Revisiting Virtual Memory Support for Confidential Computing Environments

Haoyu (Henry) Wang *Member, IEEE*, Noa Zilberman *Senior Member, IEEE*, Ahmad Atamli *Member, IEEE*, and Amro Awad *Member, IEEE*

***Abstract*—Confidential computing is increasingly becoming a cornerstone for securely utilizing remote services and building trustworthy cloud infrastructure. Confidential computing builds on hardware-anchored root-of-trust that can attest the identity and authenticity of the remote machine, the configuration, and the running software stack, in an unforgeable way. In addition to the hardware-rooted verifiable attestation mechanism, confidential computing depends on strict run-time isolation of confidential computing tasks' data and code from each other and the other tasks, including privileged ones. Such isolation is achieved via on-chip access control and cryptographically once off-chip.**

Despite the wide support of confidential computing in most modern processors, e.g., AMD SEV-SNP and ARM CCA, there is minimal discussion of the effect of such support on the performance of conventional on-chip access control. Thus, in this paper we highlight the key changes in virtual memory support required for access control in confidential computing environments, and quantify their overheads. We propose an optimized design that enables improved performance by caching confidential computing access control metadata effectively. Two design options are proposed to balance hardware overhead and performance. We evaluate two configurations with different TLB entry coverage, which mirror Arm CCA GPC and AMD RMP, respectively. Our design improves performance by 12% over the baseline access control design and 6% over the state-of-the-art.

***Index Terms*—Confidential Computing, Access Control, Arm CCA, Hardware Security, Memory Virtualization, MMU.**

I. Introduction

As cyber-attacks increase, cloud customers demand stronger data protection. Users often lack visibility into the cloud provider's hardware and software. To reduce the risks stemming from physical attacks and potentially vulnerable supply chain in cloud systems, hardware vendors now offer *Confidential Computing* support. With hardware-rooted confidential computing support, customers can directly attest the hardware running their job and rely on its security features to protect the confidentiality and integrity of their applications, even in the presence of a malicious hypervisor. Confidential computing support relies on three aspects: (i) immutable hardware root-of-trust (RoT) that can verifiably report the boot sequence, platform status, and the loaded software, (ii) proper access control and isolation for confidential jobs' data and code while on the chip, and (iii) cryptographic data and code protection while off the trusted compute base (e.g., in off-chip memory).

Although confidential computing support is commonly announced and/or supported by major chip vendors [1]–[3], limited implementation details are made public. Meanwhile, confidential computing can incur an average performance overhead of 24.5% by AMD SEV-SNP, as shown in Table I, which emphasizes the need for more efficient hardware designs and implementations. While the hardware-rooted measured boot attestation report and secure session establishment are incurred infrequently (e.g., at the job start time), off-chip data protection and access control overheads can lead to significant run-time performance degradation. To the best of our knowledge, there is a scarcity of research that explores the design space for access control implementation in confidential computing.

Haoyu (Henry) Wang, Noa Zilberman, and Amro Awad are with the University of Oxford. Ahmad Atamli is with University of Southampton.

II. Background: Confidential Computing in Virtualized Environments

The foundation of access control in modern confidential computing architectures is hardware-enforced memory isolation across co-existing domains. A confidential computing domain gains trust from (i) a minimal verifiable privileged software stack, (ii) hardware-enforced isolation, and (iii) hardware-rooted attestation. For example, ARM CCA [3] defines three domains: realm (confidential computing), secure (trusted services), and normal (OS and hypervisor). This isolation relies on an additional access control layer, which we call the *confidentiality check*. Since each domain supports multiple privilege levels, isolation is achieved hierarchically: OS manages process page tables, hypervisor manages nested VM page tables, and a minimal firmware establishes hardware memory assignment through a **Vertical Check Table (VCT)**. Conceptually, VCT is analogous to ARM's Granule Protection Table (GPT) [3] and AMD's Reverse Map Table (RMP) [4].

In a virtualized environment, a single memory reference can trigger a complex, two-dimensional address translation. The guest OS manages the first stage mapping (guest virtual to guest physical address, gVA-gPA, stored in the guest page tables), while the hypervisor manages the second stage (guest physical to system physical address, gPA-sPA, stored in the nested page tables). The page table walker, which is generally in the MMU, issues a sequence of memory reads to fetch the corresponding page table entry. Permission and access bits in the page table entries are checked. In the worst-case, a Translation Lookaside Buffer (TLB) miss can trigger two-dimensional walks requiring up to *4 (4-level guest page walk) * 4 (4-level nested page walk) + 4 + 4 =* ***24*** memory accesses to resolve a single address [5] [6] with a final memory access by the CPU to fetch the data. A large number of memory accesses are required because each step of the guest page table walk produces a guest physical address, which must then be

TABLE I: Performance Evaluation of AMD SEV-SNP: AMD Milan, 2 vCPUs; GAP Benchmark Suite, Scale 20.

Metric	Value
Normalized, geometric mean (SEV-SNP/Base)	0.75
Average overhead, arithmetic mean (%)	24.5

translated through multiple levels of nested page table walks to obtain the corresponding system physical address.

III. Motivation

However, to achieve the access control with confidentiality check, each access to system physical memory, including those required for address translation and the target memory address access, requires a VCT check to ensure that the target address can be accessed by the world initiating the request. Thus, even a conservative implementation of VCT that merely captures the world each physical page belongs to can incur **50 memory accesses for a single memory access**[1]. Note that this assumes a linear VCT implementation and hence a single memory access to obtain confidentiality check metadata. Frequent VCT checks during memory accesses add latency to the memory path and decrease instructions per cycle (IPC). We evaluated an AMD SEV-SNP confidential VM on an AMD Milan processor against a baseline system without SEV-SNP on the same processor, using the GAP benchmark suite. Table I shows a 24.5% mean IPC overhead.

While prior work has provided informative measurements of confidential computing overheads on specific platforms [7]–[10], these studies primarily evaluate end-to-end performance and extend confidential computing to additional platforms. In contrast, they do not describe the hardware changes required to support confidentiality checks or the alternative designs to implement and optimize them. The ARM RME design guide recommends a PAS-tagged cache and a world/security state-tagged TLB, but it does not provide performance data, and we believe the TLB design can be further optimized. A few papers examine architecture design, metadata bit insertion policies, and the placement of confidentiality checks, for example, separate paths near the MMU and CPU versus an integrated mechanism near the MMU.

In this paper, we explain and quantify the overheads of access control in confidential computing, and propose an effective mechanism to minimize such overheads. To this end, we investigate the overheads incurred by confidentiality check when VCT is linear vs. two-level, study the effectiveness of caching VCT metadata on the different memory access paths, and propose an optimization that embeds verification result as a part of each TLB entry to eliminate the need for checking a VCT cache in parallel with each TLB access. Our design achieves an average of 6% performance improvement over the `Shelter` [11].

IV. Threat Model

Our threat model is similar to prior work in confidential computing [1], [3]. We assume a powerful adversary with full control over the privileged software stack, including the hypervisor and host OS, which are considered untrusted and potentially malicious [4]. The Trusted Computing Base (TCB) is thus minimized to the CPU hardware and its specific trusted firmware. The attacker's primary objective is to compromise the confidentiality and integrity of a target guest VM's data-in-use by subverting its memory isolation. While physical attacks and certain side-channels are often considered out of scope, attacks leveraging software control are central to the confidential computing threat landscape.

[1]24 page table walks each requiring 24 VCT checks, plus final memory access and its corresponding VCT check.

V. Confidentiality Check Implementation

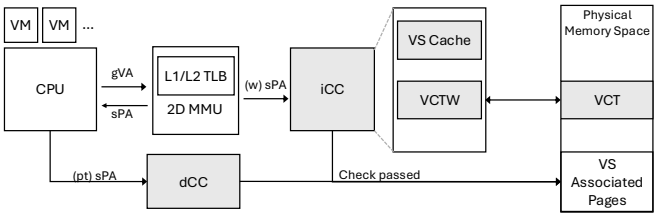

Fig. 1: Intermediate and Direct Confidentiality Check (i/dCC) Located on the Memory Access path. (Grey boxes: trusted CC related modules.)

As shown in Figure 1, confidentiality check (CC) must occur on both the direct memory access path and the intermediate memory access path, as depicted by `dCC` and `iCC` blocks, respectively. dCC checks if the resulting sPA to be accessed belongs to a Vertical Space accessibly by the current world. Meanwhile, iCC applies the same checks as dCC but for the memory requests resulting from the page table walking process. As shown in the figure, i/dCC can include logic for VCT Walking and a VS Cache, respectively. We first discuss the basic split CC design, followed by an alternative unified CC design that writes back to the verified TLB with an additional VCT walk for the final sPA.

A. The fundamental implementation: Split CC

The fundamental flow of a conventional confidentiality check, such as ARM GPC [12], employs two-level address translation from gVA to sPA to isolate two distinct Vertical Privilege Levels (VPL). During address translation, the PTW logic accesses the actual tables in physical memory, requiring further verification of security metadata by walking dedicated metadata tables. Figure 1 shows an iCC, situated between the MMU and physical memory, performing metadata checks on sPA accesses during PTW by requester side. A Vertical Check Table Walk (VCTW) logic, is integrated within the iCC to traverse the Vertical Check Table (VCT) stored in physical memory. Consequently, each PTW issued by the MMU necessitates an additional VCTW on the VCT. Therefore, after the check by the iCC passes, the pages associated with Vertical Spaces (VS) must be traversed to complete the translation. To accelerate this vertical checking process, a Vertical Space Cache (VSC) is integrated into the iCC as a widely-used way [3]. Most recently fetched VS tags are stored in the VSC. With the presence of the VSC, every sPA that requires checking by the iCC can be immediately validated upon a cache hit, eliminating the need for an additional metadata lookup from the VCT in memory.

Another path integrated with the confidentiality check module shown in Figure 1, is between the CPU and physical

memory, which is more critical than the PTW path due to the direct impact of data acquisition latency on CPU pipeline performance. This parallel access using the final sPA requires verification by the dCC, which shares an identical structure with the iCC. Therefore, a VCT walk is required if no previous verification result associated with the VS tag exists in the VSC. VSC significantly accelerates this verification flow, delivering native performance when it hits. The high parallelism is provided by double CC modules, but with higher hardware overhead.

B. Verified-TLB-Based Design: unified CC

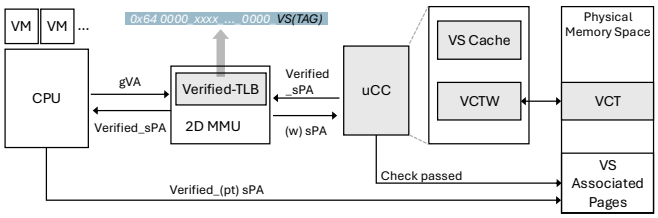

Fig. 2: Unified Confidentiality Check Locating on the Memory Access path, with Verified TLB enabled

To reduce the additional dCC hardware overhead on the CPU data path, we propose an alternative design. A Verified TLB is integrated into the 2D MMU to store the verified final sPA along with the additional VS tag returned from the iCC. In this design, the final sPA is checked by the iCC instead of being verified by the parallel dCC. We refer to the combination of the Verified TLB and the iCC as the unified CC (`uCC`), as shown in Figure 2. This approach removes the dCC but introduces an additional verification step for the final sPA.

The Verified TLB is developed by extending the TLB to store a VS tag alongside each sPA. After the uCC verifies an sPA, the ⟨sPA, VS tag⟩ pair is inserted into the Verified TLB. On a subsequent hit, the CPU can access memory by the final sPA verified by uCC. Workloads with high locality benefit from this design, as higher Verified TLB hit rates reduce the overhead imposed by the CCA mechanism. On a Verified TLB miss, the access falls back to the uCC path for verification. However, workloads with low locality may experience worse performance due to the additional VCT walk.

VI. Methodology and Results Evaluation

Our simulation experiments are conducted within the SST framework [13]. Table III summarizes the primary architectural parameters of our simulated system and the benchmark suites. We modeled a 2b-2Levels CC design analogous configuration of ARM GPC [12], and a 16B-Flat CC design with a same entry size (including the embedded Virtual Machine ID) representing a 4 KB granule, analogous configuration of AMD's RMP [4] design. We selected benchmarks known for their memory intensity, with eight of them having over 45 Misses Per Kilo Instructions (MPKI) as shown in Table II. The minimum MPKI among all selected benchmarks is around 10 (`NPB-is`), while the maximum reaches 140 (`NPB-cg`). This selection covers workloads ranging from cache-friendly to cache-stressed, enabling a balanced evaluation under varying memory intensities. All benchmarks were executed in single-threaded mode.

TABLE II: Average L1 MPKI by benchmark suite.

Suite	GAP(5)	NPB(5)	PARSEC (canneal)	XSBench
Avg. L1 MPKI	51.18	83.81	26.39	13.78

TABLE III: Hardware Configuration and Benchmarks

CPU Core	
Clock Frequency	2 GHz
Core Count	1
Max Instructions	100 Million
2-Level-Translation MMU (Samba [14])	
Translation Mode	2-Stage Nested Translation
L1/L2 TLB Hierarchy	64 entries; 1536 entries
Page Walk Cache (PTWC)	4-level, 32-entry/level, 4-way
2b-2Levels CC Model	
Protection Granularity; VS Tag	4 KB; 2 Bit
VS Cache Entry Size, #entries, Replacement Policy	16 B, 64, LRU;
VS Table Covers/Cache Entry	256 KB
16B-Flat CC Model	
Protection Granularity; VS Tag	4 KB; 16 B
VS Cache Entry Size, #entries, Replacement Policy	16 B, 64, LRU;
VS Table Covers/Cache Entry	4 KB
Memory Subsystem	
Private L1-D Cache / Private L2: Cache Size & Latency (Cycles)	32 KiB / 256 KiB, 1 / 2
Shared L3 Cache: Cache Size & Latency (Cycles)	1 MiB, 15
DRAM Access Time	100 ns
DRAM Controller Clock	1.2 GHz

A. Overall Performance Evaluation on Design Options

Our IPC results in Figures 3 and 4 are normalized to the 1D Native (Non-CC) configuration. The blue bars represent 1D Native without CC logic; the orange bars show 2D Nested translation without CC; the yellow bars indicate the split CC; the green bars present the unified CC; and the gray "CC-Base" bars denote the CC baseline without VS cache acceleration. Average values are listed in the legend.

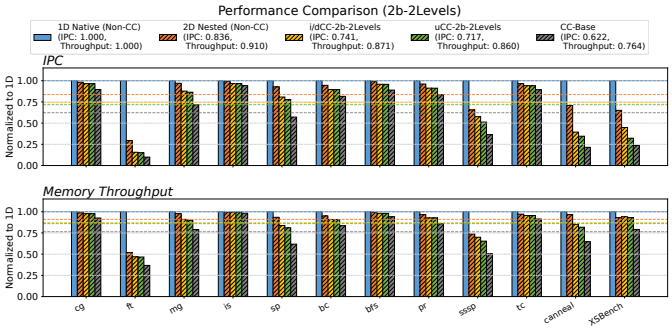

Fig. 3: Instruction Per Cycle and Memory Throughput of: Native 1D PTW, Nested 2D PTW, 2b-2Levels Split and Unified CC design, and Baseline Check Design

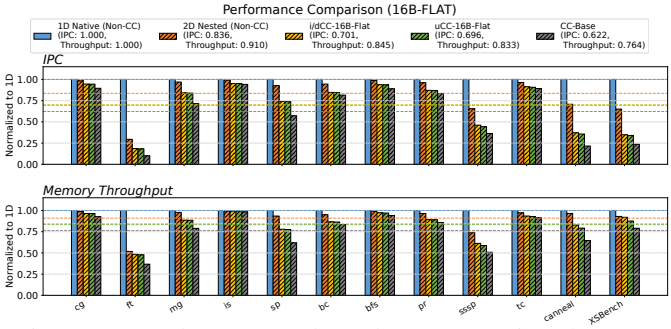

Fig. 4: IPC and Memory Throughput Comparison between 16B-Flat CC design with other designs.

Hardware-assisted virtualization is the first source of overhead. With two-dimensional page walks, average IPC drops to about 84% of the 1D Native (Non-CC). The loss follows from higher TLB-miss penalties described in Section III. Adding confidentiality checks without optimization (Base-CC) introduces a further reduction of roughly 22% relative to 2D Nested (Non-CC), because each page walk and data access

now requires a check, without any cache or TLB logic to accelerate it. Our goal is to narrow the performance gap between CC integrated design with the 1D Native (Non-CC) design.

In the unified CC design, or uCC, the VS Caches and Verified TLB improve performance by caching VS tag and verified ⟨sPA, VS tag⟩ pairs, respectively. On a Verified TLB hit, the CPU uses the verified sPA without an additional uCC check. On average, we observe a 10% gain for the 2b-2Levels CC and a 7% gain for the 16B-Flat CC. The benefit is higher for workloads with strong locality, where the Verified TLB achieves higher hit rates, such as `mg`, `sp` and `sssp`.

In the split CC design, or i/dCC, the extra Verified TLB check is not required, because the dCC performs the parallel check for CPU memory operations. Therefore, there is an additional average improvement of **12%** on 2b-2Levels CC and **8%** on 16B-Flat CC compared with the Base-CC design. Comparing to the 2D Nested results, the best design introduces only average 9% overhead on 2D Nested design, which outperforms [11] by 6%.

Across the evaluated workloads, both Figures 3 and 4 show that the base-CC limits an average normalized memory throughput to 76% of the 1D Native (non-CC) configuration, while other split and unified CC designs improve the throughput to above 83%. This strong correlation between memory throughput and IPC exists, meaning the CPU's instruction throughput is fundamentally gated by the performance of the memory hierarchy.

B. Analysis of Performance on Two Configurations

We evaluate different CC configurations by adjusting the granule mapping size for each VS Cache entry in both 2b-2Levels and 16B-Flat CC designs. The 2b-2Levels configuration demonstrates better performance because each entry covers more granules. In contrast, the 16B-Flat configuration requires an extra 16 bytes to cover a single 4KB granule, resulting in more cross page accesses due to its smaller coverage. The performance difference between the two configurations is 4% for the split CC and 2.1% for the unified CC in Figure 3 and 4.

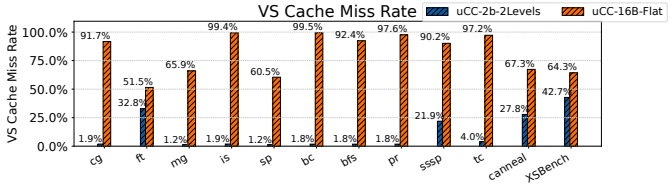

Fig. 5: VS Cache Miss Rate of 16B-Flat/2b-2Levels uCC.

To analyze these differences, we use the VS Cache miss rate in Figure 5. Workloads with irregular or sparse access, such as `cg`, `is`, and the graph set `bc`, `bfs`, `pr`, `sssp`, `tc`, show very high miss rates under the 16B-Flat uCC, often close to 100%, which triggers frequent VCT walks and reduces IPC. The 2b-2Levels uCC lowers miss rate because one entry covers a 256KB region with the smaller VCT metadata (2 bits) and captures spatial reuse even when page level locality is poor.

To reflect a practical Verified TLB in the uCC design, a 16B per entry extension is unattractive: it can nearly triple TLB storage (8B entry vs. 24B entry) in all TLB levels, whereas a Split-CC design would only introduce a relatively small VSC cache in the dCC block on the direct memory access path.

VII. Conclusion

In this paper, we pointed out, evaluated, and mitigated the critical performance overhead imposed by hardware access control mechanisms in modern Confidential Computing Architectures (CCAs). We introduced a novel architectural design featuring flexible CC configurations, which provides the community with two additional design options for the hardware and performance overhead trade-off. Our simulation results demonstrate that high-performance design improves performance by 6% over `Shelter` [11] by effectively accelerating these checks. Future investigations could focus on developing adaptive VSC caching policies and an enhanced Verified TLB design to better accommodate workloads with poor locality, thereby further mitigating the performance impact on memory-intensive applications.

References

[1] Intel Corporation, *Intel Trust Domain Extensions (TDX) White Paper*, February 2022, White paper.

[2] Advanced Micro Devices, Inc., *AMD64 Architecture Programmer's Manual, Volumes 1–5*, Advanced Micro Devices, April 2024, publication No. 40332, Revision 4.08. [Online]. Available: https://www.amd.com/system/files/TechDocs/40332.pdf

[3] Arm Limited, "Learn the architecture: Introducing arm confidential compute architecture," March 2025, document ID: DEN0125_400_en, Version 4.0. [Online]. Available: https://developer.arm.com/documentation/den0125/400

[4] Advanced Micro Devices, Inc., "Amd sev-snp: Strengthening vm isolation with integrity protection and more," Tech. Rep., January 2020, white Paper. [Online]. Available: https://www.amd.com/system/files/TechDocs

[5] R. Bhargava, B. Serebrin, F. Spadini, and S. Manne, "Accelerating two-dimensional page walks for virtualized systems," in *Proc. 13th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2008, pp. 26–35.

[6] S. Chai, J. Zhang, J. Kim, A. Wang, F. Chung, J. Stojkovic, W. Jia, D. Skarlatos, J. Torrellas, and T. Xu, "Emt: An os framework for new memory translation architectures," in *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, 2025, pp. 711–729.

[7] X. Li, X. Li, C. Dall, R. Gu, J. Nieh, Y. Sait, and G. Stockwell, "Design and verification of the arm confidential compute architecture," in *Proc. 16th USENIX Symp. Oper. Syst. Des. Implement. (OSDI)*, 2022, pp. 465–484.

[8] B. Dong and Q. Wang, "Evaluating the performance of the deepseek model in confidential computing environment," *arXiv:2502.11347*, 2025.

[9] Y. Yang, M. Sonji, and A. Jog, "Dissecting performance overheads of confidential computing on gpu-based systems," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, 2025, to appear.

[10] K. Vaswani, S. Volos, C. Fournet, A. N. Díaz, K. Gordon, B. Vembu, S. Webster, D. Chisnall, S. Kulkarni, G. Cunningham *et al.*, "Confidential computing within an AI accelerator," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, 2023, pp. 501–518.

[11] Y. Zhang, Y. Hu, Z. Ning, F. Zhang, X. Luo, H. Huang, S. Yan, and Z. He, "Shelter: Extending arm cca with isolation in user space," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 6257–6274.

[12] Arm Limited, "Arm® realm management extension (rme) system architecture," Arm Limited, System Architecture Specification ARM DEN 0129, October 2022, version A.d.

[13] A. Rodrigues, S. D. Hammond, S. Hemmert, C. Hughes, J. Kenny, and G. Voskuilen, "A-SST Initial Specification," Sandia National Laboratories, Albuquerque, NM, Tech. Rep. SAND2021-12371, October 2021.

[14] A. Awad, S. D. Hammond, G. R. Voskuilen, and R. J. Hoekstra, "Samba: A detailed memory management unit (mmu) for the sst simulation framework," Sandia National Lab. (SNL-NM), Albuquerque, NM, USA, Tech. Rep., 2016.