
17. Falliere, N., Murchu, L.O., Chien, E.: W32. stuxnet dossier version 1.4 (2011)

18. Francillon, A., Castelluccia, C.: Code injection attacks on harvard-architecture devices. In: Proceedings of the ACM Conference on Computer and Communications Security, CCS (2008)
19. Göktas, E., Athanasopoulos, E., Bos, H., Portokalidis, G.: Out of control: overcoming control-flow integrity. In: Proceedings of the IEEE Symposium on Security and Privacy, S&P (2014)
20. Goldack, M.: Side-channel based reverse engineering for microcontrollers. Master's thesis, Ruhr-Universität Bochum, Germany (2008)
21. Gu, L., Ding, X., Deng, R.H., Xie, B., Mei, H.: Remote attestation on program execution. In: Proceedings of the ACM Workshop on Scalable Trusted Computing, STC (2008)
22. Guo, S., Zhao, X., Zhang, F., Wang, T., Shi, Z., Standaert, F.-X., Ma, C.: Exploiting the incomplete diffusion feature: A specialized analytical side-channel attack against the AES and its application to microcontroller implementations. IEEE Transactions on Information Forensics and Security **9**(6) (2014)
23. Hanna, S., Rolles, R., Molina-Markham, A., Poosankam, P., Fu, K., Song, D.: Take two software updates and see me in the morning: the case for software security evaluations of medical devices. In: Proceedings of the USENIX Conference on Health Security and Privacy, HealthSec (2011)
24. Jin, Y., Makris, Y.: Hardware trojan detection using path delay fingerprint. In: Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, HST (2008)
25. Nohl, K., Krißler, S., Lell, J.: BadUSB - on accessories that turn evil. https://srlabs.de/blog/wp-content/uploads/2014/07/SRLabs-BadUSB-BlackHat-v1.pdf
26. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
27. Li, Y., McCune, J.M., Perrig, A.: VIPER: verifying the integrity of PERipherals' firmware. In: Proceedings of the ACM Conference on Computer and Communications Security, CCS (2011)
28. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards, 1st edn. Springer Publishing Company, Incorporated (2010)
29. Mason, J., Small, S., Monrose, F., MacManus, G.: English shellcode. In: Proceedings of the ACM Conference on Computer and Communications Security, CCS (2009)
30. McCune, J.M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V., Perrig, A.: TrustVisor: efficient TCB reduction and attestation. In: Proceedings of the IEEE Symposium on Security and Privacy, S&P (2010)
31. 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, Heidelberg (2014)
32. Nakutis, Z.: Embedded systems power consumption measurement methods overview (2009)
33. Rodríguez, F., Serrano, J.J.: Control flow error checking with ISIS. In: Yang, L.T., Zhou, X., Zhao, W., Wu, Z., Zhu, Y., Lin, M. (eds.) ICESS 2005. LNCS, vol. 3820, pp. 659–670. Springer, Heidelberg (2005)
34. Seshadri, A., Perrig, A., Doorn, L.V., Khosla, P.: SWATT: software-based ATTestation for embedded devices. In: Proceedings of the IEEE Symposium on Security and Privacy, S&P (2004)
35. Skorobogatov, S., Woods, C.: Breakthrough silicon scanning discovers backdoor in military chip. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 23–40. Springer, Heidelberg (2012)

36. Soll, O., Korak, T., Muehlberghuber, M., Hutter, M.: EM-based detection of hardware trojans on FPGAs. In: IEEE International Symposium on Hardware-Oriented Security and Trust, HOST, May 2014
37. Song, P., Stellari, F., Pfeiffer, D., Culp, J., Weger, A., Bonnoit, A., Wisnieff, B., Taubenblatt, M.: MARVEL: malicious alteration recognition and verification by emission of light. In: IEEE International Symposium on Hardware-Oriented Security and Trust, HOST (2011)
38. Stajano, F., Anderson, R.: The grenade timer: fortifying the watchdog timer against malicious mobile code. In: Proceedings of International Workshop on Mobile Multimedia Communications, MoMuC (2000)
39. Strobel, D., Oswald, D., Richter, B., Schellenberg, F., Paar, C.: Microcontrollers as (in)security devices for pervasive computing applications. Proceedings of the IEEE **102**(8) (2014)
40. Sugawara, T., Suzuki, D., Saeki, M., Shiozaki, M., Fujino, T.: On measurable side-channel leaks inside ASIC design primitives. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 159–178. Springer, Heidelberg (2013)
41. Theodoridis, S., Koutroumbas, K.. Pattern Recognition, 4th edn. Academic Press (2008)
42. Xu, R., Saïdi, H., Anderson, R.: Aurasium: practical policy enforcement for android applications. In: Proceedings of the USENIX Security Symposium, SEC (2012)
43. 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. Sen. Netw. **11**(2), 33:1–33:33 (2014)
44. Zhang, F., Wang, H., Leach, K., Stavrou, A.: A framework to secure peripherals at runtime. In: Kutyłowski, M., Vaidya, J. (eds.) ESORICS 2014, Part I. LNCS, vol. 8712, pp. 219–238. Springer, Heidelberg (2014)
45. Zhou, Z., Gligor, V.D., Newsome, J., McCune, J.M.: Building verifiable trusted path on commodity x86 computers. In: Proceedings of the IEEE Symposium on Security and Privacy, S&P (2012)