Hardware is the new software

Andrew Baumann, *Microsoft Research*

Abstract

Moore's Law may be slowing, but, perhaps as a result, other measures of processor complexity are only accelerating. In recent years, Intel's architects have turned to an alphabet soup of instruction set extensions such as MPX, SGX, MPK, and CET as a way to sell CPUs through new security features. Unlike prior extensions, which mostly focused on accelerating user-mode data processing, these new features exhibit complex interactions and give system designers plenty to think about.

This calls for a rethink of how we approach the instruction set. In this paper we highlight some of the challenges arising from recent security-focused extensions, and speculate about the longer-term implications.

1 Introduction

An instruction set architecture (ISA) is the key interface between the lowest-levels of software and the CPU. The x86 ISA is a complex but enduring set of semantics for instructions, registers, memory, and core devices that must be respected by CPUs, emulators and virtual machines, and all the software that runs on top. Successful ISAs grow over time, and x86 is no stranger to growth given its age and popularity. However, the last two years have seen a dramatic and rapid increase in its complexity (e.g., seen in the size of the architecture manual in Figure 1), with more extensions on the way [21].

In this paper, we examine the causes of this rapid growth, and speculate about the underlying trends driving it. We make our case concrete with a focus on the Intel x86 architecture and its recent extensions. As a market leader, Intel is often the first to add new features, but we doubt these trends are unique to Intel. As we describe in §2, recent Intel CPUs have introduced a wide range of ISA extensions. Whereas past extensions largely focused on performance improvements through new data-processing instructions (e.g., vector extensions), the recent additions are primarily motivated by security concerns, such as defending unsafe C/C++ code against known attacks. These extensions introduce new system-level functionality, often change the semantics of existing instructions, and exhibit complex interactions with other extensions and prior architectural features.

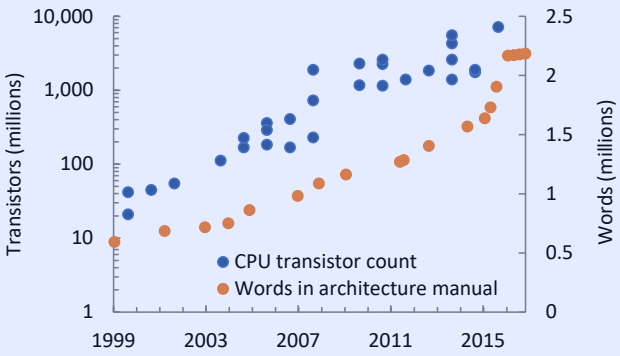

Figure 1: Complexity growth of Intel x86 CPUs and ISA

We take a detailed look at two of the most complex recent extensions: software guard extensions (SGX, §3) and control-flow enforcement technology (CET, §4), before deriving some implications for systems developers and researchers (§5). We argue that these extensions are now approaching software-like levels of complexity, yet carry all the attendant drawbacks of a hardware implementation and the slow deployment cycle that implies. We suspect that the current path may be unsustainable, and posit an alternative future with the ultimate goal of decoupling new ISA features from the underlying hardware.

2 Background: x86 extensions

x86 has long been a complex architecture. The 386 reference manual [18] lists 96 instructions. (Groups of closely-related instructions, such as varying operand widths, are counted as they are documented: usually, as one instruction.) It describes an architecture including segmentation and paging, 16-bit modes, multitasking support, exceptions, co-processing and debug features. Its modern descendants have acquired features such as floating point, many iterations of vector extensions, crypto accelerators, 64-bit mode, and hardware virtual-machine extensions.

While new instructions add CPU implementation complexity [27], past system designers could, for the most

Table 1: Summary of recent Intel x86 ISA extensions

Extension		Year of spec	Year of launch	Instructions new	Instructions chg.	Other ISA changes (excl. feature test bits, XSAVE/VMCS context)
SMEP	Block kernel exec. of user pg.	2011	2012	0	0	
RDRAND	Hardware random numbers	2011	2012	1	0	
FSGSBASE	FS/GS access instructions	2011	2012	4	0	
AVX2	256-bit vector ops.	2011	2013	30	0	wider vector registers
INVPCID	Tagged TLB invalidation	2011	2013	1	0	
VMFUNC	VM optimisations	2011	2013	1	0	
TSX	Transactional mem.	2012	2013[a]	4	0	2 new instr. prefixes, transaction aborts
ADX	Arbitrary-precision arithmetic	2012	2014	2	0	
RDSEED	Hardware random numbers	2012	2014	1	0	
PREFETCHW	Prefetch memory for write	2012	2014	1	0	
SMAP	Block kernel access to user pg.	2012	2014	2	0	
CAT	Cache partitioning	2013	2014	0	0	new model-specific registers
CLFLUSHOPT	Optimised cache flush	2013	2015	1	0	
XSAVEC/XSAVES/XRSTORS	Context switch	2014	2015	3	0	
MPX	Bounds checking	2013	2015	8	4	new instr. prefix, 7 new regs., bound table
SGX1	Secure enclaves	2013	2015	18	2	mem. access rights, exceptions, . . . (see §3)
PT	Processor trace	2013	2015	1	0	9 new model-specific registers, trace buffer
SHA	SHA crypto accel.	2013	2016	7	0	
CLWB	Cache line write-back	2013		1	0	
AVX-512	512-bit vector ops.	2013/14		129	0	wider vector registers
SGX2	Enclave dynamic mem. mgmt.	2014		8	0	
MPK	Protection keys for user-mode	2015		2	0	new register, alters page table format
CET [21]	Code-reuse attack defences	2016		10	9	control transfers, new exception, pg. table

[a] TSX launched with "Haswell" in 2013 but was later disabled due to a bug. "Broadwell" CPUs with the bug fix shipped in late 2014.

part, ignore such changes. Vector extensions (MMX, SSE, and AVX) added data processing instructions, and sometimes widened vector registers, but didn't substantially change systems interfaces. With the notable exception of 64-bit mode and virtualisation extensions, OS developers on x86 were occasionally given tweaks to improve performance (e.g., fast system calls) or correct glaring shortcomings (e.g., user-mode access to FS/GS registers, and TLB tags for non-VM address spaces) but otherwise ignored [29]. Even 64-bit mode didn't substantially increase architectural complexity—registers were added and widened and the page table format changed, but there were only a handful of new instructions. Indeed, some features were effectively removed: segmentation, task switching, and 16-bit modes.

But this has changed. Figure 1 plots the transistor count of Intel x86 CPU implementations (on a log scale), as well as the number of words in the Intel architecture software developer's manual (on a linear scale). Transistor counts were sourced from Wikipedia [40]; manuals from various sources were counted using `pdftotext|wc`. The two data sets are not comparable, but some trends are evident. First, we see Moore's Law; the recently-announced slowdown in Intel's cadence [36] does not yet appear, and aside from a recent 22-core Xeon, Intel has stopped publicising transistor counts. Second is the steady growth, and recent 2015–2016 jump in the general complexity of x86. The jump is due to extensions introduced with the "Skylake" microarchitecture, and dwarfs even 64-bit mode and virtual-machine extensions (both added in 2007).

Table 1 summarises x86 ISA extensions specified and implemented by Intel since the 2012 launch of "Ivy Bridge" CPUs. For each extension we report the year of the first public specification, year of first CPU implementation, number of new instructions, number of instructions whose behaviour was non-trivially changed, and any other significant ISA changes. Prior to 2015, the most complex additions were the AVX2 vector extensions and TSX transactional memory, both introduced with 2013's "Haswell" microarchitecture. TSX was evidently a complex feature to implement—the first implementation turned out to be buggy, and was later disabled via a microcode patch—but had relatively low ISA-level complexity, with only 4 new instructions. Other pre-Skylake extensions were minor, adding single instructions or tweaking protection (e.g., the SMEP/SMAP features).

However, Skylake introduces substantial complexity, including MPX bounds-checking instructions and registers, the processor trace (PT) feature, and SGX enclaves. In total, it adds 31 instructions and a raft of associated changes: new registers, a new instruction prefix, many new processor-level data structures, changes to page ac-

cess rights and exception delivery. Other extensions that Intel has specified but not yet implemented include wider vectors (AVX-512 and related instructions), additional SGX features, a "memory protection keys" feature, and control-flow enforcement technology (CET, §4) which defends against code-reuse attacks. All are included in the latest architecture manual [22] with the exception of CET, which has its own 136-page draft specification [21].

What changed to cause this rapid growth? It's likely that the explosion in extensions is a deliberate strategy.

Since 2007 Intel's processors have followed a "Tick-Tock" development model. Roughly every two years, a new manufacturing process with smaller transistors was introduced (a "tick", or die shrink), followed a year later by a new microarchitecture on the existing process (a "tock"). However, the 2014 roll-out of "Broadwell" CPUs was delayed due to manufacturing problems with the new 14nm process, and in early 2016 Intel settled on a new three-stage development model for 14nm and beyond [12, 36], before apparently backtracking to announce a fourth-generation 14nm architecture for 2017 [38].

The slowing pace of Moore's Law will make it harder to sell CPUs: absent improvements in microarchitecture, they won't be substantially faster, nor substantially more power efficient, and they will have about the same number of cores at the same price point as prior CPUs. Why would anyone buy a new CPU? One reason to which Intel appears to be turning is features: if the new CPU implements an important ISA extension—say, one required by software because it is essential to security—consumers will have a strong reason to upgrade.

3 Case study: SGX

The new instructions introduced by software guard extensions [19] enable strong isolation and remote attestation of software *enclaves*. An enclave is an isolated region of virtual address space, whose contents are protected from access by code outside the enclave. In contrast to prior trusted execution hardware [4, 37], SGX supports secure multiplexing: any number of distrusting enclaves may run concurrently, limited only by resource constraints, without relying on a trusted kernel or hypervisor. Nevertheless, SGX supports a mostly backwards-compatible environment for user-mode code. This compelling combination of features, along with strong physical security (memory encryption), have made SGX attractive to researchers and practitioners alike; in the short time since SGX-capable CPUs appeared, a wide range of applications have been devised [e.g., 5–7, 11, 17, 30, 31, 33], and other vendors are racing to develop similar features [24].

However, SGX introduces substantial complexity: 26 instructions described by nearly 200 pages of English/pseudocode specification [19]. Much of this derives from ambitious design goals: protecting enclaves from malicious privileged software while retaining OS-level management of physical resources using traditional mechanisms (e.g., page tables) to minimise OS changes [29], yet avoiding the need for trusted software. SGX is implemented by a combination of memory encryption hardware, a root of trust (key material) for attestation, new instructions to manipulate and execute enclaves, and changes to page access and exception semantics (i.e., changes to TLB miss and exception handlers). The SGX instructions serve as a *reference monitor for privileged operations* such as changes in the mapping/use of encrypted pages. Memory encryption operates on a fixed physical region known as the enclave page cache (EPC), and the TLB miss handler ensures that each EPC page is accessible only to the enclave that owns it by consulting the *EPC map* (EPCM), a table of metadata for every EPC page (essentially a reverse map). Software has no access to the EPCM; instead, it is updated by instructions such as EADD, which initialises a new page and adds it to an enclave, or EMODPR/EMODPE, which change permissions on an existing page. These instructions perform checks to maintain EPCM consistency and enclave isolation, preventing, for example, EPC double-mapping.

The advantage of implementing memory management in SGX instructions is that no software must be trusted. The disadvantage, compared to a simpler primitive such as a page table, is flexibility: each possible operation requires a new instruction (or set of instructions) to support it. The first version of SGX supports only enclaves whose virtual address layout and permissions are fixed at creation time. This simplifies EPC management, but practically rules out dynamic loading, and makes dynamic memory allocation impractical (the program's maximum footprint must be allocated up front). SGX version 2 will add 8 instructions for basic dynamic memory management, but still lacks the ability to perform seemingly simple operations like moving pages or sharing mappings.

SGX's embedding of memory management in the ISA further contributes to its complexity. On x86, software is responsible for maintaining *TLB consistency* when changing page mappings by flushing the TLB on relevant cores. In order to achieve this, an OS can synchronise between cores using locked data structures and interprocessor interrupts. Neither option is available at ISA level: instructions cannot loop waiting for a lock, nor

signal other cores. Instead, SGX uses a more complex scheme whereby software performs the appropriate operations (generally, forcing threads to exit enclaves) and "proves" to hardware that it has done so using yet more instructions before it may reuse EPC pages.

The SGX reference [19] devotes almost 20 pages to documenting its interactions with prior architectural features including virtualisation, system-management mode, inter-processor interrupts, trusted execution technology, machine checks, and performance monitoring. Besides the added complexity, some of these interactions lead to questionable design outcomes. For example, the CPUID instruction is always illegal inside an enclave merely because a virtual-machine monitor may have configured it to trap. Conversely, the user-mode instructions to write the FS and GS registers are legal inside an enclave, but only if the host OS has enabled them via a control register bit.

While SGX strives to avoid trusted software, this goal comes into conflict with the desire for compatibility with existing OSes: EPC resource management uses normal paging mechanisms. As a result, enclaves are vulnerable to new "controlled-channel" attacks that stem from the OS's ability to induce and observe enclave page faults [35, 41]. These attacks, which can leak enclave data, are serious enough to bring the SGX threat model of a malicious OS into doubt. Perhaps ironically, the best of the known mitigations exploits a seemingly-unintended interaction with the transactional memory extension [34]: transactions abort rather than page fault, so the OS cannot observe transactional enclave memory accesses.

4 Case study: CET

Control-flow enforcement technology defends against code-reuse attacks such as return-oriented programming (ROP). These attacks exploit vulnerabilities in unsafe code like buffer overflows, but rather than directly injecting executable code, manipulate the program's control-flow to execute legitimate instructions in an unintended context [8]. CET consists of two mechanisms: a shadow stack, and indirect branch tracking.

At its core, a *shadow stack* is a straightforward mechanism: on a function call, the processor saves the return address on both the regular and shadow stacks. The shadow stack stores only return addresses, and is inaccessible to normal code. On a return, the addresses from both stacks are popped and compared, and an exception raised if they differ, defeating ROP. The advantage of CET compared to software implementations of shadow stacks is performance, compatibility and security: by modifying the semantics of CALL and RET instructions, no program modifications are needed, and the shadow stack can be made easily and cheaply inaccessible to software through the use of a new page table attribute which protects shadow stacks from access by regular loads and stores.

CET also includes *indirect branch tracking* to prevent misdirection of function pointers: after an indirect JMP or CALL, an exception is raised unless the next instruction is a valid programmer-intended branch target, as signified by a new form of NOP instruction. While this is not full control-flow integrity [1], it restricts the available gadgets.

CET promises to add strong defenses to unsafe C/C++ code, at the cost of substantial architectural complexity. Besides a new exception vector, page table attributes, and model-specific registers, the main complexity arises from feature interaction. Control transfer in x86 is already very complex, including many forms of call and return, such as near and far calls to a different segment or privilege level. In total, nine instructions (some with many variants, such as JMP) are modified by CET. In 32-bit mode, tasking features mean that jumps or calls to particular segments can also switch stacks; CET must account for this. In 64-bit mode, an interrupt can trigger a switch to one of *seven* data stacks via the interrupt stack table. Consequently, CET adds a model-specific register pointing to a table of seven corresponding shadow stacks. Besides modifying semantics of all indirect control transfers, CET's indirect branch tracking must also handle the case where an exception occurs after a branch but before the next instruction, to ensure that an exception can be raised if needed after any return to user mode or context switch.

5 Implications

Sustainability ~~While we may disagree with some of the design choices,~~ these features are, individually, clearly desirable. What is concerning is the rate of change, and the rapid growth in complexity of systems-level features with complex interactions. As the most stable "thin waist" interface in today's commodity technology stack, the x86 ISA sits at a critical point for many systems. A faithful implementation of x86 semantics is essential to myriad computing technologies, including x86-compatible processors, virtual machines, emulators, JIT compilers, dynamic translators, disassemblers, debuggers, profilers, and so on. Of course, not every implementation of x86 must immediately implement every new feature, but over time architectural features stabilise and software generally assumes their presence. Consider, for example, how little of today's software would function on a CPU without a floating-point unit or MMX instructions. Given this, and particularly given the complex interactions be-

tween recent features, we have to question whether the core x86 promise of indefinite backwards compatibility across many implementations is sustainable.

Timescales Since they depend on deploying new CPUs, ISA features are slow to be adopted. The original SGX specification was published in 2013, but the first CPUs to implement it didn't ship until late 2015, and at the time of writing (early 2017) server-class CPUs with SGX support are yet to appear. The SGX version 2 specification was published in 2014, but has yet to be implemented. If we add the delay for sufficient deployment of SGX-capable CPUs (to achieve, e.g., widespread availability in public clouds) the end-to-end deployment time for SGX is likely to approach a decade. This represents a difficult tradeoff for software developers; prior ISA extensions have also taken a long time to deploy, but they have generally only served to accelerate existing functionality; with a feature like SGX, the developer is faced with a stark choice: wait indefinitely for security, or deploy now without it.

Hardware is the new software From a careful reading of Intel patents [23, 28], Costan and Devadas [9, §2.14] conclude that SGX instructions are implemented entirely in microcode. This is logical from an engineering perspective: EPCM updates are off the critical path, and too complex to implement in silicon (they involve multi-word updates and atomic memory accesses). Moreover, a microcode implementation of SGX allows errata to be corrected by updates. We do not have any reason to believe that SGX is unique in this respect—increasingly, new ISA features mean new microcode. Why then, must we wait so long for them to arrive packaged with a new CPU?

Intel and its peers have always been secretive about the boundary between microcode and silicon and the capabilities of microcode updates. We argue that it's time to relax this secrecy, and work to decouple as much as possible the implementation of ISA features from the underlying silicon. This could take two (non-exclusive) forms.

First, CPU vendors could ship microcode updates implementing some new ISA features for prior CPUs. While we shouldn't expect that all features can be implemented this way (some may fundamentally require silicon), nor that they will perform the same, this could be a viable path for faster deployment of features, particularly for complex extensions like SGX. The licensing and revenue model for such updates remains an open question. On one hand, we're used to getting microcode updates for free, but the availability of new features updates might depress the market for new CPUs. Like the on-demand upgrades of the mainframe world, we should probably expect to pay for new features as if we had paid for new hardware. On the other hand, a vendor may wish to encourage adoption of a feature by making it freely available on existing CPUs.

Second, we could extend the architecture to allow software below the OS or hypervisor to implement security features. This would require a privilege level akin to Alpha PALcode [14] or RISC-V machine mode [39], but not a new level of address translation. Such software would inherently be a part of the trusted computing base, but unlike microcode would be under user control, and amenable to inspection and replacement independently of the CPU.

Security A key selling point for many recent security features, SGX in particular, is that no software is trusted. Does the implementation of these features in microcode change this? We argue that microcode is more reliable than current software, but not as inherently secure as we might assume. First, microcode, whose updates are encrypted and signed, is much harder than software for an attacker to modify. Second, CPU vendors have a strong track record of testing. Intel is secretive about their validation processes, and the cost of failure can be much higher than software bugs, but what is known suggests that there is extensive (but certainly not exhaustive) testing [3, 32]. For SGX, Intel has also published the formal verification of a high-level model using an SMT solver [15, 20], and verified the linearisability of a (different) model of concurrent SGX operations [26]. These are important guarantees, but there is no known correctness proof for the implementation, which remains secret.

Ultimately both microcode and the underlying hardware remain opaque, and with ever-increasing complexity, it behoves us to search for ways to improve our confidence in their security. There is one way software can exceed microcode standards of correctness: formal verification of high-level security properties [16, 25]. Recently, Sanctum [10] showed how to implement SGX-like functionality in software for RISC-V; this addresses some of the implementation complexity and side-channel problems of SGX, but leaves open the question of trust, which we are presently tackling through formal verification.

6 Concluding remarks

The growth rate of ISA features is concerning, but has potential upsides for systems research. For one thing, new features (e.g., TSX, MPX, SGX, and PT) have traditionally heralded a slew of publications exploring their possibilities, and we see no reason for this to stop. Like the use of transactions to mitigate SGX control channels [34],

unanticipated interactions between ISA features will lead to the discovery of new techniques. For another, Intel and its peers are likely to be more receptive to implementing ideas from the research community. Finally, rising complexity leads to many familiar systems problems: managing complexity and feature interaction, maintaining legacy compatibility while enabling architectural evolution, adapting (in software, perhaps on the fly) to a heterogeneous set of hardware features, and isolating developers from optimisation trade-offs; systems researchers will find new applications here. We should also renew our efforts to design hardware *primitives* that software can use to implement the features that today become ISA extensions.

More broadly, now more than ever before [2, 13] it's time to rethink the notion of an instruction set. It's no longer the boundary between hardware and software, but rather just another translation layer in the stack.

Acknowledgements

Thanks are due to Chris Hawblitzel, Jay Lorch, Bryan Parno, Stefan Saroiu, and the anonymous reviewers, all of whom provided valuable feedback on drafts of this paper, and to the workshop attendees for a lively discussion.

References

[1] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-flow integrity. In *12th ACM Conference on Computer and Communications Security*, pages 340–353, Nov. 2005. ISBN 1-59593-226-7. doi: 10.1145/1102120.1102165.

[2] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. In *12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 2–13, 2006. ISBN 1-59593-451-0. doi: 10.1145/1168857.1168860.

[3] N. Amit, D. Tsafrir, A. Schuster, A. Ayoub, and E. Shlomo. Virtual CPU validation. In *25th ACM Symposium on Operating Systems Principles*, pages 311–327, 2015. ISBN 978-1-4503-3834-9. doi: 10.1145/2815400.2815420.

[4] *Building a Secure System using TrustZone Technology*. ARM Limited, Apr. 2009. Ref. PRD29-GENC-009492C.

[5] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer. SCONE: Secure Linux containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 689–703, 2016. ISBN 978-1-931971-33-1. URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov.

[6] A. Baumann, M. Peinado, and G. Hunt. Shielding applications from an untrusted cloud with Haven. In *11th USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2014.

[7] S. Brenner, C. Wulf, D. Goltzsche, N. Weichbrodt, M. Lorenz, C. Fetzer, P. Pietzuch, and R. Kapitza. SecureKeeper: Confidential ZooKeeper using Intel SGX. In *17th International Middleware Conference*, pages 14:1–14:13, 2016. ISBN 978-1-4503-4300-8. doi: 10.1145/2988336.2988350.

[8] N. Carlini, A. Barresi, M. Payer, D. Wagner, and T. R. Gross. Control-flow bending: On the effectiveness of control-flow integrity. In *24th USENIX Security Symposium*, pages 161–176, 2015. ISBN 978-1-931971-232. URL https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/carlini.

[9] V. Costan and S. Devadas. Intel SGX explained. Cryptology ePrint Archive, Report 2016/086, Feb. 2016. http://eprint.iacr.org/2016/086.

[10] V. Costan, I. Lebedev, and S. Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In *25th USENIX Security Symposium*, pages 857–874, Aug. 2016. ISBN 978-1-931971-32-4. URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan.

[11] S. Crosby. Using Intel SGX to protect on-line credentials, Aug. 2016. URL https://blogs.bromium.com/2016/08/09/using-intel-sgx-to-protect-on-line-credentials/.

[12] I. Cutress. Intel's 'Tick-Tock' seemingly dead, becomes 'Process-Architecture-Optimization'. *AnandTech*, Mar. 2016. URL http://www.anandtech.com/show/10183.

[13] J. C. Dehnert, B. K. Grant, J. P. Banning, R. Johnson, T. Kistler, A. Klaiber, and J. Mattson. The Transmeta Code Morphing Software: Using speculation, recovery, and adaptive retranslation to address real-life challenges. In *International Symposium on Code Generation and Optimization*, pages 15–24, Mar. 2003. ISBN 0-7695-1913-X. doi: 10.1109/CGO.2003.1191529.

[14] *PALcode for Alpha Microprocessors System Design Guide*. Digital Equipment Corp., May 1996. Order No. EC-QFGLC-TE.

[15] A. Goel, S. Krstić, R. Leslie, and M. R. Tuttle. SMT-based system verification with DVF. In *10th International Workshop on Satisfiability Modulo Theories*, pages 32–43, 2012. URL http://smt2012.loria.fr/paper2.pdf.

[16] C. Hawblitzel, J. Howell, J. R. Lorch, A. Narayan, B. Parno, D. Zhang, and B. Zill. Ironclad apps: End-to-end security via automated full-system verification. In *11th USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2014.

[17] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 533–549, 2016. ISBN 978-1-931971-33-1. URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/hunt.

[18] *Intel 80386 Programmer's Reference Manual*. Intel Corp., May 1987.

[19] *Software Guard Extensions Programming Reference*. Intel Corp., Oct. 2014. Ref. #329298-002 https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.

[20] *SGX Tutorial at ISCA 2015*. Intel Corp., June 2015. Ref. #332680-002 https://software.intel.com/sites/default/files/332680-002.pdf.

[21] *Control-flow Enforcement Technology Preview*. Intel Corp., June 2016. Ref. #334525-001 https://software.intel.com/sites/default/files/managed/4d/2a/control-flow-enforcement-technology-preview.pdf.

[22] *Intel 64 and IA-32 Architectures Software Developer's Manual*. Intel Corp., Dec. 2016. Ref. #325462-061US.

[23] S. P. Johnson, U. R. Savagaonkar, V. R. Scarlata, F. X. McKeen, and C. V. Rozas. Technique for supporting multiple secure enclaves, Dec. 2010. US Patent 8,972,746.

[24] D. Kaplan, J. Powell, and T. Woller. AMD memory encryption. http://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf, Apr. 2016.

[25] G. Klein, J. Andronick, K. Elphinstone, T. Murray, T. Sewell, R. Kolanski, and G. Heiser. Comprehensive formal verification of an OS microkernel. *ACM Transactions on Computer Systems*, 32(1):2:1–2:70, Feb. 2014. ISSN 0734-2071. doi: 10.1145/2560537.

[26] R. Leslie-Hurd, D. Caspi, and M. Fernandez. Verifying linearizability of Intel software guard extensions. In *27th International Conference on Computer Aided Verification*, pages 144–160, July 2015. ISBN 978-3-319-21668-3. doi: 10.1007/978-3-319-21668-3_9.

[27] B. C. Lopes, R. Auler, L. Ramos, E. Borin, and R. Azevedo. SHRINK: Reducing the ISA complexity via instruction recycling. In *42nd International Symposium on Computer Architecture*, pages 311–322, 2015. ISBN 978-1-4503-3402-0. doi: 10.1145/2749469.2750391.

[28] F. X. McKeen, C. V. Rozas, U. R. Savagaonkar, S. P. Johnson, V. Scarlata, M. A. Goldsmith, E. Brickell, et al. Method and apparatus to provide secure application execution, Dec. 2009. US Patent 9,087,200.

[29] J. C. Mogul, A. Baumann, T. Roscoe, and L. Soares. Mind the gap: Reconnecting architecture and OS research. In *13th Workshop on Hot Topics in Operating Systems*, May 2011.

[30] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. Oblivious multi-party machine learning on trusted processors. In *25th USENIX Security Symposium*, pages 619–636, 2016. ISBN 978-1-931971-32-4. URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko.

[31] R. Pires, M. Pasin, P. Felber, and C. Fetzer. Secure content-based routing using Intel software guard extensions. In *17th International Middleware Conference*, pages 10:1–10:10, 2016. ISBN 978-1-4503-4300-8. doi: 10.1145/2988336.2988346.

[32] H. Rotithor. Postsilicon validation methodology for microprocessors. *IEEE Design & Test of Computers*, 17(4):77–88, Oct. 2000. ISSN 0740-7475. doi: 10.1109/54.895008.

[33] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. VC3: Trustworthy data analytics in the cloud using SGX. In *IEEE Symposium on Security and Privacy*, May 2015. doi: 10.1109/SP.2015.10.

[34] M.-W. Shih, S. Lee, T. Kim, and M. Peinado. T-SGX: Eradicating controlled-channel attacks against enclave programs. In *Annual Network and Distributed System Security Symposium (NDSS)*, Feb. 2017.

[35] S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena. Preventing page faults from telling your secrets. In *11th ACM Asia Conference on Computer and Communications Security*, pages 317–328, 2016. ISBN 978-1-4503-4233-9. doi: 10.1145/2897845.2897885.

[36] T. Simonite. Intel puts the brakes on Moore's Law. *MIT Technology Review*, Mar. 2016. URL https://www.technologyreview.com/s/601102.

[37] *TPM Main Specification Level 2*. Trusted Computing Group, Mar. 2011. Version 1.2, Revision 116.

[38] M. Walton. Intel will release 8th-gen Coffee Lake chips this year—still at 14nm. *Ars Technica*, Feb. 2017. URL https://arstechnica.com/gadgets/2017/02/intel-coffee-lake-14nm-release-date/.

[39] A. Waterman, Y. Lee, R. Avizienis, D. A. Patterson, and K. Asanović. The RISC-V instruction set manual volume II: Privileged architecture version 1.7. Technical Report UCB/EECS-2015-49, UC Berkeley EECS, May 2015.

[40] Wikipedia. Transistor count. Retrieved 2017-01-12. URL https://en.wikipedia.org/wiki/Transistor_count.

[41] Y. Xu, W. Cui, and M. Peinado. Controlled-channel attacks: Deterministic side-channels for untrusted operating systems. In *IEEE Symposium on Security and Privacy*, May 2015. doi: 10.1109/SP.2015.45.