Real-time Detection and Adaptive Mitigation of Power-based Side-Channel Leakage in SoC

Pantea Kiaei*, Yuan Yao†, and Patrick Schaumont*
* Worcester Polytechnic Institute, Worcester, MA 01609 USA
† Virginia Tech, Blacksburg, VA 24061 USA

***Abstract*—Power-based side-channel is a serious security threat to the System on Chip (SoC). The secret information is leaked from the power profile of the system while a cryptographic algorithm is running. The mitigation requires efforts from both the software level and hardware level. Currently, there is no comprehensive solution that can guarantee the whole complex system is free of leakage and can generically protect all cryptographic algorithms. In this paper, we propose a real-time leakage detection and mitigation system which enables the system to monitor the side-channel leakage effects of the hardware. Our proposed system has extensions that provide a real-time monitor of power consumption, detection of side-channel leakage, and real-time adaptive mitigation of detected side-channel leakage. Our proposed system is generic and can protect any algorithm running on it.**

***Index Terms*—real-time leakage detection, power side-channel leakage, adaptive countermeasure**

I. Introduction

Side-channel leakage (SCL) discloses secret information through power consumption, electromagnetic dissipation, execution time, and other sources. In this work, our focus is on power-based SCL. Power side-channel analysis (SCA) has made the hardware attacker a particularly powerful adversary, relevant to embedded applications of secure System-on-Chips (SoC). When an SoC handles secret values, such as secret keys used in a symmetric cryptographic algorithm, the physical side-effects of these computations may be exploited as SCL. SCL is a critical vulnerability of secure SoCs as sophisticated SCA enables attackers to learn about the secret information even when the cryptographic algorithm is computationally secure.

When a program is running on a processor, unexpected SCL can happen. A software program is optimized and transformed into an executable file by a compiler. Generally, compilers are designed to optimize the speed or memory footprint of code, however, that is not always aligned with critical software security concerns. For example, in a masked implementation, compiler tasks such as register allocation and strength reduction (leading to shift operations), might cause unintended unmasking. Furthermore, even the existing algorithms for secure software design, like probing secure multiplication (ISW) [1] or bounded-moment secure multiplication [2], have been shown to have leakage caused by unintended hardware effects [3], [4]. While the software implementation causes the side-channel leakage, it is the processor hardware that creates the physical effects of SCL. Therefore, even if the software includes countermeasures against SCL, it is very hard to guarantee that the underlying hardware will be leakage free while running the software program.

A processor consists of many elements including memory units and pipeline stages. Power SCL can arise from several parts of the micro-architecture simultaneously at times and individually at others. Therefore it is challenging to pin-point the exact part of the micro-architecture causing this leakage. Current research efforts attempt to mitigate side-channel leakage from two ends. From the software side, researchers attempt to develop side-channel resistant software. This includes secure programming mechanisms [2], [5], and leakage-mitigating code-generation techniques [6]. From the hardware side, the hardware designers focus on building side-channel resistant processor designs and instruction set extensions to eliminate the dependencies of power consumption on the secret values and prevent accidental unmasking in masked software programs [7], [8], or building up design-time (pre-silicon) side-channel evaluation [9], [10]. Software-only and hardware-only approaches are sub-optimal. The software designer will have to apply countermeasure to a substantial part of the code, if not the entire code, including the leaking section while being oblivious to the detailed design of the hardware. The processor hardware designer will have to build a completely new processor with corresponding countermeasures applied [11]. These approaches cause a large overhead in software execution time and code size, or in hardware area overhead and performance.

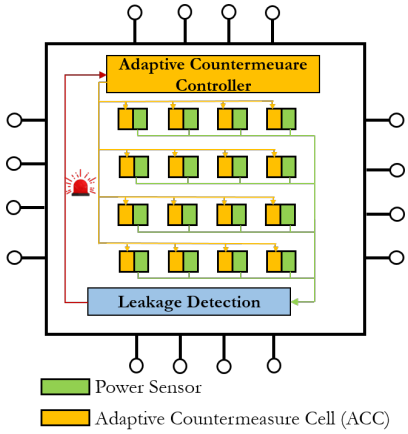

Fig. 1: Proposed mechanism. Power sensors monitor the power. Leakage detection module predicts if a power SCL is happening based on the monitored power and alarms the adaptive controller. Adaptive countermeasure controller enables local countermeasures through ACCs.

We propose to enable the processor system to monitor the side-channel leakage. We introduce a sensor that detects leakage in real-time and that triggers adaptive countermeasures. We present a proactive security mechanism based on hardware extensions.The extensions provide a real-time monitor of the power consumption (using sensor cells in Fig. 1), detection of side-channel leakage (using leakage detection cells in Fig. 1), and real-time adaptive mitigation of detected side-channel leakage (using adaptive countermeasure cells (ACC) in Fig. 1). When the system is running, the proposed security mechanism (implemented as sensors) continuously monitors the leakage status of the system. Once a leakage

E-mails: pkiaei@wpi.edu; yuan9@vt.edu; pschaumont@wpi.edu

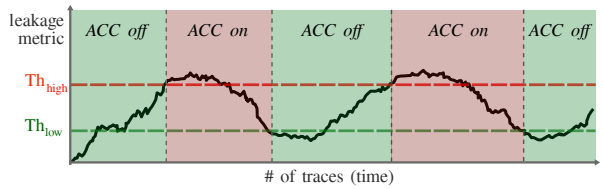

Fig. 2: Each ACC turns on as soon as the leakage metric corresponding to its nearby power sensor passes the high threshold (Th_{high}). Once the leakage decrements to lower than Th_{low}, ACC will turn off.

is detected by the implemented sensors, the alarm signal will be set. This alarm signal has two functions: the alarm points out the precise leakage source in the hardware, and it triggers the adaptive countermeasure to mitigate the leakage.

II. System Design

We propose to build an insight into tracking, identifying, and mitigating side-channel leakage, while maintaining minimal implementation overhead. Using this methodology, the underlying hardware (processor) provides protection against power SCA for any program running on it. In our solution, then, the software programmer can be unaware of the power consumption of the hardware; the programmer may write code and generate the assembly using off-the-shelf compilers. The design time is therefore dramatically reduced, and the existing compiler optimizations are applied to provide a performance boost while ensuring resistance to power analysis attacks.

The challenges facing the implementation of such a system are three-fold: detecting leakage, finding the source of leakage, and dynamically eliminating leakage. To address these challenges, our proposed real-time leakage detection system contains the following modules:

a) In situ power measurement: Traditionally, power measurements were applied to the prototype of the chip. To achieve real-time leakage detection, we will implement an on-chip in situ power measurement based on power modeling. We also intend to study the possibility of utilizing existing on-chip sensors for the same purpose.

b) Leakage detection: After getting the real-time in situ power measurement, we will come up with the leakage evaluation mechanism and implement the corresponding hardware extension within the processor to achieve leakage detection. The key challenge in leakage detection is building a suitable leakage evaluation metric that fits well within the needs of real-time leakage detection and can be used for general-purpose programs. TVLA [12] is currently the most popular leakage assessment technique, however, it requires categorization of inputs into fixed and random sets and therefore cannot be directly applied to our real-time leakage detection. Possible leakage metrics include power distribution-based characterization or machine learning-based metrics.

c) Adaptive countermeasure: Our goal is to keep the leakage probability low; start the protection once the probability of leakage is higher than a threshold and stop it when the probability is lower than another (as shown in Fig. 2). This has the advantage that while the design is kept secure, only the necessary parts of the design are protected.

III. Preliminary Implementation

As the first step to build up the whole real-time system, we implemented the power sensors as hardware extensions to an SoC with a RISC-V processor and fabricated the chip with the CMOS 180nm technology for further testing. We chose digital Ring-Oscillators (RO) as our power sensors to be able to mirror the changes in the power consumption of the chip in through their oscillating frequency. The power sensors are added as co-processors in the SoC to communicate with the processor. In the chip layout, as shown in Fig.3, the power sensors are evenly distributed throughout the design to monitor the local power consumption in real-time while the chip is running a program.

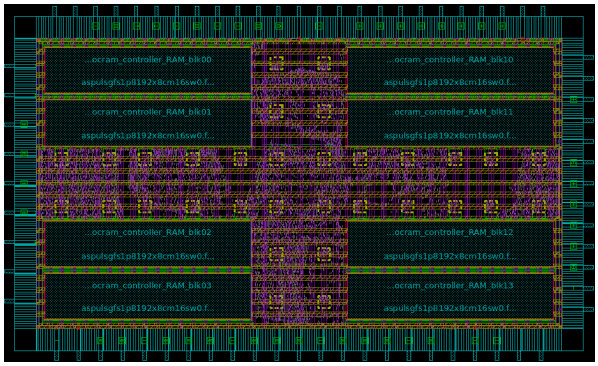

Fig. 3: Layout of the chip, integrating the power sensors scattered throughout the layout. The yellow dashed squares highlight the power sensors.

IV. Conclusion and Future Work

We believe that our work is the first to build a mechanism that provides security to any algorithm and *dynamically* applies corresponding countermeasures. The presented system is expected to be very generic; once we have an implementation of such a system, we can securely run any algorithm on it. This is also the case for a protected processor implementation. Whereas, in a software protection realm, any new algorithm would need its own security implementation from the ground up, hence having a low genericness. All these advantages of our proposed method come at the cost of silicon area overhead which is still expected to be lower than that of a protected processor implementation.

References

[1] Ishai et al., "Private circuits: Securing hardware against probing attacks", in *Annual International Cryptology Conference*. Springer, 2003, pp. 463–481.

[2] Barthe et al., "Parallel implementations of masking schemes and the bounded moment leakage model", in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 535–566.

[3] Balasch et al., "On the cost of lazy engineering for masked software implementations", in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2014, pp. 64–81.

[4] Grégoire et al., "Vectorizing higher-order masking", in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2018, pp. 23–43.

[5] Goudarzi et al., "Secure multiplication for bitslice higher-order masking: Optimisation and comparison", in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2018, pp. 3–22.

[6] Wang et al., "Mitigating power side channels during compilation", in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 590–601.

[7] Kiaei et al., "Custom instruction support for modular defense against side-channel and fault attacks", in *Constructive Side-Channel Analysis and Secure Design - 11th International Workshop, COSADE 2020, Lugano, Switzerland, April 1-3, 2020, Proceedings*.

[8] Kiaei et al., "Domain-oriented masked instruction set architecture for risc-v.", *IACR Cryptol. ePrint Arch.*, vol. 2020, pp. 465, 2020.

[9] Yao et al., "Architecture correlation analysis (aca): Identifying the source of side-channel leakage at gate-level", in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2020, pp. 188–196.

[10] Yao et al., "Verification of power-based side-channel leakage through simulation", in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2020, pp. 1112–1115.

[11] De Mulder et al., "Protecting risc-v against side-channel attacks", in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–4.

[12] Becker et al., "Test vector leakage assessment (tvla) methodology in practice", in *International Cryptographic Module Conference*, 2013, vol. 1001, p. 13.