Take It From Cryptography: Lessons to Achieve Broad Side Channel Protection

Christopher W. Fletcher; cwfletch@illinois.edu; UIUC

Building high-performance yet secure hardware systems creates a paradox for designers. On one hand, high-performance systems inherently exploit a program's sensitive data-dependent behavior to get that performance. On the other hand, adapting program execution based on sensitive data inherently reveals information about that data in the form of side channels. These side channels are insidious. Data-dependent behavior from virtual memory accesses [16] to hardware memory accesses [12, 17] to branch predictor [1] and arithmetic pipeline [2] state can be observed through hardware resource usage, power analysis [8] and electromagnetic emissions [11].

In parallel to hardware security, there is a rich literature in encrypted computation that defeats *all* side channels (e.g., [6, 7, 9]). It is important to ask: what properties of these schemes give rise to their broad security guarantees? We will take Homomorphic Encryption [7] (HE) as an example. In HE, each arithmetic operation is performed on ciphertexts directly (e.g., $\mathsf{Enc}_K(x) \oplus \mathsf{Enc}_K(y) \implies \mathsf{Enc}_K(x+y)$). Data is never decrypted, thus each unit of arithmetic is clearly side channel-free (modulo a cryptographic break). Further, since all data is encrypted, composing a larger program out of arithmetic necessitates running the program in a "straight-line code" fashion. Ergo, composition of arithmetic is input data-independent and also side channel-free. Together, easy-to-understand security of arithmetic plus easy-to-understand composition of arithmetic allows us to write all-encompassing security definitions, e.g., "under the Learning with Errors assumption, the evaluation of an arbitrary program P, between any two inputs x and $\overline{x}$, is computationally indistinguishable to any polynomial time adversary."

This white paper argues that systems should adopt holistic techniques, inspired by ideas from applied and pure cryptography (like those given above), to block broad classes of side channels. In particular, we want to mimic the following characteristics found in cryptographic schemes. First, that whole-program security reduces to the security of individual components and to the data-independent scheduling of components. This makes it easier to argue security for arbitrary programs, compositions of programs, etc. Second, that it is possible to write formal definitions of whole-program security that protect against powerful adversaries. The rest of the paper gives an overview of our recent work on the Ascend secure processor, as an example system built in this line of thinking.

Ascend is a co-processor that was designed to emulate Homomorphic Encryption in hardware. The idea is, by tolerating a hardware TCB we can avoid using cryptography (yielding a large speedup) *if* we can suitably obfuscate the program's execution from the adversary's vantage point. In the Ascend project, we assumed a powerful adversary with arbitrary access to the Ascend processor's external pins—which carry digital (e.g., requests to main memory) and analog (e.g., power draw) signals. Our goal was to meet the following security definition:

Given an arbitrary batch program P, a public length of time T and two arbitrary inputs to P namely x and $\overline{x}$: running $P(x)$ for T time is indistinguishable from running $P(\overline{x})$ for T time from the perspective of the Ascend chip's power and IO pins.

By batch program, we mean a program that takes inputs and computes a result, where the only external requests made are to external memory (i.e., to spill a large working set). The main takeaway is indistinguishability from *any* adversary monitoring the pins, given *any* program P. Ascend can tolerate an intentionally malicious program, or a program that utilizes any combination of hardware resources (e.g., cache, DRAM) in its honest execution. Further, Ascend was designed to defend against two broad classes of side channel: power draw and interactions with external memory.

Main Design Technique. Ascend uses a two-level approach, inspired by Homomorphic Encryption (above) to protect both digital and analog channels. First, we develop primitives that are able to perform atoms of work (i.e., a unit of arithmetic from HE; an SRAM lookup or ALU op in hardware) securely with respect to each side channel. Second, hardware mechanisms on the Ascend chip schedule the atoms of work in a data-independent fashion (e.g., by scheduling each at data-independent rates and performing dummy work if units are not needed when they are scheduled). The next two sections in the paper have details for each primitive (power analysis-resistant logic for CPU primitive operations and Oblivious RAM for off-chip memory).

Primitives for Analog Side Channel Protection. Ascend protects basic CPU operations (e.g., ALU ops, SRAM lookups) by architecting select hardware in optimized Wave Dynamic Differential logic [15] (WDDL) and dual-rail SRAMs [3]. These primitives decouple the time-series power signature of hardware units from inputs to those units (i.e. the adversary learns that an SRAM access occurred to a particular monolithic SRAM, but not the address sent to the SRAM). Traditional WDDL quadruples flip-flop area and doubles cycle latency for feed-forward hardware pipelines. Our optimizations reduce the overhead to doubling flip-flop area and remove the pipeline latency overhead [5].

Primitives for Digital Side Channel Protection. Ascend protects each external memory access using a hardware memory controller that implements an Oblivious RAM (ORAM) protocol [13], specifically an optimized version of Path ORAM [14]. ORAM guarantees that any sequence of memory requests (reads and writes) are indistinguishable from another sequence of the same length. Thus, ORAM hides the request type (read or write), the request data (as if it were a write) and importantly the request address.

Opportunities for Optimization. Ascend performs data-independent work using hardened primitives, thus a key question is performance overhead. To start, we point out that each primitive (WDDL, ORAM, see above) can be (and is) optimized piece-meal [5, 13]. As a representative result, our optimized hardware ORAM evaluated alone results in an average 4X program slowdown across SPEC workloads—about the cost of running an interpreted program. To optimize work scheduling, we can further take advantage of the fact that programs run in Ascend are public. (A follow-on work studies what performance optimizations can be done given a private program [10].) Given a public program, an honest-but-curious server can optimize program execution by scheduling work based on expected common-case program behavior [5], or by adapting to the program's execution at a course-grain if the user is willing to tolerate small leakage [4].

References

[1] Onur Aciicmez, Jean-Pierre Seifert, and Cetin Kaya Koc. Predicting secret keys via branch prediction. Cryptology ePrint Archive, Report 2006/288, 2006. https://eprint.iacr.org/2006/288.

[2] Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. On subnormal floating point and abnormal timing. In *SP'15*.

[3] E. Arikan and A. Ataman. A new power analysis resistant sram cell. In *International Conference on ELECO 2009.*, nov. 2009.

[4] Christopher Fletcher, Ling Ren, Xiangyao Yu, Marten Van Dijk, Omer Khan, and Srinivas Devadas. Suppressing the oblivious ram timing channel while making information leakage and program efficiency trade-offs. In *HPCA*, 2014.

[5] Christopher W. Fletcher. Ascend: An architecture for performing secure computation on encrypted data. In *MIT CSAIL CSG Technical Memo 508 (Master's thesis)*, April 2013.

[6] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. Cryptology ePrint Archive, Report 2013/451, 2013. http://eprint.iacr.org/2013/451.

[7] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[8] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK, 1999. Springer-Verlag.

[9] Yehuda Lindell and Benny Pinkas. A proof of yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004. http://eprint.iacr.org/2004/175.

[10] Kartik Nayak, Christopher W. Fletcher, Ling Ren, Nishanth Chandran, Satya Lokam, Elaine Shik, and Vipul Goyal. Hop: Hardware makes obfuscation practical. In *NDSS'17*.

[11] Alireza Nazari, Nader Sehatbakhsh, Monjur Alam, Alenka Zajic, and Milos Prvulovic. Eddie: Em-based detection of deviations in program execution. In *ISCA'17*.

[12] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of aes. In *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, CT-RSA'06, pages 1–20, Berlin, Heidelberg, 2006. Springer-Verlag.

[13] Ling Ren, Christopher Fletcher, Albert Kwon, Marten van Dijk, and Srinivas Devadas. Design and implementation of the ascend secure processor. In *TDSC*, 2017.

[14] Emil Stefanov, Marten van Dijk, Elaine Shi, T-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: An extremely simple oblivious ram protocol.

[15] K. Tiri and I. Verbauwhede. A logic level design methodology for a secure dpa resistant asic or fpga implementation. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 1, pages 246–251 Vol.1, Feb 2004.

[16] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. *CoRR*, abs/1705.07289, 2017.

[17] Y. Wang, A. Ferraiuolo, and G. E. Suh. Timing channel protection for a shared memory controller. In *HPCA'14*.