The Security of Intel SGX for Key Protection and Data Privacy Applications

Yehuda Lindell*†

August 16, 2018

1 Introduction

Intel Software Guard Extensions (Intel® SGX) [1] is an Intel technology for application developers who are seeking to protect select code and data from disclosure or modification. Intel SGX enables the execution of security-critical application code, called enclaves, in isolation from the untrusted system software. Protections in the processor ensure that a malicious OS cannot directly read or modify enclave memory at runtime. SGX includes a mechanism called *sealing* for encrypting and authenticating an enclave's data for persistent storage (this method is limited to AES-128 and longer keys cannot be used). Processors are also equipped with certified keys that can issue remotely verifiable attestation statements on enclave software configuration. These SGX mechanisms (isolation, sealing, attestation) enable the development of applications and online services with improved security. The SGX architecture is especially useful in cloud computing applications, since data and computation can be outsourced to an external computing infrastructure without having to fully trust the cloud provider and the entire software stack.

Since launching Intel SGX on 6th Generation Intel Core processors in 2015, there has been considerable work carried out by the security research community, studying various usage models and the security of Intel SGX. In this white paper, we describe some of this work which has focused on side-channel attacks on SGX.[1] As we will show, these attacks are extremely effective at extracting private data from inside an SGX enclave. The most recent of these attacks even extracted the attestation key from an Intel enclave. As a result, we argue that the community's understanding of the potential side channels and their ramifications is such that SGX should *not* be currently used for security-critical applications where *privacy* is required.

The typical response by proponents of SGX to the existence of these side-channel attacks is that they can be prevented by writing side-channel-proof code, and this is the responsibility of the developer. As we will show below, this is extremely hard, if not impossible. This position is supported by the demonstration of effective side-channel attacks on implementations in SGX that were designed to be side-channel proof and implemented by world experts in this field; see more below.

*Chief Scientist, Unbound Tech (previously Dyadic Security), Israel. Email: yehuda.lindell@unboundtech.com

†Professor of Computer Science, Bar-Ilan University, Israel. Email: lindell@biu.ac.il

[1] We stress that this white paper does not attempt to provide a detailed study of side-channel attacks on SGX. Rather, our aim is to discuss these attacks on a high level and their ramifications to the usage of SGX. For further research, see the references.

2 Background – Side-Channel Attacks

The idealized view of cryptography is that algorithms behave as a black-box, and the only adversarial attacks possible are *external* to the algorithm. For example, in order to attack an encryption scheme, we assume that the attacker can see ciphertexts, and may even be able to ask for certain messages to be encrypted (called a chosen-plaintext attack) or even to tamper with ciphertexts and obtain decryptions of these tampered ciphertexts (called a chosen-ciphertext attack). This is the way that definitions of security for encryption, digital signing, and so on, are all formulated; see, for example, [20]. This view of cryptography is natural since the internal workings of the encryption/decryption or signing algorithms do not seem to be relevant vectors of attack for adversaries. This is especially the case when the cryptographic algorithm is computed on a remote machine or inside secure hardware (historically, a smartcard or HSM, and today extended to trusted enclaves, SGX and so on), since in such cases it seems that an adversary can only learn information by providing inputs and obtaining outputs. Unfortunately, this idealized view of cryptography was obliterated in 1996 due to the groundbreaking work of Kocher [21], who showed that just being able to measure the *time* that an algorithm runs is sufficient to read out the secret key. In 1999, Kocher along with others discovered that the *power trace* of the device computing the cryptographic operation leaks the secret key since different operations consume different amounts of power, and thus one can distinguish the operations caused by different bit values of the secret key [22]. These attacks are called *side-channel attacks*, since the adversary is assumed to have access to additional (side) information about the encryption/decryption/signing process, beyond what can be obtained from pairs of input/output. Although it seems hard to believe that these attacks actually work, they have proven to be devastatingly powerful, and so all implementations of cryptography must be *side-channel proof* in order to be considered secure. Stated differently, if the implementation of a cryptographic algorithm is side-channel secure, then we can revert back to the idealized view where our definitions and proofs of security are valid.

The problem of implementing an algorithm so that it is side-channel secure is very non-trivial. If the attacker has physical access to the device, then the device must ensure that nothing can be leaked through measurements of power and more. This is difficult, since new side channels are continually being discovered (it has even been shown that secret keys can be leaked by just recording the noise emitted by a machine carrying out secret-key operations [15]). If the algorithm is run in isolation on a machine and the attacker has no physical access to it, then it suffices for the implementation to be constant-time (i.e., its running time does not depend on the secret key). However, in many cases, different processes run on the same machine. Surprisingly, it was shown that secrets can be leaked due to the fact that such processes share *physical resources* on the machine. Most notably, it was shown that by monitoring cache usage, it is possible to effectively "read the memory" of another process [34]. This is possible since the time it takes to read from memory depends on the cache contents, and there is dependence on this between different processes. To further exacerbate the situation, in modern computing environments that are virtualized, multiple different virtual machines almost always run on the same machine, opening up the possibility of cache attacks (and others) between these different machines. This was first demonstrated in practice in [28] who showed how to carry out such an attack in practice on Amazon's cloud. Later attacks demonstrate this capability, even if the attack VM and the victim VM are running on *different cores* of the same machine [6, 7] and even if they are running on *different CPUs* on the same physical machine [18, 19]. Importantly, these attacks do not require physical access to the machine running the victim code, nor even special rights on the machine.

However, they do require the capability of achieving "co-location", which means that the attacker can position its VM on the same physical machine as the victim. Surprisingly, it has been repeatedly demonstrated that this capability is very practical on real modern clouds; see [19] for just one example.

We conclude this very brief background by stressing that the issue of side-channel attacks is a very serious one, with new attacks and attack surfaces being discovered all the time. We have only scratched the surface in our description above.

3 Software Side-Channel Attacks on SGX

3.1 Demonstrated Attacks

In [3, Slides 109–121], Intel outlines the security model of SGX. The stated goals of SGX are: **(a)** to ensure the integrity of code running in an enclave instance (correct computation), **(b)** to provide confidentiality of code and data in an enclave instance (private computation), and **(c)** to provide isolation between all enclave instances. Furthermore, these security goals should be achieved in the presence of an *unprivileged software adversary*, a *system software adversary* and a *startup code/MMS* adversary (see Slides 112–114 in [3] for an exact definition of these adversaries). However, as stressed in [3, Slide 115], *SGX does not defend against side-channel adversaries* who may gather power statistics, gather performance statistics including platform cache misses, gather branch statistics via timing, or gather information on page accessed via page tables. Rather, it is the responsibility of the software developer to ensure that code running in an enclave is not vulnerable to side channel attacks, as stated in [4]:

> "Preventing side channel attacks is a matter for the enclave developer. Intel makes this clear in the security objectives for Intel SGX..."

Unfortunately, as we will show below, this is an extremely difficult – if not impossible – task. In addition, many are proposing uses of Intel SGX without taking into account the dangers of existing (and new) side channels and their ramifications on the security of the proposed solutions. This is due to the fact that side-channel attacks are often thought to require *physical access* by the attacker to the victim machine. However, there are extremely powerful side channels that are based on software only, and some of these only require that the malicious attack software is *co-located* on the same physical machine as the secure enclave (meaning that the attacker's VM runs on the same physical machine as the victim VM).

Page-based side channels. In one of the first attacks on SGX-based solutions [37], it was shown that an attacker can enforce page faults during a secure enclave's execution. Then, by observing the page-level memory access pattern, the malicious OS is able to learn which page was accessed by the enclave program. This was utilized to attack the Haven system [8], which was designed to protect applications from an untrusted cloud with SGX. The attack code was able to extract private data from Haven with very high accuracy. This attack can be carried out as long as the attacker is able to infect the VM running the application that uses SGX to protect its private data. Thus, even though data is only ever opened and processed inside SGX, it can still be extracted by malware that has infected the running VM.

Cache-based side channels. All cache-based side-channel attacks are based on the fact that cache misses of a protected application can be identified. The most basic cache attack utilizes the property that secret data values influence the data access pattern and thus cache hits or misses. As such, the secret can be extracted by just observing the access time of each cache access. More advanced cache attacks called evict-and-time, prime-and-probe, flush-and-reload, and more) all work by the attacker influencing the contents of the cache and measuring this effect. Surprisingly, cache attacks have been extremely effective; in particular, they are extremely effective at extracting RSA and ECC private keys and AES symmetric keys. See [14] for more information.

Cache attacks were carried out very effectively on cryptographic functions running in SGX. For just a few examples, [9] used a cache-attack to extract an entire RSA-2048 key during RSA decryption, and to detect specific human genome sequences during genomic indexing. The attack can be carried out if the attacker can run its attack code on the same processor as the victim application. Thus, this merely requires *co-location* (however, the co-location must be on the same core, since it utilizes the L1 cache which is specific to each core, and not the L3 cache which is joint to all processors on the machine). The attack of [9] is very powerful since it does not require interrupting the victim code. That is, the attacker can run both the victim enclave and its own process uninterrupted in parallel. As such, the attack is very hard to detect and mitigate. The method of [9] is very effective, and they extracted the RSA private key from an SGX enclave with just 300 RSA decryptions. In [16, 26], similar attacks were demonstrated for recovering AES keys. These attacks require the attacker to have root access to a Linux OS running SGX. The attacks of [26] require only 10s or 100s of AES operations, and are effective even against software implementations (running inside SGX) that include countermeasures against cache attacks. A co-location attack that only requires that the attacker code be on the same *machine* as the SGX, but not necessarily the same CPU, was demonstrated by [29] (these attacks utilize the L3 cache and so the attack can be run on any processor on the same physical machine as the victim SGX). The attack of [29] is able to extract a complete 4096-bit RSA key from the *mbedTLS* RSA implementation (that uses constant-time decryption as a protection against side-channel attacks) with just 11 RSA decryptions and 5 minutes of local computation. Furthermore, [29] demonstrate that they can even run their attack in another SGX enclave on the machine, in order to completely avoid detection. We stress that the attacker in [29] does not need any special privileges, and the attack works as long as the attacker can run code on the same physical machine. This was demonstrated by running the attack between Docker instances running on the same machine (where the victim Docker used SGX for storing and computing with the RSA private key.)

In light of the many works on attacking and defending against side channels in SGX, a broad study of the types of attacks and their effectiveness was carried out in [36]. The aim of [36] was to to identify all weaknesses related to the memory management of the enclaves. As such, the paper first presents 8 vectors of side-channel attacks related to address translation caches in CPU hardware. Next, they present a new attack called "sneaky page monitoring" (SPM) which significantly reduces the number of interrupts (caused by page faults) required to carry out the attack (which works even if all secret-dependent branches are located in the same page). The attack works by setting and resetting the pages accessed flag in the page table entry. This attack was demonstrated on EdDSA in Libgcrypt (v1.7.6), which is supposed to be cache-attack resilient.[2] Finally, the paper

[2]To be more exact, the Libgcrypt implementation provides side-channel protection for the EdDSA secret key, but not for the randomness used in generating the signature. However, given a signature and the randomness, it is trivial to recover the EdDSA secret key. Thus, the implementation is vulnerable. This is another indication of how difficult

presents an attack that works even if one uses a mechanism that mixes sensitive code and data into the same page.

A new and very powerful cache attack, called *MemJam* [27] utilizes the fact that it is (almost) impossible to write truly time independent code. In particular, in order to prevent cache attacks, code is written so that the memory access pattern is completely independent of the secret key value. However, as demonstrated by [27], this is actually highly dependent on the specific architecture where the code is run. The attack designed by [27] was able to extract an AES secret key from the Intel constant-time implementation called `Safe2Encrypt_RIJ128` that is part of the Intel IPP (Integrated Performance Primitives) library. We stress that this implementation is considered to feature true cache constant-time behavior, and is part of the Linux SGX SDK. Nevertheless, the AES secret key was extracted from an SGX enclave running this implementation. The attacker in the MemJam attack does not need any special privileges, and is only required to be co-located with the victim enclave on the same processor.

Energy management attack. A new and very powerful attack, called CLKSCREW, was demonstrated in [32]. This attack is carried out using the energy management tool, and was demonstrated on commodity ARM devices (e.g., a Nexus 6 phone) to extract secret keys from ARM TrustZone and to escalate privileges to enable an attacker to load self-signed code into Trustzone. The attack uses the frequency regulator to change the voltage in a way that may affect the output of a computed circuit. This is possible since most systems today allow software to control the frequency and voltage of the hardware platform (this is aimed to help extend battery life). As shown in [32], this capability can be used by the attacker to inject faults into the running code, which is devastating since such fault attacks are known to enable the extraction of cryptographic keys with great ease. For example, in order to extract AES keys, it suffices to induce a one-byte random corruption to the intermediate state of an AES round in the decryption process. Then, given the faulty plaintext produced by this, and the expected one, the key stored within Trustzone was obtained with only 12 minutes of computation by [32]. We remark that the CLKSCREW attack was not demonstrated specifically against SGX. However, as the researchers argue, the attack is relevant for all hardware that enables energy management (which is essentially all modern platforms). We note that AWS does provide processor state control for EC2 instances [5] as required for the CLKSCREW attack, but stress that the attack has not been demonstrated on this specific platform. Thus, the question of the attack's relevance to SGX in real cloud environments is still open.

Speculative execution. Recently, speculative execution carried out by modern processors has been shown to provide a very powerful side-channel in the form of the Meltdown and Spectre attacks [23, 25]. Speculative execution is another example of a failure of processors to provide *isolation* between different processes running. The Spectre attack (which has no known solution, but only very partial mitigations) works by influencing the branch prediction of the CPU in order to have victim code execute instructions that leave a trace on the cache. Since SGX enclaves use the same speculative execution process as all other processes running on the machine, the attack is effective against SGX as well. This was demonstrated in [10], who showed how to read register values and general memory out of an enclave. The attack was shown against enclaves written using the Intel's official SDK [2].

it is to write side-channel proof code.

Recently, a new attack called *Foreshadow* was released [35]. This attack has the unique property that it can read private data out of an enclave, even if the code is perfectly side-channel safe. In fact, the Foreshadow attack was even used to extract the Intel attestation key from the enclave, completely breaking all integrity guarantees provided by SGX. Although the specific attack method has been patched by Intel, the existence of this side channel is of great concern, and it's clear that this is not the last word on the subject.

3.2 Defenses and Mitigation

Most known defenses are designed specifically to page-fault side-channel attacks. In [31], a compiler-based approach was proposed to transform cryptographic programs to hide page access patterns that may leak information. Shih et al. [30] proposed T-SGX which exploits Intel Transactional Synchronization Extensions (TSX) to prevent page faults from revealing the faulting address. Costan et al. [12] proposed a secure enclave architecture that is similar to SGX but resilient to both pagefault and cache side-channel attacks. Chen et al. [11] proposed DÉJÀ VU, a compiler-based approach to instrument enclave programs so that they can measure their own execution time between basic blocks in their control-flow graph, thus being able to detect attacks that induce many interruptions to the victim's execution. As discussed in [36], these approaches all fail to consider the full gamut of possible memory side-channel attacks, and thus fall short of being satisfactory defenses.

In a different direction, some have suggested architectural changes to cache organization. However, these approaches would require a radical change to current cache designs, which cannot be easily implemented in practice (and are certainly not relevant for today's systems). In particular, Intel processors with SGX extensions do not implement any countermeasures against cache side-channel attacks at the architectural level. Another approach to defeat information leakage via side channels is Oblivious RAM (ORAM) [13,33,34], which provides means to hide memory access patterns of programs by continuously shuffling and re-encrypting data as they are accessed in RAM memory, disk or from a remote server. The applicability of this solution was investigated in [9] who state that

> "While one could think of using similar techniques for cache protection, they are not directly applicable, as it is challenging to store ORAM internal state securely. Without hardware support this would require storing client state in a cache side-channel oblivious way, which is unfeasible given the small size of every cache line."

Thus, although ORAM could be used effectively for hiding the access patterns to main memory, it does not seem to be viable for hiding cache access patterns.

The CLKSCREW attack [32] is not just the latest in a known exploit genre. This attack opens the door to a whole new class of energy-management based attacks. According to the authors' analysis:

> "...there is unlikely to be a single, simple fix, or even a piecemeal fix, that can entirely prevent CLKSCREW style attacks. Many of the design decisions that contribute to the success of the attack are supported by practical engineering concerns. In other words, the root cause is not a specific hardware or software bug but rather a series of well thought-out, nevertheless security-oblivious design decisions."

The Meltdown [25] and Spectre [23] attacks on speculative execution are extremely troubling. First, there is no solution to these attacks, and proposed mitigations are very partial (especially

for Spectre). Second, these are the first attacks to ever use speculative execution as a side channel. Since they are so new, it is very likely that this new side channel will be utilized in many unexpected ways in the future.

We reiterate that the proposed mitigation that side-channel-proof implementation be used is highly problematic. First, this takes extremely rare and specialized expertise, and it is unreasonable to assume that this is available to companies developing products for SGX. Second, even the best implementations known, like `Safe2Encrypt_RIJ128`, have been found to be vulnerable to cache attacks [27]. Third, some attacks like Spectre and Foreshadow do not rely on the victim code being poorly written. Thus, this cannot be relied upon as a reasonable mitigation strategy for high-security applications.

3.3 Summary of Attacks and Capabilities

As can be seen from the above survey, software side-channel attacks are extremely effective at extracting secret data from within secure enclaves in general, and from SGX in particular. The different attacks require different capabilities; some are easier to mitigate but these are typically followed up by attacks that thwart the mitigation.[3] As we have seen, the two main alternatives to an attacker are *kernel privileges* or *co-location*:

- *Kernel privileges:* First and foremost, an attacker who has kernel privileges on the machine where SGX is running can carry out extremely effective cache attacks, with ease. This means that using SGX to achieve data privacy or protect secret keys in the presence of an "untrusted OS" or an "untrusted cloud" (or a cloud that receives a subpoena) is extremely problematic. Thus, although obtaining kernel privileges is not at all easy, it is important to stress that many are proposing SGX as a way to obtain security under the exact threats for which the attacker does have kernel privileges.

- *Co-location:* Second, co-location has been shown to be sufficient to extract private data and secret keys. This means that if an attacker can run its attack code on the same physical machine as the victim, then SGX is no longer effective for protecting data privacy. Initial attacks required the co-location to be on the same processor as the victim, but these attacks have been extended so that the co-location only needs to be on the same physical machine (but the attack code can run on any core). Although co-location may intuitively seem like a difficult task, it has been shown that it is can be achieved very effectively in modern public cloud environments. Thus, even if the cloud provider is trusted, this is not sufficient.[4]

Cache side-channel attacks do take expertise to construct. However, once an attack has been implemented, it can be packaged as a tool that can be used by anyone. It is important to also recall that many of the attacks require very little time to extract secret keys, and are so extremely effective. In addition, the published attacks are *proofs of general capability*, and not comprehensive

[3] The exception is mitigation strategies that rely on completely modifying the underlying architecture. However, these are typically not feasible, and certainly will not happen in the short term.

[4] We stress that when we say that the cloud provider is trusted, this includes that we trust that they are not breached, that no internal administrator is bribed or blackmailed, and that they are not required to act upon subpoena. It is clear that no large commercial cloud provider today would willingly attack its own customers. However, the real threat landscape is much more complex, and the real threats are breaches, insiders and even governments.

attacks on all possible products and implementation. Due to the rich variety of effective attacks, the assumption should be that data privacy is not afforded via software run inside SGX.

We stress that AES encryption and decryption run inside SGX using the AES-NI hardware instruction has not been shown to be vulnerable. These instruction sets were designed to prevent cache attacks, and so far have withstood attack. Thus, AES operations only (when run inside AES-NI) seem to be safe.[5]

4 The Security of SGX for Cloud Applications

4.1 Secret-Key Protection – an SGX-Based HSM

One proposed use of SGX is for cryptographic key management and protection. This application is very attractive since secret keys are vulnerable while in use, and once stolen all cryptographic defenses are completely bypassed. Historically, secure hardware has been used to protect secret keys even while in use (this hardware is generally called an HSM, or Hardware Security Module). In an HSM, the entire cryptographic computation takes place inside a secure environment that includes protections against physical attacks. Defense against physical side-channels (like power and differential-power analysis [22]) is a part of the HSM design and is verified manually as part of the certification process. However, in modern environments, physical access by an attacker to a victim machine is very hard to achieve, and so physical protection may be less critical. For example, modern clouds have heavily secured data centers that are well protected. Nevertheless, HSMs are still attractive from a security perspective since they only run the specific cryptographic operations specified in their API. As such, it is not possible to load malware onto an HSM, to breach its OS, or to co-locate another VM on the same machine. SGX enclaves therefore sound like a perfect alternative to HSMs in cloud-like environments. If the enclave is indeed secure, and physical access is not a concern, then an SGX-based HSM would be a cheap, easy to manage, and flexible solution to the threat of secret key theft.

Despite its intuitive appeal, a large body of research already exists that shows that cryptographic keys are just not safe inside SGX. As we have shown, co-location on the same physical machine alone suffices to extract secret RSA keys, ECC keys, and AES keys (when the latter is implemented in software). This is also true of implementations that were designed (and widely believed) to be resilient to cache attacks. Currently, the only cryptographic operations that seem to be safe inside SGX are AES encryption and decryption using AES-NI hardware-based instructions only.

4.2 Data-in-Use Encryption via SGX

Another attractive application of SGX is for data-in-use encryption. In this scenario, data is encrypted and then used without ever decrypting it. The methods that currently exist for data-in-use encryption are either impractical for most applications (like fully homomorphic encryption) or provide a weak level of security and limited functionality (like format and order-preserving encryption).[6] Thus, SGX sounds like the perfect solution to this problem: all data is encrypted under a key stored inside SGX. Then, in order to process the data, it is read into the SGX, decrypted

[5] Note that the AES implementation inside the official Intel SGX SDK for Linux does not use AES-NI and implements AES in software [16, Section 4.2]. It is unclear why this is the case.

[6] Secure multiparty computation can also be used for the purpose of data-in-use encryption. However, it requires interaction and so achieves a slightly different effect.

internally, processed, and the result is finally encrypted before returning it. In this way, data is never decrypted outside the SGX and thus never vulnerable. One very notable use case for this is an encrypted SQL database in the cloud, where the cloud provider never sees plaintext data. This can be achieved by the client uploading only encrypted data to the cloud, and having the SGX enclave be the only entity to know the decryption key.

Once again, despite this sounding like the perfect solution, it is in fact very problematic. First, only AES-NI encryption can be used securely (since otherwise, the decryption key could be extracted and the attacker could then steal all the data and decrypt it). This means that the clients uploading the data must all hold the same AES key (and so, public-key encryption cannot be used). More importantly, cache attacks are not limited to extracting cryptographic key, and are actually *far more effective* against standard data processing. This is due to the fact that cryptographic operations are far more limited in scope, and so implementations attempt to be resilient against cache attacks (albeit, unsuccessfully, as we have seen). In contrast, more general applications (like SQL) operate in an extremely data-dependent manner. Thus, cache attacks can be used to extract private data. Finally, in such applications, external memory accesses have also been shown to leak a lot of information. This can be solved by using oblivious RAM. However, this severely impacts performance, and can only practically prevent leakage from external memory, but not from the cache.

4.3 Data Privacy

Another application, that is closely related to that of data-in-use encryption, considers multiple *different entities* with private data who wish to carry out a joint computation on their encrypted data. This is the classic setting of secure multiparty computation. However, it can be achieved far more easily using SGX. Specifically, all parties can send their data encrypted to an enclave, who can carry out the computation and output the result. Clearly, this application suffers from the same problems as that of data-in-use encryption.

4.4 Trusted Applications

SGX includes a mechanism called attestation, that can be used to ensure that correct code is running. This can be used to prevent malware from tampering with an application, and thus can significantly raise the security of networks. This mechanism relies on the Intel attestation procedure. The Foreshadow attack [35] that can successfully extract the Intel attestation key from an enclave brings this use of SGX into question. Although the specific threat of [35] has been patched, we believe that the lack of isolation of SGX from the processor means that it cannot be trusted in such cases.

5 Discussion and Conclusions

The current state of affairs regarding the *privacy* of data processed inside SGX is very poor. Software-based side channel attacks have proven to be extremely effective at extracting secrets from inside SGX. This is especially the case for the use of SGX to protect cryptographic material, with effective attacks shown against RSA, ECC and AES, in some cases even for implementations that are designed to be resilient to cache attacks. These attacks require malicious code to run on the same physical machine as the SGX enclave, but this has been shown to be not difficult

to achieve in practice. Furthermore, although designing these attacks takes some expertise, they are not overly difficult and can be run by anyone who obtains the attack code. In our opinion, this means that SGX is *not* suitable as an "HSM replacement", as is now being proposed. This is especially the case as even expertly written code that is supposed to be side-channel resistant has been broken inside SGX. As we have discussed, a similar problem arises for data-in-use encryption and data privacy applications.

We conclude with a paragraph, taken word for word from [36]:

> *Over the years, we observe that many side-channel studies follow a similar pattern: a clever attack is discovered and then researchers immediately embark on the defense against the attack. However, in retrospection, most defense proposals fail to consider the bigger picture behind the demonstrated attacks, thus they are unable to offer effective protection against the adversary capable of quickly adjusting strategies, sometimes not even meaningfully raising the bar to the variations of the attacks. The ongoing research on SGX apparently succumbs to the same pitfalls. We hope that our study can serve as a new starting point for rethinking the ongoing effort on SGX security, inspiring the follow-up efforts to better understand the fundamental limitations of this new TEE and the ways we can use it effectively and securely.*

Perhaps the greatest concern is that the attack space is constantly moving. At the very least, it will be years until we fully understand the security of SGX. This puts SGX currently in the cat-and-mouse zone for data privacy,[7] where the security guarantees are not sufficient for security-critical applications like cryptographic key protection.

Acknowledgements

We thank Ariel Nof for helpful discussions.

References

[1] Intel SGX. https://software.intel.com/sgx.

[2] Intel SGX SDK. https://github.com/intel/linux-sgx

[3] Intel Software Guard Extensions (Presentation). https://software.intel.com/sites/default/files/332680-001.pdf

[4] Intel SGX and Side-Channels. https://software.intel.com/en-us/articles/intel-sgx-and-side-channels. March 16, 2017.

[5] AWS. *Processor State Control for Your EC2 Instance.* http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/processor_state_control.html. Retrieved 5 Dec. 2017.

[6] G.I. Apecechea, M.S. Inci, T. Eisenbarth and B. Sunar. Fine Grain Cross-VM Attacks on Xen and VMware. In the *IEEE Fourth International Conference on Big Data and Cloud Computing* (BDCloud), pages 737–744, 2014.

[7] The "cat and mouse" zone is one where solutions are continually being built and broken, in an iterative cycle.

[7] G.I. Apecechea, T. Eisenbarth and B. Sunar. S$A: A Shared Cache Attack That Works across Cores and Defies VM Sandboxing - and Its Application to AES. In the *2015 IEEE Symposium on Security and Privacy*, pages 591–604, 2015.

[8] A. Baumann, M. Peinado and G.C. Hunt. Shielding Applications from an Untrusted Cloud with Haven. In the *11th USENIX Symposium on Operating Systems Design and Implementation* (OSDI), pages 267–283, 2014.

[9] F. Brasser, U. Mller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A. R. Sadeghi. Software Grand Exposure: SGX Cache Attacks Are Practical. ArXiv preprint https://arxiv.org/pdf/1702.07521.pdf, 2017.

[10] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin and T.H. Lai. SgxPectre Attacks: Leaking Enclave Secrets via Speculative Execution. ArXiv preprint https://arxiv.org/pdf/1802.09085.pdf, 2018.

[11] S. Chen, X. Zhang, M. Reiter, and Y. Zhang. Detecting Privileged Side-Channel Attacks in Shielded Execution with Déjà Vu. In *12th ACM Symposium on Information, Computer and Communications Security*, 2017.

[12] V. Costan, I. Lebedev, and S. Devadas. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *25th USENIX Security Symposium*. USENIX Association. 2016.

[13] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. Journal of the ACM, 43(3), May 1996.

[14] Q. Ge, Y. Yarom, D. Cock and G. Heiser. A Survey of Microarchitectural Timing Attacks and Countermeasures on Contemporary Hardware. In *IACR Cryptology ePrint Archive*, report 2016/613. https://eprint.iacr.org/2016/613.pdf

[15] D. Genkin, A. Shamir and E. Tromer. RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis. In *CRYPTO 2014*, Springer (LNCS 8616), pages 444–461, 2014.

[16] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller. Cache Attacks on Intel SGX. In *EUROSEC 2017*.

[17] M. Hähnel, W. Cui, and M. Peinado. High-Resolution Side Channels for Untrusted Operating Systems. In *2017 USENIX Annual Technical Conference* (USENIX ATC 17). USENIX Association, Santa Clara, CA, 299312.

[18] M.S. Inci, T. Eisenbarth and B. Sunar. Cross Processor Cache Attacks. In the *11th ACM on Asia Conference on Computer and Communications Security* (AsiaCCS), pages 353–364, 2016.

[19] M.S. Inci, B. Gülmezoglu, G. Irazoqui, T. Eisenbarth and Berk Sunar. Cache Attacks Enable Bulk Key Recovery on the Cloud. In *CHES 2016* Springer (LNCS 9813), pages 368–388, 2016.

[20] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, 2nd Edition*. Chapman and Hall/CRC Press, 2014.

[21] P.C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO 1996*, Springer (LNCS 1109), pages 104–113, 1996.

[22] P.C. Kocher, J. Jaffe and B. Jun. Differential Power Analysis. In *CRYPTO 1999*, Springer (LNCS 1666), pages 388–397, 1999.

[23] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, Y. Yarom. Spectre Attacks: Exploiting Speculative Execution. ArXiv preprint https://arxiv.org/pdf/1801.01203.pdf, 2018.

[24] 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 the *26th USENIX Security Symposium*, 2017.

[25] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom and M. Hamburg. Meltdown. ArXiv preprint https://arxiv.org/pdf/1801.01207.pdf, 2018.

[26] A. Moghimi, G. Irazoqui, and T. Eisenbarth. CacheZoom: How SGX Amplifies the Power of Cache Attacks. In *CHES 2017*, Springer (LNCS 10529), pages 69–90, 2017. (ArXiv preprint https://arxiv.org/pdf/1703.06986.pdf.)

[27] A. Moghimi, T. Eisenbarth and B. Sunar. MemJam: A False Dependency Attack against Constant-Time Crypto Implementations. https://arxiv.org/pdf/1711.08002.pdf, November 21, 2017.

[28] 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 *ACM Conference on Computer and Communications Security* (ACM CCS), pages 199–212, 2009.

[29] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA, 2017.

[30] M.W. Shih, S. Lee, T. Kim, and M. Peinado. T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs. In Proceedings of the *2017 Annual Network and Distributed System Security Symposium* (NDSS), 2017.

[31] S. Shinde, Z. Chua, V. Narayanan, and P. Saxena. Preventing Page Faults from Telling Your Secrets. In the *11th ACM Symposium on Information, Computer and Communications Security*. 2016.

[32] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management. In the *26th USENIX Security Symposium (USENIX Security 17)*. 2017.

[33] S. Tople, H. Dang, P. Saxena, and E. C. Chang. PermuteRam: Optimizing oblivious computation for efficiency. http://www.comp.nus.edu.sg/ shruti90/papers/permuteram.pdf.

[34] E. Tromer, D. A. Osvik, and A. Shamir. Efficient Cache Attacks on AES, and Countermeasures. *Journal Cryptology*, 23(1), 2010.

[35] J, Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T.F. Wenisch, Y. Yarom and R. Strackx. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In the *Usenix Security Conference*, 2018. https://foreshadowattack.eu

[36] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang and C. A. Gunter. Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX. In the *ACM Conference on Computer and Communications Security* (ACM CCS), 2017.

[37] Y. Xu, W. Cui, and M. Peinado. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In the *36th IEEE Symp. on Security and Privacy*, 2015.