SecScale: A Scalable and Secure Trusted Execution Environment for Servers

Ani Sunny, Nivedita Shrivastava, Smruti R. Sarangi

[a] *CSE, IITD, Hauz Khas, New Delhi, 110016, Delhi, India*

Abstract

Trusted execution environments (TEEs) are an integral part of modern secure processors. They ensure that their application and code pages are confidential, tamper-proof, and immune to diverse types of attacks. In 2021, Intel suddenly announced its plans to deprecate its most trustworthy enclave, SGX, on its 11^{th} and 12^{th} generation processors. The reasons stemmed from the fact that it was difficult to scale the enclaves (sandboxes) beyond 256 MB – the hardware overheads outweighed the benefits. Competing solutions by Intel and other vendors are much more scalable, but do not provide many key security guarantees that SGX used to provide, notably replay attack protection. In the last three years, no proposal from industry or academia has been able to provide both scalability (with a modest slowdown) as well as replay-protection on generic hardware (to the best of our knowledge). We solve this problem by proposing *SecScale* that uses some new ideas centered around a *read-first, verify-later* approach, creating a forest of MACs (instead of a tree of counters), and providing complete memory encryption (no generic unsecure regions). We perform a rigorous security analysis in this paper and show that TOCTOU (time- of-check time-of-use) attacks are not possible. Furthermore, we demonstrate a 56% speedup over the nearest competitor that provides the same degree of fault coverage.

Keywords: Trusted Execution Environments(TEEs), Enclaves, Hardware Security

1. Introduction

The attacks on remotely executing software in both public and private clouds are on the rise[1]. Along with software-based attacks, a large number

Table 1: Commercially available TEEs. *Note:* ***TD*** *refers to a virtualization-based trust domain,* ***Realm*** *and* ***World*** *are equivalent to VMs;* ***Unlim Encl*** *refers to unlimited enclaves i.e. having no restriction on the number of enclaves,* ***Isolation Gran.*** *refers to the isolation granularity.*

System	**Arch.**	**Year**	**Mem Encryption**	**Integrity**	**Freshness**	**Unlim Encl**	**Scalable**	**Isolation Gran.**
ARM-TrustZone[4]	ARM	2004	×	✓	×	✓	✓	World
SGX-Client[5]	Intel	2015	✓	✓	✓	✓	×	Process
SGX-Server[6]	Intel	2016	✓	✓	×	✓	✓	Process
SEV[7]	AMD	2016	✓	×	×	×	✓	VM
SEV-ES[8]	AMD	2017	✓	×	×	×	✓	VM
SEV-SNP[9]	AMD	2020	✓	✓	×	×	✓	VM
ARM-CCA[10]	ARM	2021	×	✓	×	✓	✓	Realm
TDX[11]	Intel	2023	✓	✓	×	×	✓	TD
SecScale	Intel	2024	✓	✓	✓	✓	✓	Process

of physical attacks such as cold boot attacks and bus snooping are also being mounted [2]. According to an IBM report[3], the cost of a data breach in 2023 was $4.5 million and *82*% of the data breaches involved data that was stored in the *cloud*.

To secure data and computation in any such remote framework, we need to use a combination of encryption, message authentication codes (MACs), and digital signatures, respectively, for ensuring the following four ACIF properties: authenticity(A), confidentiality(C), integrity(I) and freshness(F). Authenticity refers to the fact that the data was indeed written by the server's CPU; confidentiality uses encryption to prevent snooping; integrity prevents tampering (using hashes and keyed hashes(MACs)) and freshness ensures that data that was valid in the past is not being *replayed*. Table 1 shows a list of all the major commercially available TEEs including Intel SGX.

Among all the commercially available TEEs listed in Table 1, now- deprecated Intel SGX[5] (Software Guard Extensions) is the only one that provides all four ACIF guarantees in HW (referred to as *SGX-Client*). Third-party software on SGX-Client used to run in a HW-managed *enclave securely* in spite of a potentially malicious OS or hypervisor. However, this *robust* protection came at a heavy price. The performance overheads limited the enclave size to 128-256 MB [12]. As a result, Intel decided to deprecate SGX-Client in its 11^{th} and 12^{th} generation processors and supplanted it with SGX-Server[1].

[1]Both SGX-Client and SGX-Server are terms that we introduce in this paper for the

SGX-Server adopts a different mode of memory encryption and eliminates time-hungry integrity (Merkle) trees altogether. It scales to 512 GB; however, this scalability comes at the cost of security – it is possible to mount replay attacks [12, 13].

This has sadly impacted different industries and products quite adversely. For example, ultra HD Blu-ray disks require the support of SGX's digital rights management (DRM) service[14]. They can no longer be played on new Intel processors that only support SGX-Server[15]. There are similar issues with DRM-protected PC games[16] and secure 4K video streaming apps [17]. SGX-Client allowed these apps to run in an enclave and consequently guarantee that the viewer wasn't able to steal video content [18]. References [19, 20] contain a lot of examples of replay attacks in distributed systems and software such as Ethereum and Bitcoin (attacks SGX-Client could prevent).

There are two strands of contemporary work. The first has been adopted by commercial silicon vendors who provide security solutions primarily for VMs (virtual machines) [9, 21, 10]. The assumption is that the entire guest VM is trustworthy, including its software stack [22]. In the second strand of recent work, proposed in academia (and the focus of our research), two proposals stand out. *Dynamic Fault History-Based Preloading (DFP)*[23] implements a prefetching-based mechanism to improve performance and support larger enclaves. Whereas, *Penglai*[24] is a bespoke RISC-V system that relies on caching parts of the integrity verification tree (Merkle tree) in a separate physical memory that has strict architecture-level access protections. It, however, violates our fault model, where we assume that the attacker can write to any location at will.

The insights in our work *SecScale* are as follows. We observe that the *read-first, verify later* approach that we adopt in the case of secure enclaves is very different from traditional approaches of forwarding unverified data in computer architecture. In this case, there are no rollbacks, and if some malicious data is forwarded, it is regarded as a catastrophic event that leads to system shutdown. This allows us to create a TOCTOU-attack-free scheme where data arrives at the CPU along with its encrypted key. The CPU immediately starts to use the data, albeit with safeguards. It waits till the corresponding MAC (encrypted hash) is verified. In *SecScale*, we use a low-

ease of explanation.

depth MAC forest as opposed to a traditional tree of counters. This is because counters make more sense in the case of block-wise encryption, whereas we mostly rely on page-wise encryption in *SecScale* for encrypting most of the memory space. The smaller tree height and the 8× lower storage overhead make our design much more scalable. Owing to these design choices, we need not have complicated page table management schemes as there are in prior work. Our page management scheme is quite scalable and straightforward, and it stops malicious accesses by privileged software as well.

The specific **contributions** in this work are as follows: ① Design of a scalable integrity protection mechanism, where we split the physical memory space into two parts: a 128 MB region with a traditional Merkle tree, and a 512 GB region with a MAC forest. ② A secure and low-overhead page table. ③ A novel execution scheme in which the data that arrives along with its key is immediately forwarded before verification (with safeguards) and which is immune to TOCTOU (time-of-check time-of-use) attacks. ④ A detailed performance and scalability analysis of *SecScale*. The design scales to 512 GB. It is 56% faster than the nearest competing work that assumes the same threat model.

§2 introduces the necessary background. §3 outlines the threat model, §4 characterizes the benchmarks and related work, §5 presents the proposed design, §6 shows a detailed performance analysis, §7 presents the related work, and we finally conclude in §8.

2. Background of Intel SGX

Intel Software Guard Extensions (SGX [5]) creates secure execution environments known as *enclaves*. The enclaves are located within a dedicated portion of a processor's memory – the *Enclave Page Cache (EPC)* whose size is between 128-256 MB. In SGX, only the on-chip components such as the processors, caches, NoC and memory encryption engine (MEE) are assumed to be secure. The external main memory is outside the protected domain and it is assumed that any adversary can read/write any location at will.

Enclaves are created before invoking the trusted code during execution. When we call a trusted function, secure execution starts within the enclave. Once the execution completes, the function returns and the context is switched back. Subsequently, normal unprotected execution of the application resumes.

The OS manages the page tables and the TLBs. However, any update to the TLB needs to be vetted by the SGX subsystem. Hence, a dedicated HW circuit verifies the integrity of the contents of the secure page and also ensures that no "secure" virtual address is mapped to an "unsecure" physical page or vice versa using an inverted page table. When the EPC is full, we need to evict a page, encrypt it and store it in the unsecure part of memory. Some metadata corresponding to it such as the key used to encrypt it and its MAC (keyed hash) are stored in the EPC. To reduce storage space, we can create an eviction tree of such evicted pages that is similar to the classical Merkle tree. Note that entering and exiting secure mode are expensive operations ($\approx$ 20-40k clock cycles [25, 5]). So is bringing back an evicted page to an EPC, hence, **EPC misses should be minimized.**

2.1. SGX-Client: (10^{th} Gen. Intel CPUs)

SGX-Client employs a memory encryption engine (MEE) [26], which is an extension of the memory controller. To maintain the *confidentiality* of the data, the MEE encrypts the data using *Advanced Encryption Standard (AES) counter-mode encryption[27, 28] (AES-CTR).* The counter values correspond to different data blocks of a page; whenever a data block is modified, its counter is incremented by one (to stop replay attacks). The inclusion of these counters in the encryption processs guarantees that effectively a new key is used for every encryption of the same block. The *integrity* of the counters is essential to the system as their correctness directly affects the system security. Their *integrity* is ensured in SGX via MACs stored in memory and a *Merkle tree* that aggregates the counters.

2.1.1. Integrity Verification using Merkle Trees

The leaf nodes of the Merkle tree store counters for the secure pages (part of the EPC), and the internal nodes of the tree store the counters for each of their child nodes. Additionally, each node stores the MAC of its counters (encrypted hash), in such a way that a *Carter-Wegman*[29] style tree is created – the MAC is generated by encrypting the hash of the counters in the node using the counters in its parent node. The root node thus captures the information of all the nodes in the tree. If there is a change in any counter, it will get reflected at the root. We thus need to store the root of the tree in the TCB (Trusted Computing Base) and for efficiency, we can store additional nodes of the tree in the TCB such that the root need

not be updated on every write. The Merkle tree is sadly not a very scalable structure and thus it became difficult to scale the design beyond 128 MB.

2.2. SGX-Server: (11^{th} *and* 12^{th} *Gen. Intel CPUs)*

Intel launched a new version of SGX (SGX-Server) along with its 3^{rd} Gen. Xeon Processors that scales to 512 GB. In SGX-Server, the physical memory is encrypted using the AES-XTS (Advanced Encryption Standard – Tweakable Block Ciphertext Stealing) [30] encryption engine. The basic idea is that multiple keys are used to encrypt data.

Sadly, the freshness security guarantee is sacrificed – this makes replay attacks possible[31]. This means that it is possible to replace the value in a memory location along with its MAC with a pair of values that were seen in the past. The processor will not be able to perceive that the memory contents have been tampered with. Moreover, it is possible to say that the value in a memory location is the same as that at a previous point of time – the ciphertext will be the same. This is an important side channel that leaks data. As a result, many users of SGX-Client haven't been able to migrate their systems to SGX-Server[12] – this has resulted in severe disruptions to their business.

3. Threat Model

SecScale considers a threat model similar to that of Intel SGX [5]. We include only the on-chip hardware components in the TCB, which includes the cores, caches, NoC, MEE and the hardware circuits that we introduce in *SecScale*. Other than these components, we only trust the code running within the enclaves with regards to their own execution. The SGX enclaves and standard cryptographic operations maintain confidentiality and detect integrity violations. The hardware components outside the TCB, the privileged software stack and other user applications including unrelated enclaves, are considered to be untrusted [32]. Figure 1 displays the trusted and untrusted components in the system.

Similar to [25, 33], *SecScale* assumes that the attacker controls the *system software stack* and can misuse its privileges to launch attacks such as observing and modifying the contents of memory addresses [34]. The attacker can also mount physical attacks such as snooping on the memory bus or cold boot attacks – observe and modify any memory location at will[35].

We identify the following properties (as seen in SGX-Client) that must be satisfied to ensure robust security:

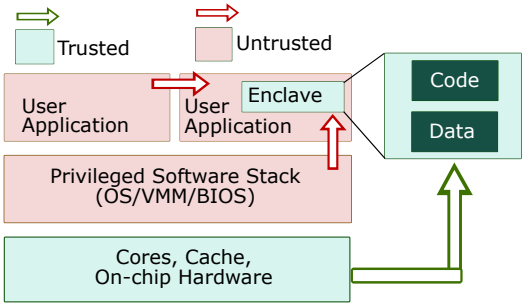


Figure 1: Threat model

1. **Property 1:** The sensitive data stored outside the EPC can be accessed and decrypted only by its owner.
2. **Property 2:** The integrity of sensitive data outside the EPC (in addition to data inside the EPC) is ensured.
3. **Property 3:** We never write unverified data to permanent storage or to I/O ports.
4. **Property 4:** The page table is not a point of vulnerability.

SecScale provides all four ACIF guarantees: authenticity, confidentiality, integrity and freshness. It protects the system against *replay attacks*[36], where the adversary may replace a data-block/MAC pair in memory. *SecScale* is immune to *TOCTOU (time- of-check time-of-use) attacks* [37, 38, 39] (refer §6). Akin to SGX and similar TEE schemes, *SecScale* does not consider side channel attacks (power, EM and cache), DoS attacks and attacks that introduce errors in the computation based on laser pulses or voltage spikes [40]. However, it is possible to incorporate the features of other solutions that mitigate side-channel attacks in SGX like Mirage[41], ScatterCache[42], Dr.SGX[43] and MoLE[44]. *SecScale* is compatible with all these solutions and can be used along with them to make it immune to side-channel attacks (refer §6).

4. Characterization

The aim of characterization is to determine the sources of performance degradation in SGX system by characterizing the behavior of benchmarks and baseline design.

4.1. Setup and Benchmarks

We ran the SPEC CPU 2017[45] benchmarks and characterized their performance on the different systems (standard practice while evaluating TEEs). These systems are modeled and simulated in a cycle-accurate simulator, Tejas[46]. Table 2 shows the simulation parameters. We used an algorithm similar to *PinPoints* [47] and *SimPoints* [48] to find the regions to simulate in each workload, and then we weighted them appropriately [49] to arrive at the final figures. We used Intel Pintools 3.21 [50].

Table 2: System Specifications

Processor			
Parameter	**Value**	**Parameter**	**Value**
Cores	1	Pipeline	4 Issue (Out of Order)
Caches			
Cache	**Size**	**Type**	**Associativity**
L1 I-cache	32 KB	Private	8
L1 D-cache	32 KB	Private	8
L2 cache	8 MB	Shared	8
Counter cache	32 KB	Shared	1
Memory			
Parameter	**Value**	**Parameter**	**Value**
Frequency	3.6 GHz	Channels	2
Ranks	2	Banks	8
Ports	1	Port Type	FCFS

Systems Modeled: We model two systems; *Baseline*, a vanilla design with no security, and *SGX*, the SGX-Client system that implements a Merkle Tree and a 128MB EPC. It guarantees all ACIF properties.

4.2. Observations

4.2.1. Performance Comparison

The performance (reciprocal of the simulated execution time) of SGX, normalized with respect to the baseline system performance, for different workloads is shown in Table 3. Compared to Baseline, SGX shows a very high performance degradation (mean: 83%) due to the overheads associated with traversing the Merkle tree and the EPC page fault penalties. The normalized performance varies from 9%, in *mcf*, to 93%, in *xalanc*.

Sources of Performance Overheads: Let us separately analyze the impact of Merkle tree traversal and EPC page fault servicing.

The performance of SGX, normalized w.r.t. the Baseline, with only the integrity tree (Merkle Tree) overheads, is shown in Table 4 (*Merkle-only*). We assume a zero EPC page

Table 3: Normalized Performance w.r.t. Baseline

Performance	deepsjeng	gcc	leela	mcf	parest	xalanc	xz	Avg.
SGX Overall	0.12	0.55	0.92	0.09	0.74	0.93	0.20	**0.17**

Table 4: Normalized performance of *SGX* w.r.t. *Baseline* – considering individual overhead sources. *(In **Merkle-only**, we assume EPC page fault overhead as '0', & in **EPC-only**, we assume Merkle tree overhead as '0'.)*

Performance	deepsjeng	gcc	leela	mcf	parest	xalanc	xz	Avg
Merkle-only	0.56	0.92	0.99	0.87	0.98	0.99	0.67	**0.83**
EPC-only	0.13	0.58	0.92	0.09	0.75	0.93	0.23	**0.18**

fault penalty. The performance degradation varies from benchmark to benchmark, with the average performance degradation being 17%. The degradation is the lowest in *leela and xalanc* benchmarks with a performance degradation of only 1%. A higher level of degradation is observed in the other benchmarks, with the highest degradation of 44% in *deepsjeng*. The memory bandwidth overhead incurred by the Merkle tree depends on the number of memory accesses and the memory access pattern. Benchmarks with more number of memory accesses show a higher degradation in performance.

As the Merkle tree height increases (with increasing memory size), the number of verifications (memory accesses) required to ensure integrity of the counters increases. Using a single Merkle tree to secure the counters for a large memory imposes significant memory access overheads, rendering it unsuitable for large systems (which is why Intel abolished the Merkle tree in SGX-Server). We must limit the integrity tree's height to control the overheads associated with memory access. Rather than a huge Merkle tree of counters, we need a more efficient structure to make the design scalable.

Insight 1: Encrypting large secure memory using counter-mode encryption with an integrity tree (Merkle Tree) fails to scale well, it rather imposes large performance overheads.

The performance of SGX, considering only the effect of overheads incurred due to EPC page fault handling is shown in Table 4 (*EPC-only*). We assume the overheads associated with the integrity tree to be zero. The performance degradation is much more drastic due to these overheads. The existing EPC page fault-handling mechanism is very costly and takes a large number of cycles ($\approx$ 40k cycles [25]) to complete. The entire page loading process is slow because of all the DRAM reads, decryption and the updation of metadata – this increases the length of the critical path. The lowest degradation is exhibited in *xalanc* (7%), which is closely followed by *leela* (8%). The performance degradation, imposed by EPC page fault handling alone, is as high as 91% in *mcf*.

4.2.2. Analyzing EPC Page Fault Overheads

To analyze the impact of EPC page fault penalties on the performance of the system, we simulate the system with varying values of the EPC page fault penalty and observe the difference in the performance (see Figure 2). The EPC page fault penalty has a direct

impact on system performance as it directly affects the latency of the critical path. The overhead increases from 44% (5k cycles) to 83% (40k cycles) relative to the baseline.

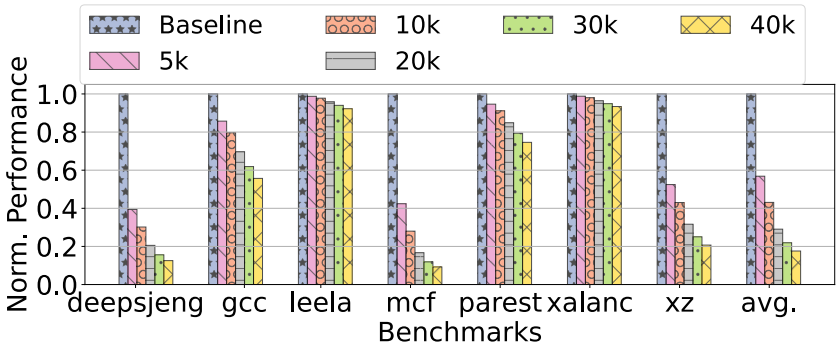


Figure 2: Performance comparison for varying page fault penalties in SGX. Baseline is the unsecure system, 5k, 10k, 20k, 30k and 40k refer to the variation in EPC page fault penalties, respectively (unit: clock cycles). *Conclusion: The page fault penalty directly increases the latency of the* ***critical path****.*

Since EPC page faults result in large performance overheads, we analyzed the frequency of such events by computing the number of evictions per 1000 instructions in different workloads (see Figure 3). We observe that the value is quite low in most cases: on an average *0.2 evictions per 1k instructions.* **Insight 2: Although infrequent, these EPC page faults have a huge impact on the system performance due to their excessively high latency.**

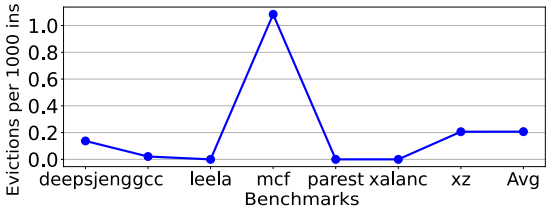


Figure 3: Evictions per 1k instructions for different workloads

4.2.3. Storage Overheads

In addition to the performance overheads that we saw earlier, the additional storage overhead can be visualized in Figure 4. The overhead varies linearly with the memory size. Over 8 GB of memory is required to store the counters for 512 GB memory. Extending the Merkle Tree to add a leaf node counter (for a page in secure memory) may require the addition of multiple nodes in the tree (parent nodes). **Insight 3: Unrestricted scalability, with freshness guarantees, can only be achieved if key management adds modest storage overheads.** There is a need to devise a more efficient mechanism for providing freshness guarantees that scale to TBs of physical memory.

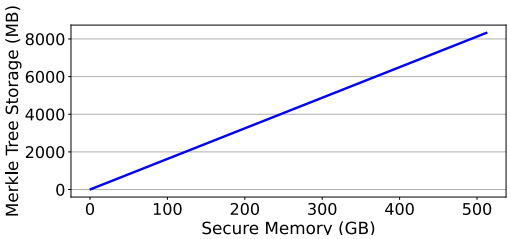


Figure 4: Storage overhead of using counters for guaranteeing freshness

Insight 1 Encrypting large memory using counter-mode encryption with a Merkle Tree does not scale well and imposes large performance overheads as the size of secure memory increases.
Insight 2 The huge latency of EPC page fault management is the reason why merely 0.02% of instructions (that cause EPC page faults) lead to about 82% performance reduction in SGX. Reducing the wait time associated with this latency is the key to reducing the latency of the critical path.
Insight 3 The management of keys to provide freshness guarantees should not lead to large storage overheads.

5. System Design

5.1. Overview

The design principle of *SecScale* revolves around the *read first, verify later* paradigm. Our system supports enclaves that are capable of handling large workloads up till 512 GB (similar to SGX-Server); the users get full ACIF security (similar to SGX-Client). In the basic design, an unlimited number of enclaves are supported (total size: 512 GB). This is achieved through efficient utilization of the pre-existing 128 MB EPC (part of SGX-Client) to create an eEPC (extended EPC) region of 512 GB (hereby, named eEPC). The design decisions are summarized in Table 5 (based on the characterization). The high-level design of *SecScale* is shown in Figure 5.

5.2. MAC Forest for Integrity Verification

Instead of a classical hash-tree of counters/keys, we propose a MAC forest mainly because a MAC is 64 bits in our system and a key is 256 bits (contains other information as well). In the eEPC region of *SecScale*, 64-bit MACs are computed at the page-level by a MAC Engine (ME) rather than at the block-level. These MACs are stored as the leaf nodes in the MAC forest that has different arities at each level (total: 3 levels for 512 GB). We group p MACs at the leaf nodes to generate a single 8-byte parent MAC. For higher levels, the arity is q. A part of the eEPC is reserved to store the internal nodes of

Table 5: Design decisions based on insights derived from characterization and high-level requirements (Section 4).

Insight 1: The Merkle tree restricts scalability.
Partition the physical memory into two regions: EPC and eEPC ('e' for extended). The former is a small region that uses block-level encryption and a Merkle tree for protecting counters (akin to Client-SGX). The latter can scale to TBs and uses page-level encryption.
Insight 2: The EPC page fault penalty needs to be reduced.
Let the data from the eEPC region come to the processor along with its key. Immediately start using the data (using safeguards) and start the verification in parallel. Before writing to durable storage, verify all outstanding reads.
Insight 3: The freshness guarantee in a scalable system requires managing billions of keys.
In the eEPC, encrypt data at the page level, and store an encrypted key for each page. Counters are not very relevant here because such pages are infrequently accessed and the storage overheads are prohibitive.
Requirement 1: Make it easy to manage the page table and the interaction with a possibly malicious OS.
If different enclaves (secure programs including the OS) have separate encryption keys, then this can be easily achieved. However, this will increase the size of a key because it needs to contain the enclave/process id as well. Hence, to save space, verify the integrity by creating a *MAC forest* as opposed to a single integrity tree comprising keys.

the subtrees in the MAC forest. The topmost level of these subtrees is securely stored in the EPC (part of the TCB). The MAC forest structure is shown in Figure 6.

We retain the use of counters and the Merkle tree for protecting the EPC region (akin to SGX-Client) with the same permission scheme as SGX-Client for accesses. In our evaluated design, we consider a MAC forest consisting of subtrees with $q = 3$ (3-level tree) and $p = 16 \times 8$ (arity 16 at the lower level and 8 at the higher level). In this case, the MACs of 16 pages will be grouped in the lower level to generate a MAC at the parent. For the next level, we group 8 MACs to generate a parent MAC which is part of the top most level of the forest. For 512 GB memory, we get a MAC forest with the topmost level containing 2^{20} MACs (securely stored in the EPC, 8 MB total storage space). By limiting the levels of the subtrees we can contain the additional memory bandwidth required for integrity verification (maximum of 4 additional accesses in our representative design).

MAC Verification Circuit (MVC): We implement a MAC verification circuit (MVC) that is responsible for verifying the integrity of the pages within the eEPC region. The

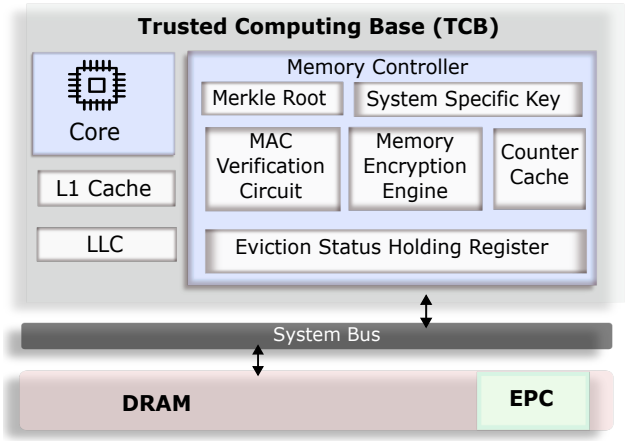


Figure 5: The high-level design of *SecScale*

MVC performs this verification when a page is loaded into the EPC. Note that we perform *deferred MAC verification* – we do not halt the normal execution to wait for the outcome of the verification process. It uses the SHA-2 algorithm to compute MACs (Throughput: 40 Gbps at 5.15 GHz frequency at the 7 nm tech. node).

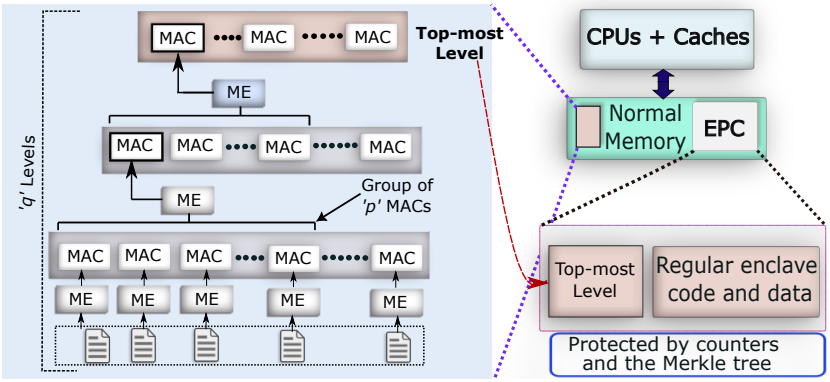


Figure 6: MAC forest with numerous subtrees. (ME refers to the MAC engine)

We observe in section 4 that the memory access overheads associated with the Merkle tree restrict the system's scalability. Table 6 shows how the increase in memory size affects the characteristics of the Merkle tree and the MAC forest. The Merkle tree height increases as the memory size increases, whereas the MAC subtree height is smaller (as shown in our evaluated design). As the tree height increases, additional memory accesses are required to verify the integrity of the counters in the system, thereby increasing the memory access latency. As shown in our evaluated design, we limit the height of the subtrees in the MAC forest, ensuring fast MAC verification. Moreover, *SecScale* takes MAC verification off the

critical path to further improve system performance. The integrity verification latency (on the critical path) is higher for systems with a large Merkle tree compared to *SecScale*, which has a small Merkle tree limited to the 128 MB EPC region. Thus, by leveraging a MAC forest of small subtrees to secure page-level MACs for the large eEPC region, we restrict the size of the integrity trees and control the associated memory access overheads.

Table 6: Merkle tree *(in previous designs)* vs MAC forest *(in* SecScale*)* for systems with large memory size.

Feature	**Impact of Large Memory Size**	
	Merkle Tree	**MAC forest**
Tree height	High *(6 levels for 512 GB)*	Limited by the design *(3 levels for 512 GB)*
Memory accesses *(for integrity verification)*	Higher *(up to 6 for 512 GB)*	Limited by the design *(up to 4 for 512 GB)*
Critical path latency	Higher due to increased tree height.	**eEPC:** MAC verification off the critical path. **EPC:** Latency controlled by using a small Merkle tree *(4 levels for 128 MB EPC region irrespective of the system memory size).*
Storage overhead *(Refer section 6)*	Higher *(8322.06 MB) for 512 GB*	Comparatively low *(1098.06 MB for 512 GB i.e 8× lower)*

The Merkle Tree and counters are limited to the EPC (like SGX-Client) and the MAC forest is used for the eEPC region (with the MVC). This enables us to limit the tree height and control mamory access overheads.(Property 2)

5.3. Full Memory Encryption

To protect the pages in the eEPC region, we use vanilla 256-bit AES-ECB encryption with a few tweaks. Every page in the eEPC region is encrypted using a different encryption key that is randomly generated every time a modification is made. This mechanism effectively safeguards against *replay attacks* because keys are not reused (probabilistically).

We use a 256-bit key to encrypt every block in a page: hardware-specific (HW) key (64 bits), enclave ID (31 bits), bits generated by a pseudo random number generator (128 bits) seeded by the boot time and HW key, physical address of the page in 512 GB memory (27 bits) and block address within the page (6 bits, used while encryption/decryption only). We keep the PRNG component large because a new value needs to be created for every

encryption. The components of the key excluding the block-specific 6 bits constitute the page specific key, K. The block-specific address bits are extracted for every block of the page and concatenated with K to generate the block-specific key k_b, where b represents the b^{th} page block. This ensures that the keys used for encrypting each block of the page are different. Thus, if the same data is stored in different memory blocks of the page, different ciphertexts will be generated.

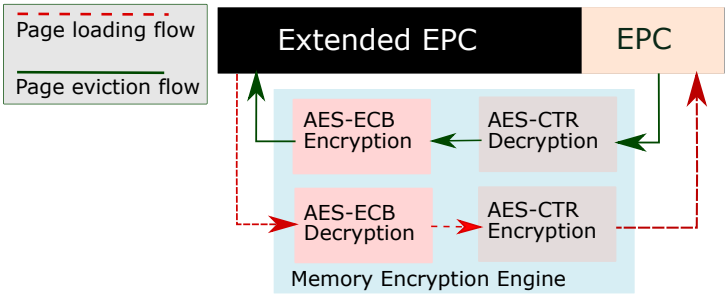


Figure 7: Encryption-decryption during EPC page eviction and loading

5.3.1. EPC – eEPC Page Transfer

The encryption-decryption process when a page is getting transferred from the EPC to the eEPC is shown in Figure 7. It is triggered by an EPC page fault. When a page is evicted from the EPC, the encrypted page is first decrypted using the AES-CTR mode (SGX-Client), and then it is re-encrypted using the AES-ECB-256 mode with a secret key generated by the MEE.

In the eEPC region, SecScale *generates a new key every time the page is written to. This guarantees freshness.*

The page specific key, K, is then encrypted using a system-specific key *SSK*. The *SSK* is a concatenation of a second device-specific key (128 bits) and a 128-bit number generated at boot time. The *SSK* is stored in a dedicated register in the TCB. The block-specific values in the key are extracted from the address at the time of cryptographic operations, and only the randomly generated page specific key, K, is stored in the eEPC region. When encrypting and sending the key to the eEPC, the block address within the page is set to zero. The encrypted page and its encrypted key are then moved to the eviction region – both are stored in the eEPC region.

A section of the physical memory called the Key Table stores all the encrypted keys for each of the physical pages. Given that we have 2^{39-12} (=128M) physical pages in our system and each key is 32 bytes, we need 4GB of storage for storing the keys. This translates to 0.8% overhead for storing the keys in physical memory. The advantage of storing the keys in this manner is that we can easily locate the key for a page and fetch it along with the evicted page when it is required in the EPC. Basically, an evicted page

comes to the EPC along with its encrypted key. We trust *the key for the time being* till verification. Thus, key storage and management overheads are effectively reduced while maintaining key *freshness*. While bringing a page from the eEPC to the EPC region, a reverse process is followed (as shown in Figure 7). First the page is decrypted using the AES-ECB mode using the encrypted key read from the Key Table and then its blocks are re-encrypted using the AES-CTR mode (within

The integrity of the keys in the Key Table are maintained as follows: Use the page-key to encrypt the hash of the page and generate the MAC. The key cannot be unilaterally changed (the MAC check will fail), the key and the MAC cannot be replayed (the MAC check at the higher level will fail) and another enclave cannot read or write the data (the MAC check will fail because of the incorrect enclave ID). Higher-level MACs are created by encrypting the hash with the SSK.

5.4. Optimization of the EPC Page Fault Handling Mechanism

Figure 8 shows the entire process of fetching a page along with its key, decrypting and verifying it.

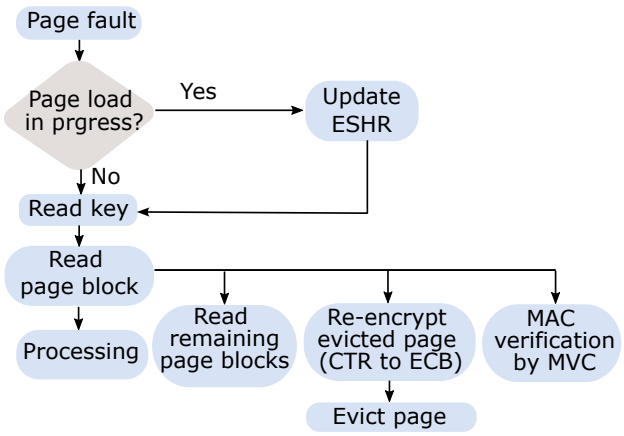


Figure 8: Handling EPC page faults

If there is no other concurrent EPC miss, this process continues without interruption. Assume another EPC page fault occurs before this page is fully loaded (i.e., before all the blocks of the mapped page have been loaded into the EPC). The current process of loading the EPC page must be run *in parallel* while also processing the new request. We need to first store the status of the ongoing eviction/loading process so that loading can be resumed later from the current state. We introduce an additional hardware structure called the ESHR Table to store this information regarding the page loading and eviction status.

Each entry in the table is an Eviction Status Holding Register (ESHR). Subsequently, the block of the new page along with its key are fetched into the EPC. Once it is fully loaded we resume the process of loading the rest of the blocks of the page whose status was saved in an ESHR.

ESHR Table To keep track of which page blocks are loaded in the EPC, we maintain a 64-bit load status vector (LS vector) in the ESHR. We have 64 bits because there are 64 blocks in a page. When a block is loaded into the EPC, its corresponding bit in the LS vector is set to 1. Once all the blocks are loaded, the valid bit (V) is reset. The required page ($LPage$) is loaded in place of the corresponding evicted page ($EPage$) as indicated in the ESHR. The eviction bit (E) is set to 1 if there is an eviction along with loading.

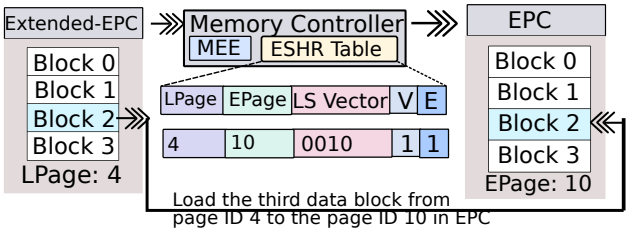


Figure 9: Handling EPC page faults using the ESHR table

The ESHR table stores 32 entries. Each entry in the ESHR table contains five fields: *EPage* represents the page ID of the evicted page; *LPage* represents the page ID of the page that is being loaded (newly mapped); *LS Vector* is a 64-bit vector that indicates the loading status of *LPage*; *E-bit* is an eviction bit indicating we need an eviction (of the *EPage*); and *V-bit* is a valid bit indicating if the page is fully loaded or not. We present a dummy example in Figure 9, where each page comprises four blocks. The page blocks in the *LPage*, which are located in the eEPC are loaded to the EPC at the position pointed to by *EPage*.

Execution Flow:

5.4.1. Read Path

When an EPC miss occurs for a read request, the data is fetched immediately from the eEPC region as it lies on the critical path (Figure 8 shows the steps).

5.4.2. Write Path

A write request does not lie on the critical path. In case of an EPC miss for a write request, we do the following: ① If there are no other requests queued in the memory bus, the write request proceeds as usual. ② If a previous operation is still ongoing, an entry for the request is made in the ESHR; the request is made to wait til the process completes. ③ If a read request arrives while this write is waiting, the read request is given higher priority . Additionally, if the two requests are for a memory region covered by the same subtree, they are grouped together such that for the higher level MACs of the subtree, a single verification operation would suffice for both the requests.

5.4.3. Verification Path:

The MAC verification of the newly mapped page is carried out concurrently by the MVC, while the page blocks are being moved to the EPC. The execution flow is not hindered by this process. Additionally, the remaining page blocks are loaded/evicted in parallel, while verification and MAC computation are underway. Thus, while fetching data from the eEPC, the execution can be restarted after just two memory reads (requested page block and its key). This drastically reduces the latency of the critical path. As the overhead incurred during EPC page faults is comparable to the overhead of fetching data from the EPC itself, the overall impact of the large eEPC is very small.

5.4.4. Communication with the OS

We maintain a separate memory region with very few pages that has relaxed security guarantees. This memory region is used for communication between trusted enclaves or between an enclave and the untrusted OS. The enclaves use this memory as a scratchpad for sending system call arguments and receiving data from the OS and other enclaves. Similar to SGX-Client, this memory region can either have no security or it could be encrypted with a session key.

5.5. Design Optimizations

MAC verification does not lie on the critical path, but it still accounts for DRAM accesses. These accesses can delay regular accesses. They also increase DRAM power consumption. Hence, there is a need to minimize such additional accesses.

5.5.1. Optimizing MAC Verification

We maintain a small cache in the TCB that stores r recently accessed top-level MACs of the MAC forest (i.e., the roots of the recently accessed subtrees in the MAC forest). Each MAC at the top level of the forest is the root MAC for a 512 KB memory region in the eEPC region. This top-level MAC, which is stored in the EPC, needs to be retrieved from the main memory while performing MAC verification of any of the pages belonging to the region covered by this subtree root (defined as its *subtree region*). Hence, we reduce one DRAM memory access by caching it.

5.5.2. Optimizing MAC Forest Updates

Updates to the MAC forest are required when pages are evicted from the EPC. The page to be evicted is selected based on an LRU (least recently used) mechanism (state stored in the EPC's metadata, EPCM). We additionally store the ID of the page that is next in line for eviction in an *evict register* (computation is off the critical path). If both the currently evicted page and the next page to be evicted lie in the memory region protected by the same subtree of the MAC forest, then the MAC updates to higher levels of the subtree, for both the pages can be clubbed together – this reduces the number of memory accesses at the higher levels of the MAC forest.

5.5.3. Corner Cases

① While the CPU execution using unverified data is in progress, if an I/O call occurs or there is a write to any form of durable storage, the processor first waits for the MAC verification to complete before taking any action.

② Without waiting to pair a write with an eviction (for reducing DRAM writes), we finish all verification operations as soon as possible.

The page table does not require significant modifications (discussed in Section 6.6.2).

5.6. SecScale Distinguishing Features

SecScale involves a fusion of multiple ideas that collectively represent a **novel solution**. We list the distinguishing features of our design that together constitute a unique approach for providing secure and scalable TEEs.

- The *read first, verify later* approach takes integrity verification off the critical path for eEPC region, significantly reducing the EPC eviction overheads.
- A fresh and unique key is generated for every encryption of every page in the secure memory. The key arrives with the data when the page is accessed.
- A MAC forest of small height for integrity verification of the enhanced enclaves (eEPC region) with a small Merkle tree for the 128 MB EPC region, limit memory access overheads and critical path latency.

To the best of our knowledge, there is no work where the data arrives along with its key, a MAC forest is used instead of a Merkle tree of counters, *read first, verify later* approach is used in the manner described, and a low-overhead secure page table is created for use in a TEE. *SecScale* offers a scalable and secure solution that provides all four ACIF guarantees and efficiently secures systems with large memory.

6. Evaluation

We evaluate *SecScale*'s performance against that of SGX, as well as the other systems in the domain, namely VAULT[25], Morphable Ctrs (Morphable Counters) [33], DFP[23], and Penglai[24]. We simulate the systems in our cycle-accurate simulator, Tejas[46].

Table 7: Security Constructs of the Models Simulated

System	**Integrity Tree**	**EPC**
Baseline	No	No
SGX-Client	Merkle Tree	128MB EPC
VAULT	Merkle Tree	No
Morphable Ctrs	High-Arity Merkle Tree	128MB EPC
DFP	Merkle Tree	128MB EPC
Penglai	Mountable Merkle Tree	No

Table 7 shows the security constructs of the systems. *VAULT* implements a large Merkle Tree for the entire memory (512 GB). The whole memory can serve as the EPC.

Morphable Counters implements a high-arity morphable Merkle Tree (128 arity) and 128 MB EPC. The high arity of the tree reduces its height (compared to VAULT). *DFP* has a Merkle tree and a 128MB EPC, which is the same as SGX-Client. It additionally implements a predictor that predicts page faults in the near future and prefetches the pages. *Penglai* has a Mountable Merkle Tree (MMT), with a root tree and multiple subtrees, that can support 512 GB of secure memory but does not have an EPC. It only caches the 32 most recently seen subtree roots. For every LLC miss, multiple additional memory accesses are required to retrieve the tree nodes for integrity verification.

6.1. Performance Analysis

Performance is proportional to the reciprocal of the simulated execution time. In most workloads, the memory accesses are not very irregular. Hence, the frequency of EPC page faults is low in general. The exceptions arise in the cases where the memory accessed is very large and the pattern of page accesses is random. These benchmarks experience an increased number of EPC page faults and incur a higher memory traffic overhead for integrity verification. Consequently, the benchmarks with a larger number of page faults will show a greater degradation in performance, as we can see in Figure 10.

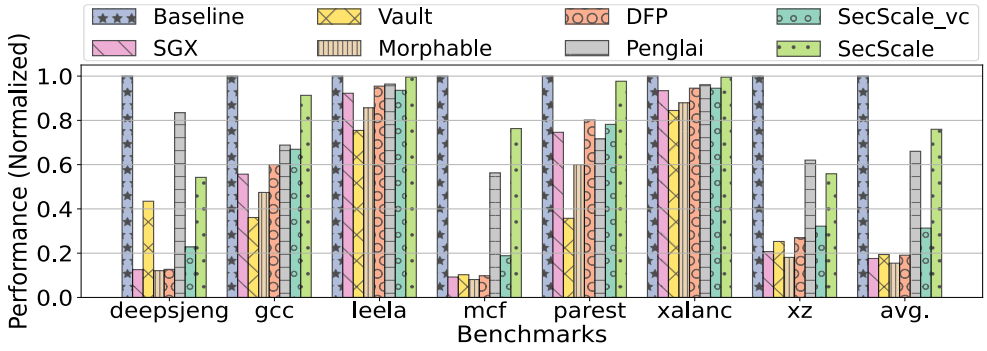


Figure 10: Performance of different systems. (*SecScale* exhibits performance improvement over all the systems.)

SGX exhibits a drastic degradation in performance compared to the baseline (83%). Comparing the performance of the related work with that of SGX, we see that VAULT and DFP show a 2% improvement in performance, and Penglai performs 49% better than SGX. Morphable Counters, however, performs 2% worse than SGX. In 5/7 of the workloads, *SecScale* performs much better than all the others and exhibits the lowest degradation in average performance(24%) with respect to the baseline. On an average, *SecScale exhibits a 58% improvement in performance* vis-a-vis SGX-Client. Additionally, it exhibits a performance improvement of 56%, 60%, 56%, and 10%, over VAULT, Morphable Counters, DFP, and Penglai (respectively). A different version of *SecScale*, called *SecScale_vc*, where data is forwarded to the processor only after integrity verification is complete, was implemented to analyze the system performance without employing the *read first, verify later* approach. *SecScale_vc* performs better than SGX and Morphable Counters for all

the workloads. Its performance exceeds that of VAULT for *gcc, leela, mcf, parest, xalanc and xz* by 30%, 18%, 8%, 43%, 10% and 7% respectively, and that of DFP for *deepsjeng, gcc and xz* by 10%, 6% and 5% respectively. It performs better than Penglai for *parest* benchmark. Its average performance degradation is 69%.

We observe that the average performance of DFP, VAULT, and Morphable Counters is even worse than the version of SecScale that does not rely on the *read first, verify later* strategy. Although VAULT does not have a limited EPC region (no EPC eviction), it uses a single Merkle tree to protect the entire memory space. Morphable Counters utilizes a high-arity tree to reduce the tree height. Even though these systems have explored various enhancements on the Merkle tree, they do not successfully optimize memory access overheads in large systems. These systems are specifically not designed to support a large EPC size and thus underperform on many fronts. We cannot rely on one large Merkle tree, like these designs do. *SecScale* overcomes the limitations of previous implementations and enables large enclaves. To minimize memory access overheads, *SecScale* employs a MAC forest (with several 3-level MAC subtrees) to protect the vast eEPC memory and a small Merkle tree to secure the EPC region. In contrast to the previous solutions that leverage a large Merkle tree, *SecScale* limits the height of the Merkle tree by restricting it to the EPC region, and the small height of the MAC subtrees ensures fast completion of the MAC verification (for eEPC pages). Thus, leveraging a MAC forest with small subtrees for the eEPC region and a small Merkle tree for the EPC region proves to be an appropriate design choice for a scalable solution. Furthermore, comparing the performance of the two versions of *SecScale*, we can infer that the *read first, verify later* approach – leading to a reduced wait time associated with EPC page fault management – is the key contributor to the reduced execution time in *SecScale*.

6.2. Detailed Analysis

6.2.1. Impact of EPC Miss Rate on Performance

The EPC page fault rate, in terms of EPC misses (%) with respect to LLC misses is shown in Figure 11. This map gives us an estimate of the spatial locality in the benchmark suite. The benchmarks that access more memory pages are associated with a higher EPC page fault percentage. The access pattern (randomness) of the pages also affects the EPC page faults. The benchmarks with a high percentage of EPC misses with respect to LLC misses like *deepsjeng* experience drastic degradation in performance (88% degradation w.r.t. baseline in SGX) as shown in Figure 10. On the other hand *leela*, which has a very low EPC miss rate w.r.t. to LLC misses, experiences a very low degradation (8% degradation w.r.t. baseline in SGX).

6.2.2. Impact of Eviction Rate on Performance

Figure 12 shows the evictions per EPC miss for different models. The page access pattern affects the eviction rate in the system.

The eviction rate is the same in both SGX and Morphable Counters, as their EPC implementation is the same. Note that the number of evictions could be less than 1 if we have already created space for the page using prefetching (like in DFP). Specifically, the eviction rate varies (increases/decreases) in DFP, compared to SGX, because of its

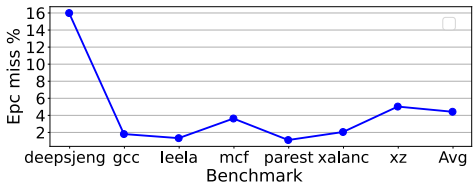


Figure 11: EPC miss % w.r.t. LLC misses for different benchmarks. (The benchmarks with *higher* EPC miss rates *(page faults)* are the ones with a *greater performance degradation.*)

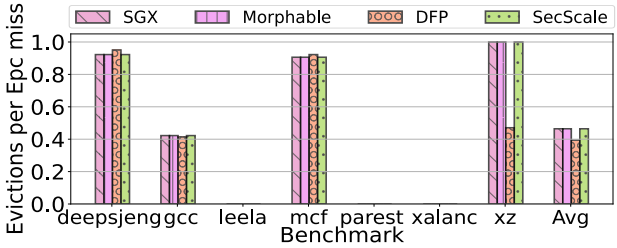


Figure 12: Evictions per EPC miss. *(Eviction rate directly affects the extent of performance degradation.)*

predictions/mis-predictions. In *deepsjeng and mcf*, the evictions increase for DFP because of mispredictions, whereas in *xz*, asynchronous preloading of correctly predicted faulting pages reduces the evictions in the critical path. DFP, thus, shows better performance than SGX in the case of *xz* even though both SGX and DFP impose the same penalty for evictions. However, in *SecScale*, the system performs much better than SGX, Morphable Counters and DFP in all the benchmarks (even though it has the same eviction rate as SGX), including *deepsjeng, mcf and xz* with and improvement of 42%, 65%, and 35% respectively, over SGX, because it drastically reduces the penalty associated with EPC misses and evictions.

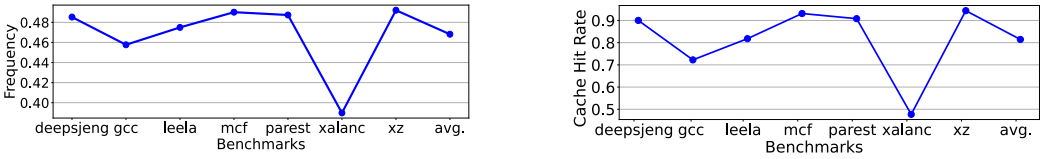
The page access pattern affects the integrity tree access overheads in every system. In Penglai, it impacts the mountable Merkle tree (MMT) access overheads. The additional bandwidth associated with integrity verification varies depending on this pattern, as can be seen in the case of *parest* where it performs worse than SGX by 3% and DFP by 9% even though it does not have an EPC (and EPC associated overheads). In VAULT, the memory bandwidth overhead is very high due to the higher number of levels in the single large integrity tree (for 512 GB memory) that it maintains. As such, its performance is worse than SGX in 4/7 of the workloads (*gcc, leela, parest and xalanc* by 19%, 17%, 39%, 9% resp.). Morphable Counters performs worse than SGX in all the workloads. In the case of *SecScale*, the Merkle tree protects the counters of only those pages that reside inside the EPC, and therefore it has a fixed size. *This ensures that the Merkle tree overhead is minimized in SecScale.* The result of these optimizations is evident in the performance improvement seen in *parest* for *SecScale* (23% over SGX, 62% over VAULT, 37% over Morphable Counters, 17% over DFP and 26% over Penglai).

6.3. Proposed Optimizations for Reducing MAC Accesses

The MACs are accessed during the MAC verification and MAC update phases. These processes require fetching MACs from different levels of the MAC forest (stored in memory). Although these accesses are not on the critical path, every memory access increases DRAM traffic and DRAM power. We introduced optimizations in the design to reduce the number of additional memory accesses required to perform these operations.

6.3.1. Optimizing MAC Updates

We club the MAC updates for the higher levels of the subtrees in the MAC forest during consecutive evictions of pages that belong to the same *subtree region*. This reduces the number of memory accesses required for updating the MACs in the higher levels of the MAC forest. Figure 13a shows the frequency of clubbing of the updates observed in our workloads (an average of 46% clubbing was observed). *This plot depicts the spatial locality in the benchmarks within a subtree region (512 KB memory region).* Note that clubbing of updates is possible only if consecutive pages fall in this region.



(a) (b)

Figure 13: (a) Frequency of clubbing *(clubbing of updates is possible for the pages protected by the same subtree.)* (b) Top-level MAC cache hit rate.

6.3.2. Optimizing MAC Verification

We introduced a small cache in the TCB to store the 8 recently accessed MACs from the top-most level of the MAC forest in the TCB. We attempt to leverage any locality of accesses that might exist in the workloads for the pages secured by the subtrees of the cached top-level nodes. This reduces the number of memory accesses required to retrieve the top-level nodes from the EPC for MAC verification. Figure 13b shows the cache hit rates in our design for various workloads. The average hit rate is 81.4%.

These optimizations leverage the temporal and spatial locality of EPC misses in order to reduce the number of memory accesses to the higher level nodes in the MAC forest. If we observe closely, both these figures show a similar pattern – they depict the extent of locality in each workload. Comparing the pattern with DRAM traffic reduction (shown in Figure 14), we see that the benchmark *mcf* that has a high MAC cache hit rate and frequency of clubbing exhibits a larger decrease in DRAM traffic (24%), compared to xalanx (16%) which has a low cache hit rate and clubbing frequency. We observe that by employing both these optimizations, the overall additional memory accesses to retrieve the MACs reduces by 23.5% in our workloads.

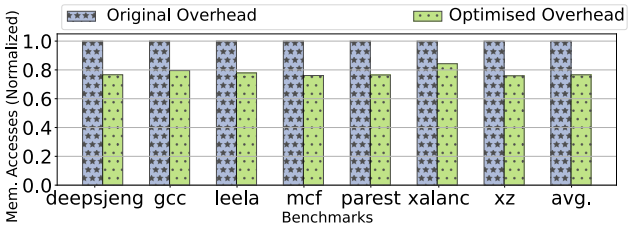


Figure 14: Reduction in DRAM traffic due to optimizations for MAC accesses. *(Result: additional accesses reduced by 23.5% in our workloads.)*

6.4. Extended Performance Analysis

We conducted additional simulations using custom workloads, representing VM-like behavior, to extend our performance evaluation. A typical virtual machine (VM) can be abstracted as a bag of processes running a full operating system and supporting multi-tasking workloads, similar to a physical machine. The internal workings of these processes—including memory mappings—remain invisible to the host system.

We aggregate different processes (SPEC 2017 Benchmarks) to form a *bag of processes*. Each workload (WL) is composed of a distinct combination of processes. These workloads are then run on the various systems to compare their average performance. Figure 15 shows the performance across all the systems.

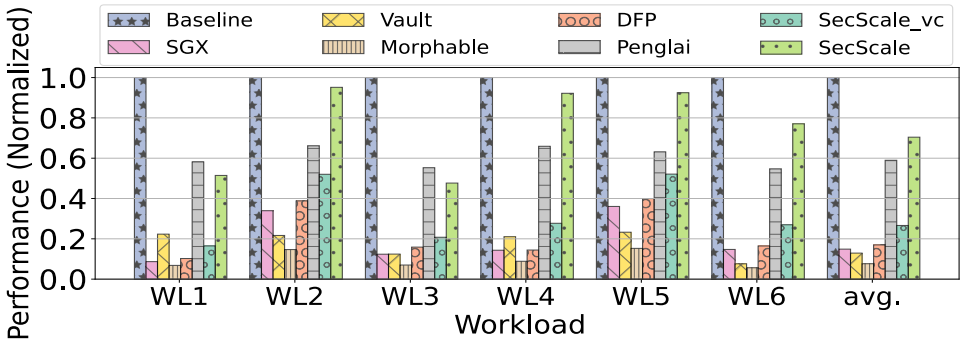


Figure 15: Performance of different systems on Cloud-like Workloads. (*SecScale* exhibits performance improvement over all the systems.)

The performance of different models varies for different workloads and depends on the constituting processes in each WL. We compare the average performance of running the different WLs on the various systems. SGX exhibits a performance degradation os 86% compared to the baseline. The performance degradation for VAULT, Morphable Counters, and DFP is observed to be 87%, 93%, and 83%, respectively. We observe a lower performance degradation for Penglai (42%). In 4/6 VMs, *SecScale* performs better than all the others and exhibits the lowest average performance degradation (29%). *SecScale* performs 57% better than SGX-Client. Additionally, it exhibits a performance improvement of 45%, 51%, 41%, and 12% over Vault, Morphable Counters, DFP, and Penglai

(respectively). *SecScale_vc* (the *SecScale* version that doesn't implement 'read first, verify later' approach) exhibits a better average performance (72% degradation) than SGX, DFP, Vault, and Morphable Counters.

Table 8: Comparison of *SecScale* with Confidential Virtual Machines (CVMs). *(MKTME refers to Multi-Key Total Memory Encryption for encrypting private memory in each trust domain (VM), SW refers to Software & HW refers to Hardware.)*

Feature	**CVM *(TDX)***	***SecScale***
Isolation Granularity	VM	VM & Process
TCB	HW & SW *(On-chip HW components & SW Stack of Guest VMs)*	HW *(On-chip HW compponents)*
ACIF Guarantees	ACI ***(No Freshness)***	ACIF *(Key freshness: Merkle tree for EPC & RNG for eEPC)*
System Slowdown (Compared to unsecure baseline)	Lower *(*1.274*)*	Higher *(*1.418*)*
Enclave Count	Limited by # MKTME keys (1 key per VM) *(62-63 in 4th and 5th Gen Intel Xeon Scalable processors)*	Unlimited *(No limit on #keys)*
Storage Overhead (MAC Storage)	Higher *(28 GB for 512 GB secure memory)*	Lower *(1.07 GB for 512 GB i.e* ***26× lower than TDX****)*

6.4.1. SecScale vs Confidential Virtual Machines(CVMs)

To perform a comprehensive analysis, we compare *SecScale* with CVMs (Confidential Virtual Machines) in Table 8. We analyze the trade-offs between CVMs (specifically Intel TDX (Trust Domain Extensions)) and *SecScale* in terms of security, performance, and resource usage. We modeled *Intel TDX* and simulated new workloads (representing VM-like behavior) on both TDX and *SecScale* to evaluate performance slowdown. We ran the different workloads on both systems and evaluated system performance relative to an unprotected (baseline) system. The observed slowdowns with respect to the baseline were 1.274 for TDX and 1.418 for *SecScale*. While *SecScale* incurs a slightly higher slowdown, it offers stronger security guarantees (satisfying all ACIF properties) and significantly lower storage overhead (26× lower compared to TDX). Moreover, *SecScale* is a scalable design that supports both process- and VM-level isolation and protection.

6.5. Sensitivity Analysis

The hyperparameters in our system comprise *the number of levels* and *the arity* of the subtrees in the MAC forest, which are used during the integrity verification of the pages.

We compute an 8-byte MAC for every 4 KB page. A 512 GB memory, contains 2^{27} MACs (1 MAC per page), which is the number of nodes in the lowest level of the MAC forest. We set the hyperparameters – q and p – such that the performance overhead is minimized. ***Subtree Level Analysis (q)*** As we have observed in SGX and Penglai, the number of levels in the integrity trees/forests correlate very well with the memory access overheads. Additionally, the number of levels also influence the storage overheads. Keeping both in mind, we chose ***q*** as **3** because it maximized our performance.

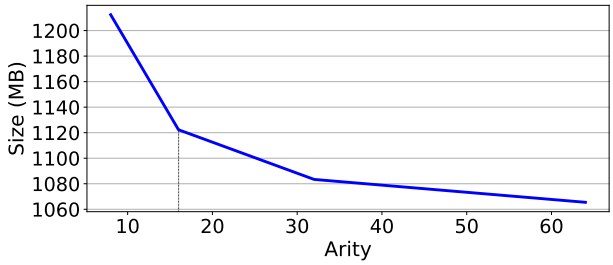


Figure 16: Storage overhead of varying arity of subtrees in the MAC forest

Subtree Arity Analysis (p) The storage and maintenance of the MACs is another concern for the MAC forest. The arity of the subtrees also dictates the number of memory accesses required while verifying or updating the MACs in the subtree region. Thus, we decide to keep the arity small. We set the arity of the higher level as half of that of the lower level to reduce the frequency of updates in the higher level nodes. Additionally, we plotted the the storage overhead for different values of the hyperparameter p (see Figure 16). The storage overhead shows a sharp decline in the beginning after which the descent is more gradual. We thus decided to select the arity close to the knee of the curve and set it to 16 for the lower level and 8 for the level above it..

The total storage space required for our forest is 1096 MB (for all three levels of the tree). Our Merkle Tree (for the EPC) has arity ($32 \times 32 \times 32$) and a size of 2.06 MB. Thus, the combined storage overhead of both these structures for 512 GB memory in our design sums up to 1098.06 MB. This is 8 times smaller than what the SGX-Client Merkle Tree would require (8322.06 MB) for securing 512 GB memory.

6.6. Security Analysis

SecScale ensures robust security guarantees (ACIF) across the complete system. Note that the EPC provides all four guarantees because it is a superset of SGX-Client. Let us thus focus on the eEPC.

▶ **Authenticity (A)** For authentication of the enclave pages in the eEPC, the key specifically contains an enclave specific enclave ID (enclave-level). This ensures that only the enclave that owns the page can access it. The MAC check for any other enclave including the OS will fail. They will not have a valid enclave id to construct the key that is needed to access (read/write) the page and recompute the MAC for verification. We thereby ensure the authenticity by cryptographic means. This satisfies Property 1.

▶ **Confidentiality (C)** is guaranteed by encrypting the data using standard AES-CTR mode encryption for the EPC and AES-ECB mode encryption for eEPC regions. This ensures that only the writing enclave can decrypt and access the original plaintext data.
▶ **Integrity (I)** In order to protect the data integrity of the eEPC, we maintain page-level MACs for each eEPC page. These MACs are protected with a multi-level MAC forest whose top-level nodes are stored in the EPC. Hence, any integrity violation will be caught in the MAC verification phase. The integrity of the Key Table (stored in the EPC) is established using the key to encrypt the hash of the page and construct the lowest level MAC. This satisfies Property 2.
▶ **Freshness (F)** is guaranteed by generating a new key every time a page is written back to the eEPC. We use a PRNG to generate the new key along with a bunch of other fields. We consider this to be secure enough given that we don't expect the same key to repeat in any practically relevant duration of time. However, if more security is desired then a global counter can be used.

6.6.1. Secure Execution and Side Channel Defence

The nature of data forwarding (before verification) in *SecScale* is very different from that in transactional memory systems or microarchitectural mechanisms such as branch prediction. Any security violation in *SecScale* is a catastrophic event – the system immediately shuts down. There are no rollbacks. The only way to exfiltrate data is if the learned values are sent to another system or durable storage. We shall show that this is not possible in our design.

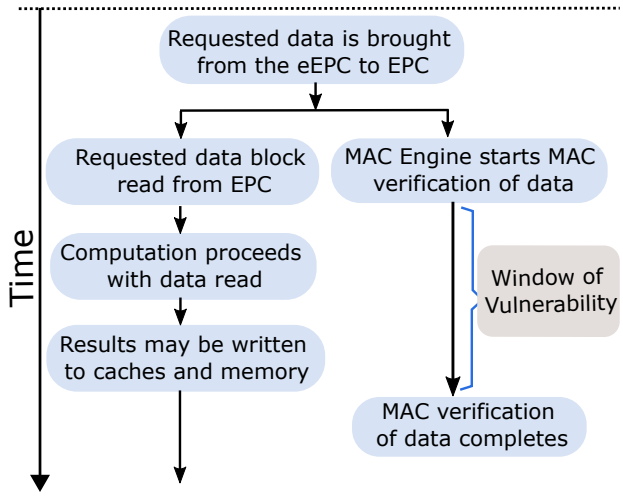


Figure 17: Window of Vulnerability

We define the time interval between the time that the data is *read* and *its MAC verification is complete*, as the *window of vulnerability* (as shown in Figure 17), which is 4.63 μs in our system. While the MAC verification is in progress, the I/O and network ports

are disabled such that no unverified data is exfiltrated to other systems. Even if unverified data is sent to another process that is running on the same system, it cannot send the data to any external entity, nor can it write it to permanent durable storage. If the attacker modifies the data along with its MAC to launch a *replay attack* corresponding to that address, only the lower two levels of the 3-level MAC subtree can be modified. Limiting the height of the MAC subtrees to 3 levels ensures a narrow *window of vulnerability* and consequently fast MAC verification. Since detecting a security breach at the end of MAC verification results in system shut down (no rollbacks), the corrupt data will never leave the system. As such, even if we assume that the adversary learns something from such a replay attack, there is no way to exfiltrate the data to an external entity, as we do not allow an I/O call to begin unless all pending MAC verifications are completed across cores. This satisfies Property 3. Thus, *SecScale* is protected against all such TOCTOU attacks (time-of-check time-of-use). There is no need to adopt more complex taint-based solutions as proposed in [51, 52].

SGX per-se is not immune to side-channel attacks. However, we can complement our design by adding elements of other designs to make it immune to side-channel attacks. For cache-based side channels, we can implement Mirage [41] or ScatterCache [42] that randomize physical addresses. Mirage defends against conflict-based attacks by fundamentally changing how cache evictions occur (like global random evictions, i.e., evicting from any random line in the cache). It is deployed at the last-level cache (LLC) so attackers can no longer learn victim access patterns via eviction set construction, which is the basis of Prime+Probe–style attacks. ScatterCache changes how memory addresses are mapped to cache sets. It utilizes the skewed-associative cache design, and each memory address is mapped to cache sets using a keyed cryptographic function. Its integration is entirely at the hardware cache controller level. The enclaves need no modification to implement either of these mitigation techniques. To randomize virtual addresses and avoid attacks that try to get the page-access sequence, we can use classic techniques over and above *SecScale*, such as Dr.SGX[43] and MoLE[44]. They add a dedicated compiler pass to dynamically randomize and re-randomize virtual addresses such that the page-access sequence cannot be used to obtain worthwhile information.

NOTE: System shutdown during a security breach is a well-accepted standard approach that helps keep the data from being exposed to the attacker and minimizes the effects of the attack. Intel SGX, for example, uses Asynchronous Enclave Exit (AEX) to safely stop enclave execution upon detecting exceptions or integrity violations, thereby protecting sensitive state. AMD SEV-SNP injects #VC exceptions and halts secure VM execution if integrity checks fail. Similarly, other secure systems employ similar mechanisms, like entering prevention modes and halting execution. Our approach aligns with these industry precedents. Moreover, cloud-based systems are expected to have redundancy, so a redundant system can ensure availability if one machine fails. The system treats a severe violation as a terminal event, triggering a controlled shutdown. This minimizes both the exposure window and potential cascading effects.

6.6.2. Security Analysis of the Page Table

The page table needs to be protected in designs that have large unrestricted, unsecure memories or in cases where enclave isolation is not guaranteed (e.g. Intel SGX and

Penglai). We argue that in *SecScale*, we do not need this kind of protection because of the following reasons. Let us list the possible attacks that the OS can mount using the page table as the via media.
Ⓐ *Secure → Unsecure Mapping:* This is not relevant in our case because in our system the entire memory is protected. However, this is a genuine problem in systems like Intel SGX and ARM TrustZone as they have large unsecure memories.
Ⓑ *Unsecure → Secure Mapping:* This cannot happen for the same reason as outlined in the previous point.
Ⓒ *Secure → Secure Mapping:* Another possibility is when the OS maps the secure page of an enclave to the secure region of another enclave. The unauthorized enclave cannot read or write the contents of the page since the enclave ID is a part of the key. The MAC check will fail and this will be a catastrophic event. An OS can only create an enclave or fully tear it down – it cannot access any page within it. For maintenance of enclave IDs, we use the same system as SGX-Client. Thus, Property 4 is satisfied.
Shared Memory: A small unencrypted memory (other than the EPC and eEPC region) is reserved for interaction with the OS and external components, which can be considered the shared memory region. *(In SGX, any memory outside the enclaves (EPC) is shared and unprotected, while CVMs typically designate memory pages as private or shared[53, 54, 55]).* Only the shared memory is utilized for communication with external entities such as memory-mapped I/O (MMIO) or device direct memory access (DMA).
Data-Sharing between Enclaves: Direct enclave-to-enclave memory sharing (i.e., mapped and hardware-encrypted) is not supported in *SecScale* (akin to SGX). When two enclaves want to share data with each other, they can perform local attestation and establish a protected secure channel. Further communication between the enclaves can take place through this channel. The application is responsible for securing the data it shares with the other enclaves.

Our security analysis shows that all the security properties specified as part of the design are satisfied by *SecScale*. As such, it is evident that *SecScale* offers secure and scalable TEEs.

7. Related Work

The size of the enclaves can be enhanced using two main approaches - by using bespoke secure systems designed for server applications or by using certain optimization techniques to enhance the enclave size (refer to Table 9).

7.1. Confidential Virtual Machines (CVMs)

Most state-of-the-art secure servers use a virtualization-based isolation mechanism to enable large enclave support and provide virtual machine (VM) level isolation. Although they extend the enclave to include the entire memory, they expand the trusted computing base (which includes the guest VM, along with its software stack), affecting the system's vulnerability.

AMD's *SEV-SNP (Secure Encrypted Virtualization - Secure Nested Paging)* [9] supports both main memory encryption and encrypted virtual machines (VMs). It does not provide freshness or protection against some physical attacks – attacking the DDR bus

Table 9: Comparison of related work. *Note:* ***Conf.*** *refers to Confidentiality,* ***Int.*** *refers to Integrity,* ***Fresh.*** *refers to Freshness,* ***HW (excl.)*** *refers to exclusively hardware based security,* ***Pg Fault Lat. Red.*** *refers to reduction in EPC page fault management latency and* ***'-'*** *means not applicable.*

		Security Components			***Security Features***		***Enclave Enhancement***	
System	**Arch**	**Conf.**	**Int.**	**Fresh.**	**HW (excl.)**	**Standalone Security**	**Scalable**	**Pg Fault Lat. Red.**
SGX-Client[5]	Intel	✓	✓	✓	✓	✓	×	×
SGX-Server[6]	Intel	✓	✓	×	✓	✓	✓	×
VAULT[25]	Intel	✓	✓	✓	✓	✓	×	-
Morphable Ctrs[33]	Intel	✓	✓	✓	✓	✓	×	×
TDX[11]	Intel	✓	✓	×	×	✓	✓	-
SEV-SNP[9]	AMD	✓	✓	×	×	✓	✓	-
CoSMIX[56]	Intel	✓	✓	✓	×	✓	×	×
DFP[23]	Intel	✓	✓	✓	×	✓	×	×
Penglai[24]	RISC-V	✓	✓	✓	×	×	✓	-
ARM-CCA[10]	ARM	✓	✓	×	×	✓	✓	-
SecScale	Intel	✓	✓	✓	✓	✓	✓	✓

while the VM is actively running. Additionally, it doesn't inherently provide full disk encryption. Intel *TDX (Trust Domain Extensions)*[11] provides security at the trust domain (TD) level, which is built upon the existing virtualization infrastructure. Both SEV-SNP and TDX expand the attack surface and do not protect the system against physical replay attacks. Alternatively, *SecScale* provides robust security guarantees while enabling vast enclaves, capable of supporting multi-tenant server workloads.

ARM's recent Confidential Compute Architecture (ARM CCA)[10] is also based on similar secure virtualization technologies. It introduces *Realms*, which enables isolated memory for secure execution, and a page-locking mechanism to support large enclaves (realms). However, CCA does not employ encryption and cannot defend against physical attacks like cold boot attacks, live probing, or replay attacks. In contrast, *SecScale* provides robust protection against such physical attacks through encryption and memory isolation, while supporting huge enclaves.

7.2. Bespoke Systems

In *VAULT* [25], the entire memory can serve as the EPC, which helps avoid penalties associated with EPC page faults. However, it implements a single Merkle Tree for the entire memory region, which leads to a large Merkle tree. The tree height increases with the memory size, leading to high memory access overheads. Although VAULT performs better than traditional SGX designs, its scalability is limited. In contrast, the small Merkle tree with the MAC forest in *SecScale* has very low overheads and is highly scalable. In Table 6, we have highlighted the impact of large Merkle trees and compared them with the MAC forest design of *SecScale*.

Morphable Counters[33] implements a high-arity Merkle tree and can dynamically change the size of selected minor counters based on the memory access pattern to reduce expensive overflows. Although increasing the arity of the tree reduces its height, it is not an optimal solution. *Morphable Counters* has a small EPC, resulting in huge EPC page fault penalties that severely degrade the system performance. The single large Merkle tree and EPC page fault penalties hinder its scalability. *SecScale*, however, offers a holistic solution by taking a novel, multi-faceted approach. It makes use of a small Merkle tree and a MAC forest along with the *read first, verify later* approach that together deal with the memory access overheads (refer Table 6 for comparison between Merkle tree and MAC forest) as well as the EPC page fault overheads.

A different approach is adopted in *Penglai* [24], which is a software-hardware co-designed system that creates dedicated hardware augmentations on a RISC-V core. There is one large EPC and its recipe for scalability is to mount sub-trees of the Mountable Merkle tree (MMT) on demand. It caches few subtree roots in the TCB. In the event of an LLC miss, if the MMT root is found in the MMT cache, then the counters can be verified with additional memory accesses. However, if there is a miss, then the penalty is quite large. Additionally, it relies on dedicated HW support to ensure that the memory region that stores the MMTs is not tampered with. This is not possible to ensure in the context of our threat model where we allow the attacker to modify any memory location at will.

7.3. Enclave Enhancement via Memory System Optimizations

CoSMIX [56] proposes a software cache to store evicted EPC pages. It instruments the application code such that accesses to enclave memory are replaced with accesses to allocated memory buffers, which reduces EPC page faults and gives the illusion of having enclaves with larger capacity. However, providing the same level of security in software as hardware offers is not possible [23].

Liu et al. [23] (DFP) attempt to decrease the number of EPC page faults on the critical path by prefetching pages into the EPC. They leverage sequential access patterns and use a list-based prefetcher. This approach is ineffective when memory accesses are random, and increased mispredictions could cause negative effects. Consequently, the accuracy of the predictor is low.

8. Conclusion

We introduced three new ideas in this paper, which allowed us to solve a problem that was known for a long time and had become a matter of great concern ever since Intel deprecated SGX-Client in 2021. Sacrificing *freshness* is the industry standard today mainly because providing it requires maintaining counters for every block and a Merkle tree, which are not scalable by design. We leveraged the fact that the catastrophic nature of a security verification failure can be used to take verification off the critical path. To ensure immunity from TOCTOU attacks, we employ two safeguards – the system shuts down in the event of an integrity verification failure, and all I/O calls are blocked until integrity verification completes. The state-of-the-art has put its full might behind protecting the integrity aspects of the key, such as the counters. However, we opt for a diametrically

different approach, where a read arrives at the processor with a key that has *supposedly been used to encrypt it*. This allowed us to create a MAC forest where we could verify the integrity of the key and the data together in a delayed fashion. A MAC forest has a much lower storage overhead than a Merkle tree and is a far more scalable alternative. These ideas and some design optimizations to reduce DRAM accesses allowed us to achieve a 56% speedup over our nearest competitor, VAULT, and a 58% speedup over SGX-Client.

References

[1] H Kanakadurga Bella and S. Vasundra, A study of security threats and attacks in cloud computing (2022).

[2] S. F. Yitbarek, M. T. Aga, R. Das, T. Austin, Cold boot attacks are still hot: Security analysis of memory scramblers in modern processors, in: 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), IEEE, 2017, pp. 313–324.

[3] Cost of a Data Breach Report 2023 , `https://www.ibm.com/reports/data-breach`, online; accessed 12 April 2024 (2023).

[4] S. Pinto, N. Santos, Demystifying arm trustzone: A comprehensive survey, in: ACM Computing Surveys, 2019.

[5] V. Costan, S. Devadas, Intel sgx explained, in: Cryptology ePrint Archive, TechnicalReport086, MIT., 2016.

[6] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, C. Rozas, Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave, in: HASP '16: Proceedings of the Hardware and Architectural Support for Security and Privacy, 2016.

[7] AMD, Amd memory encryption, in: Whitepaper, 2021.

[8] D. Kaplan, Protecting vm register state with sev-es, in: Whitepaper, 2017.

[9] AMD, Amd sev-snp: Strengthening vm isolation with integrity protection and more, in: Whitepaper, 2020.

[10] X. Li, X. Li, C. Dall, R. Gu, J. Nieh, Y. Sait, G. Stockwell, Design and verification of the arm confidential compute architecture, in: Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation, 2022.

[11] P.-C. Cheng, W. Ozga, E. Valdez, S. Ahmed, Z. Gu, H. Jamjoom, H. Franke, J. Bottomley, Intel tdx demystified: A top-down approach, in: Accepted in ACM Computing Surveys, 2024.

[12] M. El-Hindi, T. Ziegler, M. Heinrich, A. Lutsch, Z. Zhao, C. Binnig, Benchmarking the second generation of intel sgx hardware, in: DaMoN '22: Proceedings of the 18th International Workshop on Data Management on New Hardware, 2022.

[13] Trends of Cloud Computing and Cyber Security in 2023 , https://cionews.co.in/trends-of-cloud-computing-and-cyber-security-in-2023/, online; accessed 9 April 2024 (2023).

[14] Intel's sgx deprecation impacts drm and ultra hd blu-ray support, https://www.techspot.com/community/topics/intels-sgx-deprecation-impacts-drm-and-ultra-hd-blu-ray-support.273156/page-2#google_vignette, online; accessed 9 April 2024 (2022).

[15] Bill Toulas, New intel chips won't play blu-ray disks due to sgx deprecation, https://www.bleepingcomputer.com/news/security/new-intel-chips-wont-play-blu-ray-disks-due-to-sgx-deprecation/:~:text=Why%20did%20Intel%20abandon%20SGX,numerous%20vulnerabilities%20and%20attack%20methods, Online; accessed 9 April 2024 (2022).

[16] Jimmy Pezzone, Intel's SGX deprecation impacts DRM and Ultra HD Blu-ray support, https://www.techspot.com/news/93006-intel-sgx-deprecation-impacts-drm-ultra-hd-blu.html, online; accessed 9 April 2024 (2022).

[17] 4K Blu-Rays, https://rog-forum.asus.com/t5/promotions-general-discussions/4k-blu-rays/td-p/647492/page/2, online; accessed 9 April 2024 (2017).

[18] Tudor Malene, Demystifying SGX — Part 4— Secure hardware, https://medium.com/obscuro-labs/demystifying-sgx-part-4-secure-hardware-59cd09687d53, online; accessed 9 April 2024 (2023).

[19] Charlotte Bowyer, Understanding the growing threat of replay attacks, https://onfido.com/blog/understanding-the-growing-threat-of-replay-attacks/, online; accessed 9 April 2024 (2023).

[20] Web3 Security, Replay Attack in Blockchain Networks and Nodes, https://www.immunebytes.com/blog/replay-attack-in-blockchain-networks-and-nodes/, online; accessed 13 April 2024 (2023).

[21] E. Aktas, C. Cohen, J. Eads, J. Forshaw, F. Wilhelm, Intel trust domain extensions (tdx) security review, in: Google security review, 2023.

[22] K. Kollenda, General overview of amd sev-snp and intel tdx, in: Research Article, 2023.

[23] X. Liu, W. Wang, L. Wang, X. Gong, Z. Zhao, P.-C. Yew, Regaining lost seconds: Efficient page preloading for sgx enclaves, in: Middleware '20: Proceedings of the 21st International Middleware Conference, 2020.

[24] E. Feng, X. Lu, D. Du, B. Yang, X. Jiang, Y. Xia, B. Zang, H. Chen, Scalable memory protection in the penglai enclave, in: Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation., 2021.

[25] M. Taassori, A. Shafiee, R. Balasubramonian, Vault: Reducing paging overheads in sgx with efficient integrity verification structures, in: ASPLOS '18: Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, 2018.

[26] S. Gueron, A memory encryption engine suitable for general purpose processors, in: Cryptology ePrint Archive, 2016.

[27] D. Blazhevski, A. Bozhinovski, B. Stojchevska, V. Pachovski, Modes of operation of the aes algorithm, in: The 10th Conference for Informatics and Information Technology, 2013.

[28] B. Schneier, Applied cryptography: protocols, algorithms, and source code in C, john wiley & sons, 2007.

[29] Mark N. Wegman and J. Lawrence Carter, New classes and applications of hash functions (1979).

[30] L. Martin, Xts: A mode of aes for encrypting hard disks, IEEE Security & Privacy 8 (3) (2010) 68–69.

[31] M. van Dijk, J. Rhodes, L. F. G. Sarmenta, S. Devadas, Offline untrusted storage with immediate detection of forking and replay attacks, in: Proceedings of the 2007 ACM workshop on Scalable trusted computing, 2007.

[32] C.-C. Tsai, D. E. Porter, M. Vij, Graphene-sgx: A practical library os for unmodified applications on sgx, in: Proceedings of the 2017 USENIX Annual Technical Conference (USENIX ATC '17), 2017.

[33] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, J. A. Joao, M. K. Qureshi, Morphable counters: Enabling compact integrity trees for low-overhead secure memories, in: 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2018.

[34] D. Gens, Os-level software & hardware attacks and defenses, in: Proceedings of the 2018 Workshop on MobiSys 2018 Ph.D. Forum, 2018.

[35] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, E. W. Felten, Lest we remember: cold-boot attacks on encryption keys, in: Communications of the ACM, 2009.

[36] A. Naha, A. Teixeira, A. Ahlén, S. Dey, Sequential detection of replay attacks, in: IEEE Transactions on Automatic Control, 2022.

[37] M. Payer, T. R. Gross, Protecting applications against tocttou races by user-space caching of file metadata, in: VEE '12: Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environment, 2012.

[38] D. Tsafrir, T. Hertz, D. Wagner, D. D. Silva, Portably solving file tocttou races with hardness amplification, in: 6th USENIX Conference on File and Storage Technologies, 2008.

[39] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, , Y. Yarom, Spectre attacks: exploiting speculative execution, in: Communications of the ACM, 2020.

[40] K. Murdock, D. Oswald, F. D. Garcia, J. V. Bulck, D. Gruss, F. Piessens, Plundervolt: Software-based fault injection attacks against intel sgx, in: IEEE Symposium on Security and Privacy (SP), 2020.

[41] G. Saileshwar, M. Qureshi, Mirage: Mitigating conflict-based cache attacks with a practical fully-associative design, in: Proceedings of the 30th USENIX Security Symposium, 2021.

[42] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, S. Mangard, Scattercache: Thwarting cache attacks via cache set randomization, in: Proceedings of the 28th USENIX Security Symposium, 2019.

[43] F. Brasser, S. Capkun, A. Dmitrienko, T. Frassetto, K. Kostiainen, A.-R. Sadeghi, Dr.sgx: automated and adjustable side-channel protection for sgx using data location randomization, in: Proceedings of the 35th Annual Computer Security Applications Conference, 2019.

[44] F. Lang, W. Wang, L. Meng, J. Lin, Q. Wang, L. Lu, Mole: Mitigation of side-channel attacks against sgx via dynamic data location escape, in: Proceedings of the 38th Annual Computer Security Applications Conference, 2022.

[45] Standard Performance Evaluation Corporation, SPEC CPU® 2017, `https://www.spec.org/cpu2017/`, online; accessed 13 April 2024 (2017).

[46] S. R. Sarangi, R. Kalayappan, P. Kallurkar, S. Goel, E. Peter, Tejas: A java based versatile micro-architectural simulator, in: International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2015.

[47] H. Patil and R. Cohn and M. Charney and R. Kapoor and A. Sun and A. Karunanidhi, Pinpointing representative portions of large intel ® itanium ® programs with dynamic instrumentation (2005).

[48] Erez Perelman and Greg Hamerly and Michael Van Biesbrouck and Timothy Sherwood and Brad Calder, Using simpoint for accurate and efficient simulation (2003).

[49] A. Phansalkar, A. Joshi, L. K. John, Analysis of redundancy and application balance in the spec cpu2006 benchmark suite, in: Proceedings of the 34th annual international symposium on Computer architecture, 2007, pp. 412–423.

[50] Pin 3.21 User Guide, `https://software.intel.com/sites/landingpage/pintool/docs/98484/Pin/html/index.html`, online; accessed 13 April 2024.

[51] T. S. Lehman, A. D. Hilton, B. C. Lee, Poisonivy: Safe speculation for secure memory, in: Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016.

[52] W. Shi, H.-H. S. Lee, Authentication control point and its implications for secure processor design, in: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06), 2006.

[53] M. Misono, D. Stavrakakis, N. Santos, P. Bhatotia, Confidential vms explained: An empirical analysis of amd sev-snp and intel tdx, in: ACM SIGMETRICS Performance Evaluation Review, 2025.

[54] AMD, Amd sev-tio: Trusted i/o for secure encrypted virtualization, in: Whitepaper, 2023.

[55] Intel, Intel tdx connect architecture specification, in: Whitepaper, 2025.

[56] M. Orenbach, Y. Michalevsky, C. Fetzer, M. Silberstein, Cosmix: A compiler-based system for secure memory instrumentation and execution in enclaves, in: Proceedings of the 2019 USENIX Annual Technical Conference, 2019.