Side-channel attacks: A short tour

Frank Piessens and Paul C. van Oorschot*

March 2024

We provide a brief, accessible introduction to *side-channel attacks*, a growing subarea of computer security. We explain the key underlying ideas, give a chronological overview of selected classical attacks, and characterize side-channel attacks along several axes.

Introduction

In computer science we often think of a program as a function that maps input data to output data. However, execution of a program on a real machine is much more involved. What is typically viewed as input or output is only a small fraction of actual interactions that take place between a program and its environment during execution.

For example, executing a program on a digital electronic computer will consume power, emit electromagnetic radiation, alter memory, and take a measurable (often observable) amount of time. Depending on the execution environment, program execution may cause many other effects, including microarchitectural effects such as moving data from main memory to a cache, architectural effects such as page faults, and programming language runtime effects such as garbage collection.

A *side channel* is any exploitable source of information about a program or its data (aside from a program's *intended* input or output channels) obtained by examining interaction or interfaces between the executing program and its environment. Side-channel attacks use the side channel to learn information about some aspect of program state (as classical examples, by measuring time or power consumption) or to unexpectedly modify program state (such as by inducing bit flips through power glitching). A typical goal is to extract secrets such as a cryptographic key or password.

The closely related term *covert channel* is used when a program is designed or altered to intentionally use such a channel to exfiltrate information; in this article, we focus on side channels where a victim program inadvertently leaks information.

*Author's version of the Security Knowledge column from *IEEE Security & Privacy*, pp.75-80 (Mar-Apr 2024), https://www.computer.org/csdl/magazine/sp/2024/02

Side-channel leakage can occur while a program is doing computation (for example, execution time could depend on data being processed), or when moving input/output (I/O) through an intended channel—for example, the time between keystrokes could leak information on what keys are being pressed.

A vast array of attack techniques that exploit side channels has been developed and studied over several decades, and side-channel attacks are a serious threat to computer security in many settings. As noted, cryptographic secrets are an important class of targets; a cryptographic primitive might be secure when considered as an abstract mathematical function, yet depending on the software implementation details and platform used, could be vulnerable to a range of side-channel attacks.

We now proceed with a historical overview of some classical attacks.

Leaks through physical side-effects

Electronic computing devices emit electromagnetic signals. That these can often be picked up by an attacker has been well known for decades in military communities (and longer for electronic communication devices). Ross Anderson's book [1] notes many examples providing historical context.

Side-channel attacks came to the attention of the broader security community in 1996, when Paul Kocher [6] showed that the execution time of common implementations of public-key cryptographic algorithms leak information about their private keys, in some cases enough to recover the entire key.

Example 1. To understand the idea behind time-based side-channel attacks, recall that two famous public-key algorithms, RSA and Diffie-Hellman key exchange, require discrete exponentiation with large secret integer exponents. This is commonly implemented using a *square-and-multiply* algorithm, which proceeds through each bit position in the exponent. For each bit position, one squaring is done, plus a further multiply conditional on the exponent bit in that position being 1 (not when 0). A 1-bit therefore requires extra computation, and the time difference leaks this bit. Recovering this bit from the overall execution time is trickier, but accumulating timing data over a sufficient number of executions often allows (perhaps surprisingly) recovery of the entire secret key.

Power use also leaks information about a computation in different ways, so that even if an implementation is protected against timing leaks, it may remain vulnerable to a side-channel attack that observes power consumption. It was again Kocher and collaborators [7] who demonstrated this through a general attack technique called *differential power analysis.*

Example 2. A differential power analysis attack is a statistical attack that exploits correlations between the data processed by a program and the program's power consumption. To explain, suppose the program's immediate power consumption at one specific time point T in execution correlates with the value of one specific bit b of the program state. Power measurements are stochastic and noisy, and a single measurement may not suffice to infer anything, but the correlation can be exploited statistically.

For a sufficiently large set S of power measurements of different program executions (always at time T in the execution), let $avg(S)$ be the average of these measurements. Differential power analysis relies on the observation:

- If S is *randomly* partitioned into subsets S' and S'', then $avg(S')$ will be (statistically) equal to $avg(S'')$, and both statistically equal to $avg(S)$.
- However, if we partition S in subsets S_0 (all executions where $b = 0$) and S_1 (for $b = 1$), then due to the correlation, $avg(S_0)$ and $avg(S_1)$ will differ.

The attacker cannot partition S based on the value of b directly, as they do not know b for each of the executions in S. Instead, the attack uses a *selection function* that partitions executions, based on the (variable) public inputs or outputs of the program (e.g., ciphertext), and on a part of a (constant) secret to be extracted from the program (e.g., a byte of the key). The selection function is designed such that a correct secret will partition executions according to the value of b (while an incorrect secret will partition executions more or less randomly). The attacker now partitions S with the selection function using the known public inputs of each individual execution, and a *guess* for the secret, and determines whether the guess is correct or not based on the observation above. Recovering the secret is now a simple matter of trial-and-error. Differential power analysis attacks turned out to be surprisingly powerful in practice.

Many of these early attacks focused on smartcards, for an attacker with direct physical access to it. Timing and power attacks were then developed further to work as remote attacks. In 2003, Brumley and Boneh [4] convincingly demonstrated a timing attack that recovered an OpenSSL-based private key from a remote server over a local network. For power attacks this took longer—a 2021 paper [10] shows how to perform power attacks without physical access, via software-accessible interfaces that enable measurement of power consumption.

Leaks through micro-architectural side-effects

An important next step in side-channel attacks was the observation that not only physical side-effects mattered. Processors use a plethora of optimization techniques such as caching, pipelining, branch prediction, and out-of-order execution. These optimizations require the processor implementation (or *micro-architecture*) to maintain state, like the contents of the cache, or the state of the program flow branch predictor. Side-effects of program execution then extend to this micro-architectural state.

Attacking programs by observing micro-architectural side-effects or their consequences was an idea already noted in Kocher's 1996 paper on timing attacks. The first practical attacks appeared in the early 2000s independently by several researchers.

Example 3. Micro-architectural optimizations such as caching may simply enhance the power of attacks that measure execution time, as in the 2006 *cache-collision attack* by Bonneau and Mironov [3], recovering 128-bit AES keys. The core idea is that accessing a variable's value takes less time than otherwise if it is already in the cache, and some instances of program data (here, plaintexts or ciphertexts for a fixed unknown key) yield more cache hits than others. This particular attack exploited data-dependent table lookups into cached memory; the lookup indexes depended on known data bytes and unknown key bytes. Software analysis (*whitebox analysis*) provided equations relating unknown values to values that were known or could be deduced. Some unknown values had higher probability in the case of a cache hit, and hits were identifiable by times that differed significantly from averages. From this, compiled timings and data values from a sufficient number of samples enabled key recovery. This attack was for specific software implementations of AES running on Pentium III processors, and required 2^{13} samples (today's processors have AES opcodes, obsoleting the use of AES lookup tables and thus this particular attack).

A more fundamental novelty of micro-architectural side-channel attacks is that the attacker can measure micro-architectural effects by measuring the execution time of *attacker code* rather than victim code, as in the next example.

Example 4. A simple but very powerful example is a *flush+reload* attack. This assumes the attacker and victim share some memory (such as the code pages of a shared library). The attacker flushes (removes) a specific memory address from the cache, then waits for the victim to run. Then the attacker measures how long it takes to reload from the same memory address. A fast reload means the victim accessed the memory address (causing it to be cached again). If the reload is slow (not cached), the victim likely did not access it subsequent to the attacker's flush. Thus the attacker learns something about the victim program state from the presence (or absence) of the victim memory access. Yuval and Falkner [16] demonstrate the power of the attack by using it to extract private encryption keys from a victim program.

Following early methods such as these, many variants of cache attacks were developed to match different attacker models. And aside from the cache, various other micro-architectural elements have been enlisted for side-channel service; Ge et al. [5] provide an excellent survey circa-2018.

The breadth and severity of micro-architectural side-channel attacks came into focus in 2019 with the discovery of two *transient execution attacks*, Spectre and Meltdown [8, 9]. Processors that use out-of-order and speculative execution rely on a roll-back mechanism to suppress the effects of mis-speculated execution; instructions that are speculatively executed and later rolled back are called *transient* instructions.

The key idea of a transient execution attack is to *influence* a processor's prediction and speculation mechanisms so as to transiently execute code snippets that send out secrets over a micro-architectural side channel. We thus view these not as traditional side-channel attacks, but attacks that create conditions whereby information becomes available through a side channel.

With a transient execution attack, an attacker (to some extent) controls the *sending side* of the side channel, and this significantly amplifies the amount of information that can be leaked. Our Sept/Oct 2023 column [13] provides a detailed explanation of transient execution attacks.

Leaks through architectural side-effects

Analogous to processors, system software implementations employ optimizations that while intended to be invisible to application software, may enable new side channels. Two examples of optimizations that have been exploited are *memory deduplication* and *demand paging*, as explained next.

Memory deduplication is an optimization in virtual memory systems. If different processes or virtual machines have pages with identical contents in their virtual memory address space, system software can *deduplicate* them and store the corresponding page only once in physical memory. The two virtual address spaces then *share* this physical page, saving physical memory. The system software uses the *copy-on-write* technique to recreate two versions of the physical page as soon as one of the processes writes to the deduplicated page. Hence on systems employing this optimization, program execution has side-effects on the deduplication state.

Suzaki et al. [14] first proposed using such side effects to attack other programs on the same system. The timing delay from the copy-on-write introduces a side channel that informs attackers whether a page with the same content exists elsewhere in the system. This was first used to detect the presence of executables or downloading of files in other virtual machines in a cloud setting, and later refined to work from more restricted settings (such as JavaScript code in a browser) and to leak more fine-granular information.

A trend amplifying the danger of such *architectural* side channels is the desire to remove system software from the trusted computing base. *Confidential computing*, supported by trusted execution technologies such as Intel SGX, Intel TDX, and AMD SEV, aims to enable the execution of application software in the cloud, without the requirement to trust the cloud provider.

In such a setting, system software is necessarily untrusted; protected applications are shielded from system software, but a direct consequence is that many architectural side-effects become dangerous side channels. Xu et al. [15] introduced the notion of *controlled-channel attack*, a category of side-channel attack allowing untrusted system software to learn information about such protected applications by observing architectural side-effects of their (shielded) execution.

The prime example of such a side-effect is a page fault due to demand paging (whereby pages of a process are loaded into main memory only when needed). System software can mark memory pages of the protected application as invalid, and will then be notified by the occurrence of a page fault, whenever the protected application accesses a memory address in one of these pages.

Controlled-channel attacks have advanced to be more fine grained and more stealthy (for instance by relying on the system software's control over interrupts), and are now among the most powerful attacks against confidential computing systems, along with micro-architectural side-channel attacks.

Characterizing side-channel attacks

The range of side-channel attacks has expanded tremendously since 1996. The breadth is highlighted by characterizing attacks along two axes. The first dimension follows our historical overview, grouping attacks based on the phenomena on which the information channel is built:

- *physical* phenomena, such as execution time, consumption of energy or power, electromagnetic radiation, emission of sound;
- *micro-architectural* phenomena, such as cache state, predictor state, contention for processor resources like execution ports or floating point units, data-dependent execution time of instructions; and
- *architectural* phenomena, such as page faults, page deduplication, performance counters, interrupts.

The second axis acknowledges that whether a specific side channel is exploitable depends heavily on the attacker model—the kinds of power they have, in terms of observation or manipulation. We identify the following cases:

- the *physical-access* attacker—with physical access to the computing device running the victim program. They can place probes or sensors to observe and measure physical phenomena. This is an important attacker model for smartcards and IoT systems.
- the *remote* attacker—with only remote access to the target computing device through a network interface. They are limited to measuring time, or side-channel signals available through the network communication itself. This model is relevant for Internet servers.
- the *shared-platform* attacker—who can run code on the platform where the victim program is executing, but is isolated from that program by some mechanism such as process isolation or virtual machine isolation. This attacker can monitor micro-architectural and architectural phenomena. This model is relevant for any platform that runs code from multiple stakeholders, thus including cloud, mobile and desktop platforms.
- the *privileged* attacker—who can control the system software of the platform on which the target program runs under protection of trusted execution technologies. This attacker can monitor a broad selection of physical, micro-architectural and architectural signals. This model is relevant for the emerging field of confidential computing.

Attack Model	**Information Channel Phenomena**		
	Physical	*Micro-architectural*	*Architectural*
Remote	timing [4]		
Shared platform		flush+reload [16] fault injection [12]	deduplication [14]
Privileged	power [10]		controlled-channel [15] fault injection [11]
Physical access	timing [6] power [7] fault injection [2]	cache [3]	

Table 1: Characterizing selected side-channel attacks.

Table 1 classifies the attacks we discussed along these two axes.

A further distinguishing aspect is whether information is leaked by *computation*, or by *communication* on one of the intended I/O channels. Our discussion has focused mainly on leakage through computation, but side-channel attacks have been applied to a wide variety of I/O devices, showing for example how a phone's accelerometer can be used to deduce where a user is tapping on the screen, or showing how to reconstruct the image on a computer display from the electromagnetic radiation emitted by the display.

Finally, while our main focus has been side channels that directly leak secrets from a victim program (a confidentiality threat), one can also consider using side channels to influence program state (an integrity threat), e.g., by inducing faults as in *fault injection* attacks. On this, insightful 1997 work by Boneh et al. [2] explained how transient hardware faults (or inducing them) would enable attacks on public-key algorithms, inspiring Biham and Shamir's *differential fault analysis* attacks on symmetric-key algorithms later that year. The original context was the physical access model (as per classic side channel attacks), but more recent attacks are applicable in shared-platform or privileged attack models.

As one example, the well known RowHammer attacks [12], first published in 2014, cause bit-flips in DRAM memory by very frequently accessing memory locations in the same DRAM bank, but different rows. As another, PlunderVolt [11] injects faults in an Intel SGX protected application by using privileged software interfaces to manipulate CPU voltage and frequency.

We end by noting that having grown immensely in scope and impact since 1996, we expect side-channels to continue as a hot topic in research. They are recognized as an attack surface to be addressed in the design and implementation of almost all deployed systems. Due to timing attacks in particular, constant-time implementations are now an important feature in cryptographic libraries, to decouple execution times of cryptographic algorithms from the values of secrets, for example ensuring that branching decisions are not dependent on bit-values in secret keys. Likewise due to cache timing attacks, memory accesses should be decoupled from the values of secrets. We plan to say more about defenses against side-channel attacks in a future article.

References

[1] R.J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems.* Third edition. John Wiley and Sons, 2020.

[2] D. Boneh, R.A. DeMillo, R.J. Lipton. On the importance of checking cryptographic protocols for faults. EUROCRYPT 1997: 37-51.

[3] J. Bonneau, I. Mironov. Cache-collision timing attacks against AES. CHES 2006: 201–215.

[4] D. Brumley, D. Boneh. Remote timing attacks are practical. USENIX Security 2003: 1–13.

[5] Q. Ge, Y. Yarom, D. Cock et al. A survey of micro-architectural timing attacks and countermeasures on contemporary hardware. *J. Cryptogr. Eng.* 8, 1-27, 2018.

[6] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. CRYPTO 1996: 104–113.

[7] P. Kocher, J. Jaffe, B. Jun. Differential power analysis. CRYPTO 1999: 388–397.

[8] P. Kocher et al. Spectre attacks: Exploiting speculative execution. IEEE Symp. on Security and Privacy 2019: 1–19.

[9] M. Lipp et al. Meltdown: Reading kernel memory from user space. USENIX Security 2018: 973–990.

[10] M. Lipp et al. PLATYPUS: Software-based power side-channel attacks on x86. IEEE Symp. on Security and Privacy 2021: 355–371.

[11] K. Murdock et al. Plundervolt: Software-based fault injection attacks against Intel SGX. IEEE Symp. on Security and Privacy 2020: 1466–1482.

[12] O. Mutlu and K.S. Kim. RowHammer: A retrospective, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 39(8): 1555-1571, Aug 2020.

[13] F. Piessens. Transient execution attacks. *IEEE Security & Privacy*, 21(5): 79–84, Sept/Oct 2023.

[14] K. Suzaki et al. Memory deduplication as a threat to the guest OS. 4th European Workshop on System Security 2011: 1–6.

[15] Y. Xu et al. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. IEEE Symp. on Security and Privacy 2015: 640–656.

[16] Y. Yarom and K. Falkner. Flush+reload: A high resolution, low noise, L3 cache side-channel attack. USENIX Security 2014: 719–732.