

Side Channels and Runtime Encryption Solutions with Intel[®] SGX

by Andy Leiserson, Chief Architect at Fortanix

Executive Summary

Fortanix[®], the leader in Runtime Encryption, has received numerous inquiries about the impact of side channel attacks on Fortanix's solutions and more generally on Intel[®] SGX. This white paper briefly explains side channel attacks, then discusses some specific side channel attack techniques in the context of SGX and ways of defending against those attacks.

It is important to be aware that side channels exist in any digital system, not just SGX. The interest in side channels attacks in the context of SGX arises because many kinds of attacks that are possible in conventional computing environments are no longer possible in SGX, leaving side channel attacks as one of few remaining attack vectors. Several factors that are not always discussed in academic literature can make it infeasible or extremely difficult to mount a side channel attack in a practical setting.

This paper also describes various defenses used by the Fortanix Self-Defending Key Management Service[™] (SDKMS) and by Fortanix Runtime Encryption solutions to protect against side channel attacks. Fortanix leverages its considerable hardware-based security, side-channel, and cryptography expertise to deliver Runtime Encryption solutions that ensure data remains protected even when in use.

This paper is intended for an audience that is generally familiar with processor architecture and Intel[®] SGX technology. Familiarity with side channel attacks is helpful but is not assumed.

Introduction to Side Channel Attacks

A **side channel attack** is a way to extract sensitive information from a system by some means other than the intended input and output channels. A conventional attack on the security of a digital system might work by supplying malicious input that, due to a logical error in the implementation of the system, results in sensitive data being included with the output. In contrast, a side channel attack might look at a property like the response time of the system and determine secret information based on changes in the response time. A side channel attack is like the digital equivalent of a safecracker using a sensitive listening device to determine the state of mechanical components in a lock. The sensitive information gleaned by a side channel attack is known as **side channel leakage**, and the system subject to the attack is said to **leak side channel information**. A side channel leakage can take many forms, including variation in the time taken by the system to process different inputs, variation in the electrical activity of a circuit when processing different inputs, electromagnetic emissions from a circuit, and even sound emitted from a cryptographic device. In addition to monitoring side channel leakage, the attacker may supply the system with chosen inputs, or otherwise tamper with the operation of the system to maximize the utility of the leakage.

Side channel attacks are commonly described in the context of cryptographic systems, where the leaked information is a cryptographic key. In this paper, we consider a more general definition that includes leakage of any kind of sensitive information.

Example

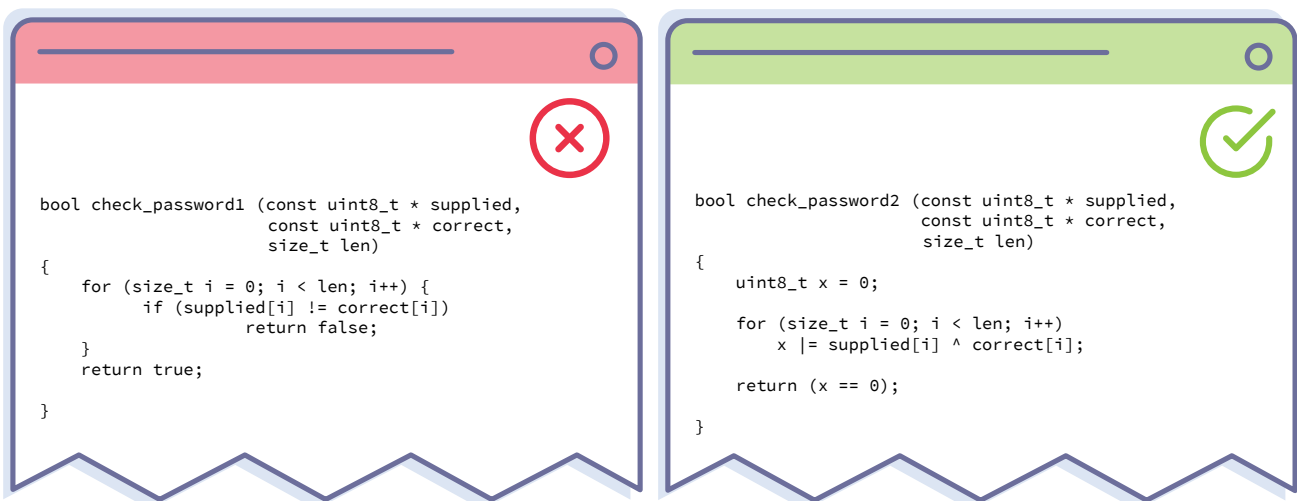
As a simple example of a side channel attack, consider a system that verifies whether an entered password is correct.



A naïve implementation may use a function like `strcmp` to test whether the entered password is equal to the correct password. The `strcmp` implementation may stop processing the input upon finding the first incorrect character. When the first character of the entered password is incorrect, the system will immediately report that the password is incorrect. When the first character of the entered password is correct, the system will take slightly longer to report that the password is incorrect. By watching carefully for this difference in response time, an attacker can try each possible first character in turn until the correct one is found, then try

each second character in turn until the correct one is found, and so on. Eventually the attacker will have determined the correct password, using many fewer guesses than it would take to try all possible passwords.

A side channel resistant implementation might use a function that compares passwords in constant time regardless of the entered password. One way of accomplishing this is comparing the hash of the strings rather than comparing the raw strings. Comparing the hashes ensures that attacker cannot determine the position of the first mismatch.



Sources of Side Channels and Mitigation Strategies

In the case of digital computation, a side channel can exist when execution of a program modifies externally-visible state, outside of explicit inputs and outputs to the program [1]. Thus, side channels can be organized by the resource(s) used to construct the channel. The password comparison example above is often called a **timing side channel**, where a secret can be inferred based on the execution time of the application.

In the context of cloud computing, a primary concern is side channels that exist in hardware shared by co-resident virtual machines or processes. Most proof-of-concept side channel attacks in the cloud environment infer secrets based on changes to shared hardware caches [2], [3]. The caches are part of the CPUs memory subsystem, so these attacks are classified as using a **memory side channel**.

Another side channel used in SGX attacks [4] is the **branch predictor side channel**. The CPU's branch predictor holds information about observed branch behavior and thus may reveal control flow within an enclave.

Ingredients in a Side Channel Attack

A successful side channel attack requires the following key ingredients in the target:

Ingredient 1

The side channel itself, meaning a resource like the ones above, shared by attacker and target.

Ingredient 2

Secret-dependent, externalized behavior. For example, a branch decision (if/else statement) may depend on the value of a secret and the two sides of the branch have different impact on the resource in the previous ingredient.

Ingredient 3

Sufficient measurement precision and volume. Because many unrelated operations can also affect the shared resource, the attacker needs to be able to measure the side channel leakage with enough accuracy to recover useful information.

For instance, in the simple password checker example above, it is relatively easy to measure the execution time of the password comparison executed in isolation.

In a larger application, such as a database or web server, separating the time spent on password checking logic from other operations like establishing a connection or spinning up a worker thread can be more difficult. In the presence of measurement noise, the attacker may need to make a larger number of measurements to mount the attack successfully, which increases the chance of detection.

Mitigation Strategies

Most strategies for preventing against side channel attacks try to disable one or more of the key ingredients above. Here, we summarize a few of the major families of defenses.

Defense 1

Removing secret-dependent behavior. An example is removing secret-dependent branches from code, instead computing results from both sides of the branch and selecting the correct output at the end. Another option is to ensure that both sides of a branch have equivalent impact, for example by ensuring that both branches touch the same cache lines or virtual addresses, thereby making page faults or cache behavior indistinguishable. For timing channels, some defenses have set an expected execution time and always execute for this period, even if not doing useful work. Other defenses terminate after a fixed interval even if the computation is not finished.

Defense 2

Hiding secret-dependent behavior. Most successful side channel attacks require careful analysis of the target to find and characterize the side channel leakage. Although not a perfect defense, keeping the details of the target (e.g. the application binary) secret from the attacker can raise the difficulty of a successful side channel attack. In principle, randomized compilation or other techniques that randomly perturb control flow may work, however, correct implementation is difficult. Strategies like the out-of-the-box address space layout randomization (ASLR) in modern operating systems are too coarse to eliminate side channels [5], [6].

Defense 3

Making the victim sensitive to measurements of side channel leakage. When an attacking process is measuring a side channel leakage in a shared resource, the measurement itself can affect the victim. In the shared cache example above, execution of measurement code will itself perturb the victim's cache behavior, and consequently its execution time. Thus, some defenses have the victim measure its own timing, and cease execution if an attack is suspected [7], the attack successfully, which increases the chance of detection.

Meltdown and Spectre

Security researchers recently announced the discovery of **Meltdown** and **Spectre** [8], [9]. Meltdown and Spectre are side-channel vulnerabilities affecting CPUs. The vulnerabilities arise from the **speculative execution** functionality in modern high-performance CPUs. Unlike early CPUs, which executed a sequence of machine instructions one by one, modern CPUs may speculatively begin processing subsequent instructions while a previous instruction is still in flight. If the previous instruction fails, the CPU must ensure that the machine state exposed to software does not reflect any effects of the speculatively executed instructions. The **Meltdown** and **Spectre** vulnerabilities arise because a process can cause the CPU to speculatively access data that the process should not have access to. The CPU will not expose the data to the process directly, but the process may be able to recover the speculatively accessed data via a side channel. Spectre and Meltdown are a reminder of the ubiquity of side channels, and that side channels exist everywhere, not just in SGX or in any other single environment.

Introduction to Intel® SGX

Intel® Software Guard Extensions provide a protected execution environment within an x86 CPU that significantly reduces the attack surface for code running in that environment. Using the SGX instructions, an application can create a private region of memory that is isolated from all other processes, even those with higher privilege levels. Thus, even if a malware or an insider has access to operating system (OS) root privileges, or if the hypervisor or BIOS are compromised, the SGX-protected software can still operate with integrity and be able to help protect both its code and data.

Traditionally, x86 architecture follows a hierarchical privilege mode with various software components operating at different privilege levels. Less privileged software components have no privacy from processes with greater privilege. SGX enables programmers to create a stand-alone execution environment for applications. This environment, called an **enclave**, operates with a private region of system memory that the CPU makes available only to the enclave. No other software component, not even those running with higher privilege level, can access the enclave memory. The software trust boundary of an enclave is exactly the boundary of the enclave. Other software on the host, and system administrators with access to the host, are outside the trust boundary.

Promise of Intel® SGX

Intel® SGX enables the processor hardware to serve as the root of trust for software executing in SGX enclaves. This creates an execution environment secure against snooping or tampering by compromised local system software or a malicious privileged user. Data in enclave memory is transparently encrypted by the CPU prior to being written to DRAM. The processor security hardware enforces that only the enclave, and not any other software, can access this memory.

SGX also provides attestation functionality. This functionality enables a remote party to verify the integrity of enclave software by examining a cryptographic proof (the attestation). Attestations are issued by the processor hardware itself, thus protecting the attestation process from software attacks of any sort.

By combining encryption of data in use with attestation, SGX offers secure computation in an untrusted environment, without exposing the data being operated upon.

However, the ultimate strength of any secure system depends on many factors. Nearly any x86 code can be run in an SGX enclave. To achieve a truly secure system, the design, implementation, and testing of code running in an enclave is critically important. An error such as a buffer overflow in an enclave application may leave the enclave

vulnerable despite all the security protections afforded by the SGX platform. Enclave applications must conform to general secure coding principles as well as security considerations specific to SGX. As discussed in the Intel® Software Guard Extensions Developer Guide [10], the SGX platform does not automatically render applications resistant to side channel attacks. To avoid side channel vulnerabilities in their applications, developers must exercise great care in the design and implementation of their application.

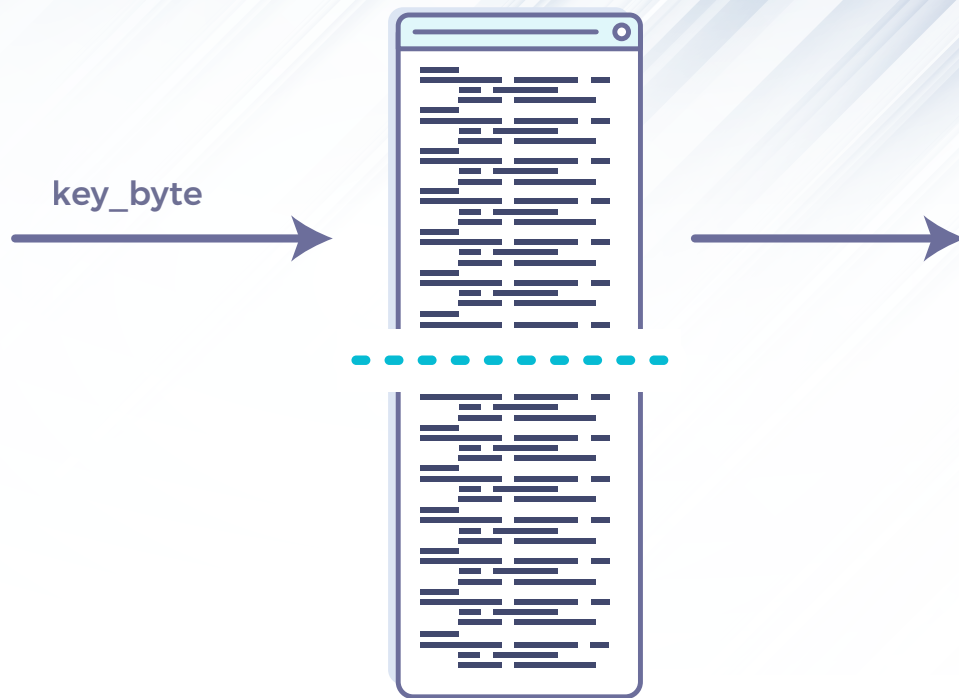


Side Channels Applicable to SGX Enclaves

The vulnerability of enclave software to side channel attacks depends greatly on details of the software implementation. Cryptographic primitives are easier to protect, because they tend to have regular structure and are relatively limited in complexity and scope. Enclave applications that make use of large memory data structures are harder to protect, because the memory access pattern may leak information. The SGX platform does not natively protect against this kind of leakage, however, there is research into implementing protection on top of SGX [11].

Timing Side Channels

Timing side channels apply to SGX enclaves. The nature of the timing side channel in an enclave application will in most cases be the same as it would be in a non-enclave version of the same application. For example, an attack that measures response time to network requests is not unique to the enclave environment. In the case of timing side channels, similar design strategies and countermeasures can be used in either case to provide resistance to this type of attack.



Memory Side Channels

SGX enclaves have been shown to be vulnerable to memory side channel attacks. This can be the same as the cache side channel that exists more generally in applications running on a shared processor, or there are some variants tailored to SGX. Memory access side channels are exploitable when the memory access pattern depends on sensitive data. An example is a cryptographic algorithm with a key-dependent lookup table. If the side channel reveals which part of the lookup table is accessed, an attacker can make some conclusion about the secret key.

Several forms of memory side channel attacks on SGX have been demonstrated. One method generates cache accesses in untrusted code that reveal which cache lines are accessed by the enclave. Alternately, an the attack may use the ability of the untrusted OS to control SGX paging to identify which memory page(s) are being accessed by the enclave [5]. The untrusted OS is responsible for swapping enclave pages from their primary storage, the enclave page cache (EPC), to secondary storage (other RAM or disk) when necessary.

The OS can swap out enclave pages and then monitor page faults from within the enclave to determine which pages the enclave is accessing. A third method modifies the second by monitoring tracking information in the OS page tables to determine which pages the enclave is accessing without inducing page faults [6].

Branch Predictor Side Channels

A final class of side channel that has been shown to apply to SGX enclaves is the CPU's branch prediction hardware. The purposes of the branch predictor is to record information about recently seen branches so that the CPU may more accurately guess the result of future conditional or indirect branches. Aspects of the branch predictor are shared between enclave and non-enclave processes, which may allow the non-enclave process to infer secret-dependent control flow in the enclave [4].

Discussion

In the general cloud computing case, a straightforward but potentially costly countermeasure is to avoid sharing hardware resources between code with different trust levels (i.e. code belonging to different cloud tenants), thus removing potentially malicious processes from a position where they can mount an attack. However, a goal of SGX is to provide security even when inherently co-located components like the OS or hypervisor are not trusted. A compromised OS or hypervisor has control over scheduling, timing, and most shared resources (ingredient #3).

Although the processor's built-in single-step mechanism is disabled while running enclaves, the OS can use system resources like high-precision timers to schedule enclave operations at near-single-step frequency [12]. This allows an attacker with control over the OS to collect nearly noise-free information about memory accesses by the enclave.

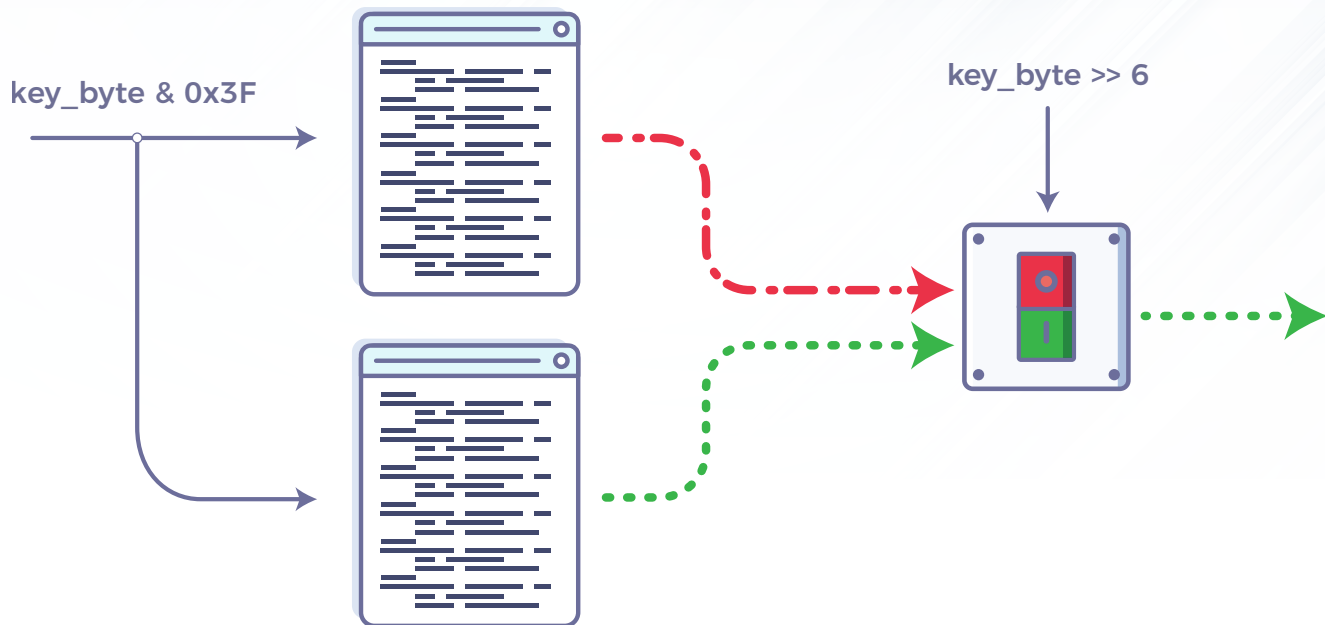
However, the privileged position of the attacker matters only if a vulnerability exists. Side channels are unavoidable, but a well-designed application can ensure that they do not carry sensitive information. Several works demonstrating memory side channel attacks on cryptographic algorithms in SGX enclaves have targeted well-known cryptographic libraries (e.g. OpenSSL, mbedTLS, GnuTLS) that can be run either inside or outside of an enclave. In some cases, the implementations targeted by the attack have known weaknesses that exist independent of whether it runs in an enclave or not [4], [13]. In one case, a new vulnerability was found, but the vulnerability applies generally to the cryptographic library, not specifically to SGX [14].

Most security literature defines a threat model, which explicates how strong of an adversary the defense will resist (or how much power the attacker requires to succeed).

Literature claiming to have found a vulnerability must be evaluated in the context of the threat model. Although some SGX papers treat side channels as out of scope, all SGX work adopts a worst-case threat model where the OS or hypervisor has already been compromised. Papers targeting scenarios other than SGX may assume a weaker threat model, such as assuming the hypervisor has not been compromised. The choice of threat model in a formal security analysis has no bearing on the actual probability of an OS or hypervisor compromise, or a malicious cloud provider. In practice, using SGX places you at no more at risk of a compromised OS or hypervisor than not using SGX. However, assuming an OS or hypervisor is compromised, the SGX adversary is limited to searching for and using side channels to extract information, unlike the non-SGX adversary, who upon compromising the OS may immediately read sensitive information from application memory. There is no evidence to suggest that moving an application to SGX makes it more vulnerable to side channel attacks than it already was.

Protection Against Side Channel Attacks

This section describes some strategies for protecting applications from side channel attacks. It follows the families of defenses discussed earlier.



Removing Secret-dependent Behavior

The most direct and robust strategy to protect against side channel attacks is to remove secret-dependent behavior that may leak through the side channel. This can be applied to the case of memory side channels (in SGX, or in general) by ensuring the program exhibits the same pattern of memory accesses regardless of the data being operated upon. For example, the code on both sides of an if/else conditional would be placed in the same cache line. Similarly, any data access with indirect addressing that could span more than one cache line would need to be replicated such that all cache lines are accessed regardless of the lookup index [6], [13].

Another strategy that removes secret-dependent behavior is using machine-intrinsic cryptographic instructions, rather than software implementations, wherever possible. Machine instructions are likely to be implemented in a manner that is free of timing side channels. This is the case for Intel's AES-NI instructions.

In several of the cases where memory side channels have been used to attack an enclave, the attack works because it can determine which entry in an array or hash table is being accessed. A simple countermeasure is to utilize randomized hashing. This is a good idea in many networked applications, whether running in SGX or not, to defend against denial-of-service attacks where an attacker supplies inputs chosen to produce hash collisions. Hashes and other randomized data structures should be periodically rebuilt with different randomization, to prevent an attacker from collecting sufficient observations of the structure to infer the location of any particular data.

Hiding Secret-dependent Behavior

One technique for hiding secret-dependent behavior is to deliver the enclave code in encrypted form. The only plaintext code in the enclave image would be a minimal loader, responsible for decrypting the rest of the application. Protecting the enclave image from inspection makes it much more difficult for an attacker to reverse-engineer the enclave code to identify exploitable side channel leakage. An advantage of this countermeasure is that it does not require modifying the runtime application code.

A technique that hides the secret-dependent behavior accessible via the page fault side channel is to enforce the use of large pages for enclave memory, greatly reducing the resolution of the side channel [17].

Making Enclaves Timing Sensitive

Several techniques have been proposed to make enclaves sensitive to the side channel leakage measurements. One strategy uses transactional memory [15], [16]. Traditional locking protects against conflicting simultaneous access to a resource by having each task hold a “lock” for the duration of the operation. With transactional memory, an application may designate certain memory operations as belonging to an atomic transaction. The transaction can either complete, in which case all the operations in the transaction will be visible to other tasks, or the transaction may “abort”, and none of the operations will be visible to other tasks. In the case of an aborted transaction, the important result is that none of the memory operations that might normally expose side channel information are applied to the shared memory resource. Transactional memory is useful for protecting SGX applications because common measurement techniques like forcing a page fault also causes any transaction referencing those pages to abort. When an enclave observes too many aborts that indicate a likely attack, the enclave can refuse to run, lest it potentially leak confidential data. To deploy this countermeasure, an application must be recompiled with suitable transactions inserted into the program code.

A variation of the transactional memory protection updates a counter within a transaction, to detect when the enclave has been suspended. This requires less extensive analysis of the enclave program than protection using transactions alone, but incurs an overhead to maintain the counter [7].

Side Channel Defenses in Fortanix Solutions

Fortanix Self-Defending Key Management Service™ (SDKMS), secured with Intel® SGX, delivers unified key management and hardware security module (HSM) capabilities.

In the Fortanix SDKMS, all cryptographic algorithms are hardened against side channel attacks. Varying protection strategies are used for different algorithms:

- Use of hardware-intrinsic cryptographic instructions that exhibit the same memory access and timing characteristics regardless of input data.
- Manual alignment of critical lookup tables to fit entirely within a cache line, so that secret-dependent access to the lookup table does not leak via side channels. When lookup tables do not fit within a cache line, multiple tiered lookup tables or masking is employed to ensure that the conditional probability of any externally-observable memory access is the same for all secret values.
- Blinding of large integers so that the actual values computed by any big-integer arithmetic operation do not correlate with any secret value.

When appropriate, non-cryptographic algorithms are also protected against side channel attacks. For example:

- The time taken to verify passwords and API keys is ensured to be independent of the secret password or API key.
- Processing of potentially sensitive data (e.g. decoding REST API input data) employs algorithms that do not leak information.

Fortanix Runtime Encryption solutions protect applications and their data in use. Within Fortanix’s Runtime Encryption solutions, cryptographic primitives internal to the library are protected using techniques similar to those used in SDKMS.

Conclusion

Side channel attacks are a very real concern in any digital system that operates on sensitive data. As with many aspects of digital security, new attacks are continually under development, and systems must be constantly updated to remain secure. SGX offers the power to run most x86 applications in an enclave, protect applications' data against unauthorized memory reads. To ensure that enclaves are secure in practice, great care is needed, including the care to avoid leaking secrets through side channels. Recently published literature identifies side channels that are present in the SGX technology, but whether there is information available from the side channel and whether the side channel is accessible to an attacker are properties of each application and deployment, and not of SGX itself. Fortunately, applications can use countermeasures to control these parameters. Fortanix solutions have been designed with various defenses, leveraging expertise in hardware-based security and cryptography, to protect against side channel attacks. The security experts at Fortanix have significant experience dealing with side channels and are happy to help identify appropriate solutions to your security challenges.

Bibliography

- [1] B. W. Lampson, "A Note on the Confinement Problem."
- [2] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds," in Proceedings of the 16th ACM Conference on Computer and Communications Security, New York, NY, USA, 2009, pp. 199–212.
- [3] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-VM Side Channels and Their Use to Extract Private Keys," in Proceedings of the 2012 ACM Conference on Computer and Communications Security, New York, NY, USA, 2012, pp. 305–316.
- [4] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing," in 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, 2017, pp. 557–574.
- [5] Y. Xu, W. Cui, and M. Peinado, "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems," in Proceedings of the 2015 IEEE Symposium on Security and Privacy, Washington, DC, USA, 2015, pp. 640–656.
- [6] J. V. Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, "Telling Your Secrets without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution," in 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, 2017, pp. 1041–1056.
- [7] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, "Detecting Privileged Side-Channel Attacks in Shielded Execution with Déjà Vu," in Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, New York, NY, USA, 2017, pp. 7–18.
- [8] "Meltdown and Spectre." [Online]. Available: <https://meltdownattack.com/>.
- [9] "Reading privileged memory with a side-channel." [Online]. Available: <https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html>.
- [10] "Intel® Software Guard Extensions Developer Guide." .
- [11] S. Sasy, S. Gorbunov, and C. W. Fletcher, "ZeroTrace : Oblivious Memory Primitives from Intel SGX," 549, 2017.
- [12] M. Hähnel, W. Cui, and M. Peinado, "High-Resolution Side Channels for Untrusted Operating Systems," in 2017 USENIX Annual Technical Conference (USENIX ATC 17), Santa Clara, CA, 2017, pp. 299–312.
- [13] F. Brasser et al., "DR.SGX: Hardening SGX Enclaves against Cache Attacks with Data Location Randomization," ArXiv170909917 Cs, Sep. 2017.
- [14] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware Guard Extension: Using SGX to Conceal Cache Attacks," ArXiv170208719 Cs, Feb. 2017.
- [15] M.-W. Shih, S. Lee, and T. Kim, "T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs.
- [16] D. Gruss, J. Lettner, and F. Schuster, "Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory."
- [17] Y. Fu, E. Bauman, R. Quinonez, and Z. Lin, "SGX-LAPD: Thwarting Controlled Side Channel Attacks via Enclave Verifiable Page Faults," in Research in Attacks, Intrusions, and Defenses, 2017, pp. 357–380.