INTERNATIONAL JOURNAL OF INFORMATION TECHNOLOGY RESEARCH STUDIES (IJITRS)
(Open Access, Double-Blind Peer Reviewed Journal)
ISSN Online: 3139-3209



Confidential Computing with Trusted Execution Environments: Architectures, Attacks and Applications

Win Mathew John

Associate professor, PG Department of Computer Applications, Marian College Kuttikkanam (Autonomous), Kerala, India

Article information

Received: 20th June 2026
Received in revised form: 24th July 2026
Accepted: 27th August 2026
Available online: 30th October 2026

Volume: 2
Issue: 4
DOI: https://doi.org/10.63090/IJITRS/3139.3209.0036

Abstract

Encryption protects data on disk and on the network. A program must still decrypt its data to compute on it, and at that moment whoever runs the machine can read it. Confidential computing tries to close this gap with hardware. A trusted execution environment isolates code and data from the operating system, the hypervisor and the operator, and remote attestation lets a distant party check what was loaded. This article reviews the published research on that idea. We describe the two designs now in use, process-level enclaves and confidential virtual machines, with the hardware families behind them. We then go through what attestation proves, what it costs to run unmodified software in an enclave, and the attack literature grouped by class: transient execution, controlled channels and interrupts, fault injection, weaknesses in memory encryption, and abuse of the host interface. Applications to machine learning follow, with a short comparison against homomorphic encryption. This is a review and no new experiment is reported. Our reading is that the trust model is narrower than the name suggests.

Keywords:- Confidential Computing, Trusted Execution Environment, Remote Attestation, Enclave, Confidential Virtual Machine, Side-Channel Attack, Privacy-Preserving Machine Learning

.

I. INTRODUCTION

A hospital that rents a cloud server can encrypt its records on disk and again on the wire. Protecting them during use is harder to arrange. With ordinary hardware the records sit in plain memory while a program works on them, and the cloud user is left relying on the provider's staff and on its whole software and hardware platform not to expose the data [1]. Confidential computing is the attempt to remove that reliance by extending hardware-enforced protection to data while it is in use [2].

The mechanism is the trusted execution environment, or TEE. Mainstream instruction sets have gained features that create protected isolates, offering integrity and confidentiality to the code and data inside them on a machine whose other software is not trusted [3]. The owner of a workload no longer has to believe the operating system or the hypervisor. What the owner must believe instead is the processor, its firmware and the company that made both. That set is smaller than before. It still has to be trusted, and the rest of this review is largely about how far that trust has held up.

This article reviews published, peer-reviewed work on TEEs. We ran no experiment of our own. Three questions run through it: what the hardware isolates, what a relying party can conclude from remote attestation, and how much of the promise is left after a decade of published attacks. Attacks are described at the level of class and consequence, as in earlier surveys [4], [5].

II. TWO DESIGNS AND FIVE HARDWARE FAMILIES

The first design isolates part of a process. Intel's Software Guard Extensions (SGX) added instructions and a software model for running a region of an application, the enclave, in isolation [6]. An application built this way can protect itself from a potentially malicious operating system or hypervisor [7], and the hardware is also meant to defend against physical attacks such as memory probes [1]. Existing applications do not run in an enclave as they stand, a cost Section IV returns to.

The second design isolates a whole virtual machine. AMD's Secure Encrypted Virtualization (SEV) encrypts a guest's memory so that the hypervisor cannot read it. An early analysis, made from the documentation before silicon was on sale, found that the first version left the control block and the registers exposed and gave the hypervisor control of the nested page tables [8]. Two later versions, SEV-ES and SEV-SNP, were released to deal with the unencrypted state and the missing integrity protection [9]. Intel's Trust Domain Extensions (TDX) take the same route, with virtual machines whose processor state and memory are encrypted and integrity-protected [10]. The second design exists because enclaves proved awkward: confidential virtual machines were introduced to ease the programming and deployment difficulties of enclaves [11]. Figure 1 sets the two side by side.

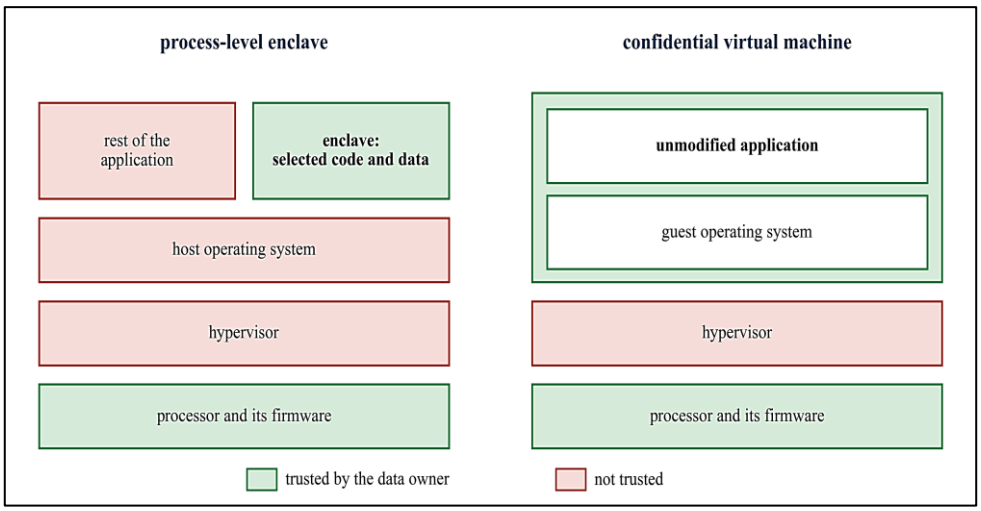


Fig. 1: What the data owner must trust in each design. Schematic drawn by the authors from the descriptions in [7], [10], [11].

Two families come from outside the x86 server line. Arm TrustZone is a system-wide security technology found on Arm application processors and on the newer Arm microcontrollers, and the survey by Pinto and Santos reviews the trusted execution environments and virtualisation systems built on it, together with their main weaknesses [12]. On RISC-V there is no single vendor design. Keystone is an open-source framework for building customised TEEs from simple hardware abstractions, namely memory isolation and a programmable layer beneath the untrusted operating system, and it runs on unmodified RISC-V hardware [13]. Table 1 lists the five families.

Table 1. The five hardware families, as characterised in the cited sources.

Family	Unit isolated	Characterisation in the source	Source
Intel SGX	Part of a process (enclave)	Application protects itself from a malicious operating system or hypervisor; existing programs need a runtime layer	[6], [7]
Arm TrustZone	System-wide secure environment	On application processors and newer microcontrollers; hosts TEEs and hardware-assisted virtualisation	[12]
AMD SEV, SEV-ES, SEV-SNP	Virtual machine	Memory encryption first; later versions address unencrypted state and missing integrity protection	[8], [9]
Intel TDX	Virtual machine	Encrypted processor state and memory, integrity protection, remote attestation	[10]
Keystone (RISC-V)	Enclave, customisable	Open-source framework; TEEs built from memory isolation and a programmable layer under the operating system	[13]

III. WHAT REMOTE ATTESTATION PROVES

Isolation alone does little for a remote user, who cannot see the machine. Attestation fills the gap. A third party can ask for an isolate to be launched on an untrusted machine and then know, with a high degree of assurance, that a given payload of code and data was loaded into a legitimate isolate with a particular configuration [3]. Ménétrey and colleagues review how SGX, TrustZone, SEV and the emerging RISC-V designs each implement the idea [14]. The protocols are intricate. When Sardar, Musaev and Fetzer wrote a formal specification of TDX attestation they found subtle discrepancies in Intel's documents, and only after resolving them did they prove, in a symbolic model, that their specification kept the secret confidential and the report authenticated [15].

A valid report has limits. It does not say that the attested code is well written. An analysis of eight open-source enclave runtimes found 35 flaws in how they sanitised data crossing the enclave boundary, some of them exploited to leak secret keys [16]. Attestation identifies what was loaded and not whether it is sound, so in our reading it offers no defence against flaws of this kind. Nor does a report prove that the hardware keys behind it are still secret. Foreshadow extracted keys from Intel's own architectural enclaves and used them to forge attestation responses [17]. On the Naples series of AMD EPYC processors, keys fundamental to SEV attestation were extracted through firmware flaws, and although AMD had fixed the flaws, the processors did not prevent the installation of earlier firmware versions [18].

Patching is where the scheme tends to break down in practice. Van Schaik and colleagues studied two commercial products built on SGX, a blockchain for private smart contracts and a Blu-ray rights-management player. Neither vendor was able to meet the security goals it had set out with, which the authors put down, with a careful 'presumably', to long update timelines and a manual update process [5].

IV. RUNNING EXISTING SOFTWARE, AND ITS COST

Few users want to rewrite an application for an enclave. Haven, by its authors' account the first system to run unmodified legacy applications under shielded execution, did so for SQL Server and Apache using SGX [1]. SCONE did the same for Docker containers through a secure C library interface, with user-level threading and asynchronous system calls to cut the cost of thread synchronisation and of calls out of the enclave, and its authors measured between 0.6 and 1.2 times native throughput [19]. Graphene-SGX argued that a full library operating system need not be ruinous. Its overheads ranged from matching a Linux process to less than a factor of two in most single-process cases [7].

Confidential virtual machines sidestep the porting problem, since the guest kernel comes along. They have costs of their own. An empirical comparison of SEV-SNP and TDX found effects on boot time, memory management and input/output, and it identified inefficiencies in how the two platforms switch context [11]. An earlier experience report, which built one publish/subscribe service on both SGX and SEV, describes trade-offs in latency, throughput and energy [20]. We could not find a study that compares all five families on one workload.

V. THE ATTACK LITERATURE

Two surveys organise the attacks on SGX in detail. One proposes explicit criteria for rating attacks and countermeasures [4]; the other tracks which attack applies to which SGX architecture and what information it leaks [5]. We use five classes (Fig 2, Table 2).

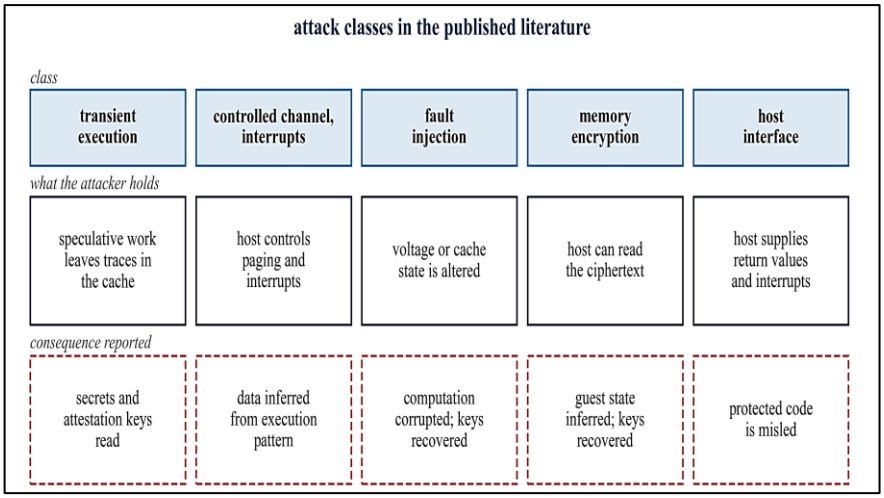


Fig. 2: Five attack classes, the control the attacker holds in each, and the consequence the cited studies report. Schematic drawn by the authors; sources are given in Table 2.

A. Transient execution

Processors guess the outcome of a branch and execute ahead. Spectre showed that a victim can be induced to perform, speculatively, operations that leak its confidential information through a side channel, on processors from Intel, AMD and Arm [21]. Foreshadow turned a speculative execution bug on SGX. It made no assumption about the victim enclave's code, leaked plaintext enclave secrets from the processor cache, and ended with the forged attestations mentioned above [17]. Intel has had to patch SGX repeatedly [5]. In our reading, an enclave's confidentiality rests on microarchitectural behaviour that the instruction set does not specify, a gap the Spectre authors also point to when they ask for instruction sets to state what computation state a processor is permitted to leak [21].

B. Controlled channels and interrupts

The TEE threat model hands the operating system to the attacker. Xu, Cui and Peinado named the resulting leaks controlled-channel attacks: an untrusted operating system extracted complete text documents and the outlines of JPEG images from applications protected by Haven and by a second shielding system [22]. SGX-Step later packaged the necessary control as an open-source framework that configures timer interrupts and tracks page-table entries, and can interrupt an enclave at the granularity of a single instruction [23]. These are not bugs in the usual sense. They follow from letting the adversary schedule and page its victim.

C. Fault injection

Fault attacks go after integrity. Plundervolt abused a privileged software interface for voltage scaling to corrupt computations inside SGX enclaves, recovering cryptographic keys and inducing memory-safety errors in code that had none [24]. CacheWarp did something comparable to SEV-ES and SEV-SNP guests by reverting modified cache lines to a stale state; its authors argue that mitigation is difficult because the root cause is in the hardware [25]. Physical access widens the field. A voltage glitch against AMD's security processor let attackers run their own firmware, decrypt a guest's memory and extract endorsement keys on all three microarchitectures that supported SEV at the time [26].

D. Memory encryption and the host interface

Memory encryption has a class to itself. A hypervisor that watches the ciphertext of an SEV guest change can infer the guest's execution state, and CIPHERLEAKS used this to steal private keys from constant-time RSA and ECDSA code [9]. A follow-up study found that the leak was not confined to one special page. Its authors trace the root cause to a stateless, unauthenticated encryption mode combined with unrestricted read access to ciphertext, and report that this remained unfixed [27].

The last class abuses whatever the host still supplies to the guest. Checkoway and Shacham showed in 2013 that a malicious kernel can lead a protected process to act against its interests through nothing more than a chosen sequence of system-call return values [28]. Heckler is the same lesson one layer down. A hypervisor injects interrupts whose handlers alter register state, and on both SEV-SNP and TDX this was enough to bypass authentication in OpenSSH and sudo [29]. Anything the hypervisor controls is an input the guest has to validate.

Table 2. Attack classes, with the consequence each cited study reports. No exploit detail is given.

Class	Studies	Consequence reported	Source
Transient execution	Spectre; Foreshadow	Speculative operations leak secrets; enclave secrets and attestation keys extracted from SGX	[17], [21]
Controlled channel and interrupts	Controlled-channel attacks; SGX-Step	Documents and image outlines recovered; enclave interrupted at single-instruction granularity	[22], [23]
Fault injection, software	Plundervolt; CacheWarp	Enclave or guest computation corrupted; keys recovered	[24], [25]
Fault injection, physical	Voltage glitch on AMD's security processor	Custom firmware run; guest memory decrypted; endorsement keys extracted	[26]
Memory encryption	CIPHERLEAKS and follow-up	Keys stolen from constant-time code; root cause reported unfixed	[9], [27]
Host interface	Iago attacks; runtime analysis; Heckler	Protected code misled by return values, unsanitised inputs or injected interrupts	[16], [28], [29]

VI. MACHINE LEARNING IN ENCLAVES

Mo, Tarkhani and Haddadi have systematised the work on machine learning in TEEs by whether it provides confidentiality or integrity [30]. An early SGX system trained support vector machines, neural networks, decision trees and other models with data-oblivious algorithms, chosen so that memory access patterns give nothing away, and reported overheads several orders of magnitude below those of multi-party cryptographic approaches [31]. Enclaves are slow at linear algebra, though. Slalom hands the linear layers of a network to an untrusted GPU and checks the results inside the enclave, which raised throughput by 6 to 20 times for verifiable inference over running everything in the enclave, and by 4 to 11 times when the inputs must also stay private [32].

Federated learning puts TEEs at both ends. PPFL trains inside the client's TEE and aggregates inside the server's. Because current TEEs have little memory it trains one layer at a time, and it reports client-side overheads of up to about 15% in processor time, 18% in memory and 21% in energy [33].

VII. CRYPTOGRAPHIC ALTERNATIVES AND OPEN PROBLEMS

Homomorphic encryption aims at the same goal under a different trust assumption: a third party operates on encrypted data without decrypting it, so nothing depends on a chip maker. The obstacle is cost. A survey of homomorphic schemes concluded that fully homomorphic encryption still needed to improve a great deal before it was practical on every platform [34]. The Slalom authors, comparing with a published cryptographic inference protocol, report 450 images per second for an eight-layer CIFAR-10 model in their SGX baseline, against a published 3.5 seconds for a single image under the protocol [32]. That is one model, and thin evidence. Still, the shape of the trade looks stable: a TEE is fast and asks for trust in hardware, firmware and a patching process, where cryptography is slow and asks for trust in a mathematical assumption.

Several problems are open. Side channels remain the developer's burden. For SEV, Li and colleagues report that AMD's patch for the first ciphertext attack was insufficient and that the root cause remained unfixed [27]. For SGX, van Schaik and colleagues describe Intel patching repeatedly while deployed products lagged behind the fixes [5]. Attestation tells a relying party which firmware a platform runs, not whether that firmware is sound. Confidential virtual machines bring a whole guest kernel inside the trust boundary, and the interrupt-injection result suggests that kernel was not written with a hostile hypervisor in mind [29]. Fig 3 groups the works cited here by theme.

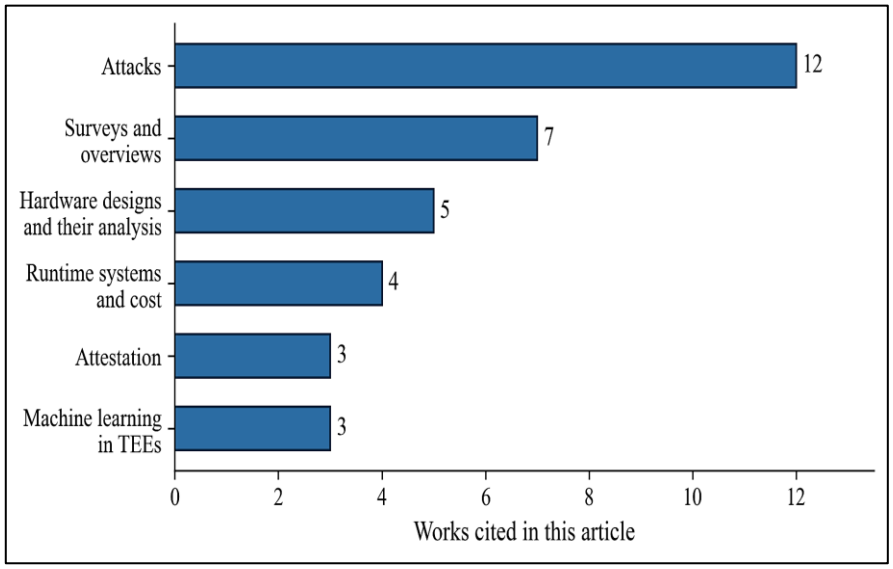


Fig. 3: The 34 works cited in this article, grouped by the role each plays in the review. The chart counts our own reference list and reports no experimental result.

VIII. CONCLUSION

The literature supports a modest reading of confidential computing. TEEs take the operating system and the hypervisor out of the set of things a data owner must trust, and existing software can run inside them at a measurable but often moderate cost. The published attacks give less comfort against an adversary who studies the processor. Each class reviewed here removed an assumption the early designs took for granted, whether about speculation, scheduling, supply voltage or the silence of ciphertext.

For a practitioner the consequences are concrete. A TEE is a strong reduction of the attack surface and should not be read as a proof of confidentiality, so it is worth deciding in advance which firmware and microcode versions an attestation policy accepts, and planning for the day one of them is withdrawn. Enclave code should be written as if the host can watch its memory accesses and interrupt it at will. Where data must stay secret even from the chip maker, cryptographic methods are the remaining option, and for now they are slow.

REFERENCES

[1] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with Haven," ACM Transactions on Computer Systems, vol. 33, no. 3, pp. 1–26, 2015, doi: 10.1145/2799647.

[2] M. Russinovich et al., "Toward confidential cloud computing," Communications of the ACM, vol. 64, no. 6, pp. 54–61, 2021, doi: 10.1145/3453930.

[3] D. P. Mulligan, G. Petri, N. Spinale, G. Stockwell, and H. J. M. Vincent, "Confidential computing—a brave new world," in Proc. Int. Symp. Secure Private Execution Environ. Design (SEED), 2021, pp. 132–138, doi: 10.1109/seed51797.2021.00025.

[4] S. Fei, Z. Yan, W. Ding, and H. Xie, "Security vulnerabilities of SGX and countermeasures: A survey," ACM Computing Surveys, vol. 54, no. 6, pp. 1–36, 2022, doi: 10.1145/3456631.

[5] S. van Schaik et al., "SoK: SGX.Fail: How stuff gets eXposed," in Proc. IEEE Symp. Security Privacy (SP), 2024, pp. 4143–4162, doi: 10.1109/sp54263.2024.00260.

[6] F. McKeen et al., "Innovative instructions and software model for isolated execution," in Proc. 2nd Int. Workshop Hardw. Archit. Support Secur. Privacy (HASP), 2013, Art. no. 10, doi: 10.1145/2487726.2488368.

[7] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: A practical library OS for unmodified applications on SGX," in Proc. USENIX Annu. Tech. Conf. (USENIX ATC), 2017, pp. 645–658.

[8] F. Hetzelt and R. Buhren, "Security analysis of encrypted virtual machines," in Proc. 13th ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ. (VEE), 2017, pp. 129–142, doi: 10.1145/3050748.3050763.

[9] M. Li, Y. Zhang, H. Wang, K. Li, and Y. Cheng, "CIPHERLEAKS: Breaking constant-time cryptography on AMD SEV via the ciphertext side channel," in Proc. 30th USENIX Security Symp., 2021, pp. 717–732.

[10] P.-C. Cheng et al., "Intel TDX demystified: A top-down approach," ACM Computing Surveys, vol. 56, no. 9, pp. 1–33, 2024, doi: 10.1145/3652597.

[11] M. Misono, D. Stavrakakis, N. Santos, and P. Bhatotia, "Confidential VMs explained: An empirical analysis of AMD SEV-SNP and Intel TDX," Proceedings of the ACM on Measurement and Analysis of Computing Systems, vol. 8, no. 3, pp. 1–42, 2024, doi: 10.1145/3700418.

[12] S. Pinto and N. Santos, "Demystifying Arm TrustZone: A comprehensive survey," ACM Computing Surveys, vol. 51, no. 6, pp. 1–36, 2019, doi: 10.1145/3291047.

[13] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in Proc. 15th Eur. Conf. Comput. Syst. (EuroSys), 2020, pp. 1–16, doi: 10.1145/3342195.3387532.

[14] J. Ménétrey et al., "Attestation mechanisms for trusted execution environments demystified," in Distributed Applications and Interoperable Systems (DAIS 2022), Lecture Notes in Computer Science. Cham, Switzerland: Springer, 2022, pp. 95–113, doi: 10.1007/978-3-031-16092-9_7.

[15] M. U. Sardar, S. Musaev, and C. Fetzer, "Demystifying attestation in Intel Trust Domain Extensions via formal verification," IEEE Access, vol. 9, pp. 83067–83079, 2021, doi: 10.1109/access.2021.3087421.

[16] J. Van Bulck, D. Oswald, E. Marin, A. Aldoseri, F. D. Garcia, and F. Piessens, "A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes," in Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS), 2019, pp. 1741–1758, doi: 10.1145/3319535.3363206.

[17] J. Van Bulck et al., "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in Proc. 27th USENIX Security Symp., 2018, pp. 991–1008.

[18] R. Buhren, C. Werling, and J.-P. Seifert, "Insecure until proven updated: Analyzing AMD SEV's remote attestation," in Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS), 2019, pp. 1087–1099, doi: 10.1145/3319535.3354216.

[19] S. Arnautov et al., "SCONE: Secure Linux containers with Intel SGX," in Proc. 12th USENIX Symp. Operating Syst. Design Implementation (OSDI), 2016, pp. 689–703.

[20] C. Göttel et al., "Security, performance and energy trade-offs of hardware-assisted memory protection mechanisms," in Proc. IEEE 37th Symp. Reliable Distrib. Syst. (SRDS), 2018, pp. 133–142, doi: 10.1109/srds.2018.00024.

[21] P. Kocher et al., "Spectre attacks: Exploiting speculative execution," in Proc. IEEE Symp. Security Privacy (SP), 2019, pp. 1–19, doi: 10.1109/sp.2019.00002.

[22] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in Proc. IEEE Symp. Security Privacy (SP), 2015, pp. 640–656, doi: 10.1109/sp.2015.45.

[23] J. Van Bulck, F. Piessens, and R. Strackx, "SGX-Step: A practical attack framework for precise enclave execution control," in Proc. 2nd Workshop Syst. Softw. Trusted Execution (SysTEX), 2017, pp. 1–6, doi: 10.1145/3152701.3152706.

[24] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against Intel SGX," in Proc. IEEE Symp. Security Privacy (SP), 2020, pp. 1466–1482, doi: 10.1109/sp40000.2020.00057.

[25] R. Zhang et al., "CacheWarp: Software-based fault injection using selective state reset," in Proc. 33rd USENIX Security Symp., 2024, pp. 1135–1151.

[26] R. Buhren, H.-N. Jacob, T. Krachenfels, and J.-P. Seifert, "One glitch to rule them all: Fault injection attacks against AMD's Secure Encrypted Virtualization," in Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS), 2021, pp. 2875–2889, doi: 10.1145/3460120.3484779.
[27] M. Li, L. Wilke, J. Wichelmann, T. Eisenbarth, R. Teodorescu, and Y. Zhang, "A systematic look at ciphertext side channels on AMD SEV-SNP," in Proc. IEEE Symp. Security Privacy (SP), 2022, pp. 337–351, doi: 10.1109/sp46214.2022.9833768.
[28] S. Checkoway and H. Shacham, "Iago attacks: Why the system call API is a bad untrusted RPC interface," ACM SIGPLAN Notices, vol. 48, no. 4, pp. 253–264, 2013, doi: 10.1145/2499368.2451145.
[29] B. Schlüter, S. Sridhara, M. Kuhne, A. Bertschi, and S. Shinde, "HECKLER: Breaking confidential VMs with malicious interrupts," in Proc. 33rd USENIX Security Symp., 2024, pp. 3459–3476.
[30] F. Mo, Z. Tarkhani, and H. Haddadi, "Machine learning with confidential computing: A systematization of knowledge," ACM Computing Surveys, vol. 56, no. 11, pp. 1–40, 2024, doi: 10.1145/3670007.
[31] O. Ohrimenko et al., "Oblivious multi-party machine learning on trusted processors," in Proc. 25th USENIX Security Symp., 2016, pp. 619–636.
[32] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in Proc. Int. Conf. Learn. Represent. (ICLR), 2019.
[33] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino, and N. Kourtellis, "PPFL: Privacy-preserving federated learning with trusted execution environments," in Proc. 19th Annu. Int. Conf. Mobile Syst., Appl., Services (MobiSys), 2021, pp. 94–108, doi: 10.1145/3458864.3466628.
[34] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," ACM Computing Surveys, vol. 51, no. 4, pp. 1–35, 2019, doi: 10.1145/3214303.