2026 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 979-8-3195-0602-3/26/$31.00 ©2026 IEEE | DOI: 10.1109/IPDPS65963.2026.00075

SecPerf: Demystifying Cost of Confidential HPC

Marcin Chrapek, Siyuan Shen, Patrick Iff, Tiancheng Chen, Mikhail Khalilov,
Marcin Copik, Maciej Besta, and Torsten Hoefler
Department of Computer Science,
ETH Zurich
Zurich, Switzerland
{marcin.chrapek, siyuan.shen, patrick.iff, tiancheng.chen, mikhail.khalilov}@inf.ethz.ch
{marcin.copik, maciej.besta, torsten.hoefler}@inf.ethz.ch

***Abstract*—Confidential Computing (CC) is becoming a de facto security standard in data centers, as evidenced by the growing number of hardware platforms that offer CC capabilities. CC addresses the needs of applications that process sensitive data across fields such as artificial intelligence (AI), finance, and healthcare. As we observe the convergence of cloud and high-performance computing (HPC), confidential HPC (cHPC) is becoming increasingly crucial for meeting the performance requirements of these compute-intensive workloads. Yet, the current understanding of end-to-end performance and cost trade-offs of cHPC's hardware remains unclear, despite being essential for the broad adoption and design of cHPC systems. We address this gap by introducing SecPerf, the first performance benchmarking platform tailored explicitly to secure cHPC solutions. Since performance is directly tied to the guarantees provided, SecPerf introduces a taxonomy of security protections that clarifies what is offered at what cost. SecPerf also categorizes the abundant security hardware in modern HPC nodes and supports extensibility by leveraging the Docker framework. We demonstrate SecPerf's potential by evaluating the performance and cost of primitives, secure networking, offloading, computing, and acceleration on state-of-the-art hardware across 25 algorithms, 18 network primitives, and configuration variants of 29 modern applications. Our results highlight key performance bottlenecks and deepen understanding of the cost of cHPC. By providing a unified benchmarking platform and empirical insights, SecPerf lays the groundwork for the future development and adoption of cHPC.**

***Index Terms*—Security, Confidential Computing, Performance, Benchmarking, Accelerators, Cloud Convergence, TEEs**

I. Introduction

Modern compute-intensive applications increasingly operate on confidential data, marking a shift toward a data-centric era driven by advances in artificial intelligence (AI). AI-driven systems are now deployed in highly sensitive domains, such as healthcare [1], finance [2], legal processing [3], and language translation [4], where breaches such as loss of model weights can result in substantial legal and financial repercussions [5]. As these and other high-performance computing (HPC) workloads are increasingly deployed on cloud-converged clusters, traditional trust assumptions for administrators and cluster users are being increasingly invalidated. This erosion of trust motivates organizations to adopt costly private deployments, limiting scalability and elasticity. Coupled with mounting regulatory pressure [6], [7] and rising user awareness [8], these dynamics elevated security to a first-class concern in the cloud and HPC environments, giving rise to the emerging paradigm of confidential HPC (cHPC) [9].

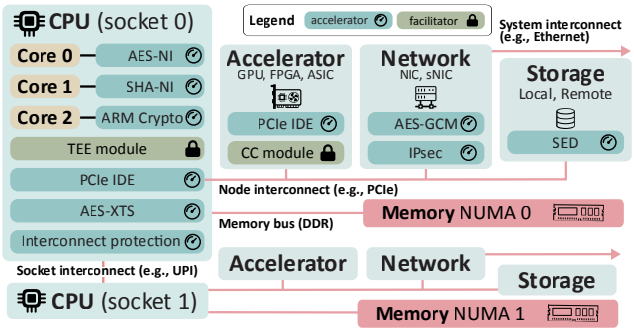

Fig. 1: An overview of secure hardware in modern HPC nodes categorized as in Section III-A.

In response to growing security demands, vendors have introduced a wealth of security-related hardware. Figure 1 shows a simplified view of a modern HPC node with examples from existing products, which we discuss in Section III-A. Notably, the paradigm of confidential computing (CC) has emerged as a de facto security standard [9]. CC addresses the security needs of confidential applications leveraging Trusted Execution Environments (TEEs), which shield users' data and programs from administrators and other tenants at the hardware level [10], [11]. The relevance of CC has been emphasized through expansions to GPUs (NVIDIA's H100s [12]) and PCIe (TEE Device Interface Security Protocol (TDISP) [13], Integrity and Data Encryption (IDE) [14]).

Although secure hardware is prevalent across all significant node components, understanding the performance costs and their trade-offs with guarantees is often lacking. As more security features proliferate across academia and industry, gaining such understanding is crucial for the broad deployment of cHPC and the design of future solutions. As outlined in Section II, existing performance studies of secure hardware focus on a single system, omit end-to-end guarantees, and employ workloads that are not cHPC-centric. Crucially, none of these efforts provides a unified platform that enables practitioners and designers to systematically compare secure systems based on their security assurances and costs.

Our research addresses the above gap by introducing SecPerf, a first-of-a-kind unified benchmarking platform for secure systems. Similar to how MLPerf [15] and SeBS [16]

1530-2075/26/$31.00 ©2026 IEEE
DOI 10.1109/IPDPS65963.2026.00075

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

have driven the understanding of trade-offs between different AI and Function-as-a-Service solutions, SecPerf achieves the same for secure systems. SecPerf introduces two taxonomies that categorize secure systems by the guarantees they provide and their purpose. The former is critical as performance is typically in trade-off with security. We design a categorization that is generalizable across secure systems, based on trust and concepts of confidentiality, integrity, and freshness for data in use, at rest, and in motion. The latter is crucial to understanding the interfaces of solutions. We introduce a categorization with generalizable classes based on hardware purpose: facilitators, accelerators, and their subtypes. Facilitators enable security (e.g., trust), while accelerators protect data and improve performance (Section III-A).

Based on these, SecPerf provides unified and expandable benchmarking interfaces. For programmable accelerators, it offers C++ templates for primitives. For transparent accelerators and facilitators, it provides a benchmarking Python interface specifying details such as parameters and installation instructions. SecPerf then offers a runtime measurement scaffolding, evaluating not only performance but also other costs such as energy or resource usage (e.g., cores). We open-source our reference implementation of SecPerf with Docker support, enabling easy portability and extensibility. While we focus our evaluation on cHPC critical solutions, SecPerf also generalizes to other systems that may not be hardware-accelerated, such as those that use homomorphic encryption.

We use SecPerf to identify areas for development in cHPC. We run SecPerf on six modern systems, evaluating secure hardware in CPUs, GPUs, NICs, and smartNICs, from Intel, AMD, ARM, and NVIDIA. Our study encompasses all significant areas of cHPC, including networking, offloading, acceleration, and computation. For programmable accelerators, we present results from 25 algorithms that implement cryptographic primitives, such as hashing and authenticated encryption with associated data (AEAD). These set the upper bound on performance for any solutions that leverage them. We then show characteristics of transparent accelerators that form the basis for facilitators. For example, we highlight that secure networking with technologies such as IPsec is currently neither scalable nor performant. We finish with an analysis of facilitators by comparing the performance of widely available TEEs, such as TDX [17], SGX [18], SEV-SNP [19], and confidential GPUs [12]. To the best of our knowledge, SecPerf is the first benchmarking platform for evaluating the performance of secure systems, providing a comprehensive and in-depth view of the cHPC hardware landscape.

In summary, our contributions are:

- We introduce SecPerf, the first end-to-end benchmarking platform for continuous evaluation, analysis, and comparison of secure system performance and cost.
- We propose generalizable security and system taxonomies that contrast security guarantees with performance, enabling expandability and facilitating system comparisons.
- We implement and open-source (github.com/spcl/secperf) a reference version of SecPerf ready for containerized evaluation.
- We conduct an extensive evaluation of SecPerf across six cHPC systems, covering 25 cryptographic primitive implementations, secure networking, offloading, and widely available CPU and GPU TEEs, sharing 11 novel key insights and 8 design recommendations.

II. Related work

We compare SecPerf with related performance works on four core design requirements, summarizing the results in Table I.

1 **Generalizability** (§III-A) A benchmarking platform requires a method to classify different secure hardware across provided guarantees and interfaces. This is necessary as understanding performance in secure systems is meaningless without understanding the protections. While TEEMon [20] and TEEBench [21] are closest to being benchmarking platforms, even they do not provide such a framework. To the best of our knowledge, SecPerf is the first work to introduce generalizable taxonomies of security and hardware type to differentiate solutions in the context of performance benchmarking.

2 **Comprehensiveness** (§III-B) Because heterogeneous HPC encompasses a wide range of systems, a benchmarking platform should support many of them out of the box. Existing performance studies focus primarily on TEEs. Some evaluate the general performance of one [22]–[25], two [26], [27], or three [28], [29] TEE implementations. Other works focus only on encrypted TEE memory [27], [30]. Some establish AI inference performance on one [31]–[35] or three [36] TEEs, sometimes providing additional performance metrics and optimizations [24], [32], [35], [36]. While designed to support more than one CPU TEE, TEEMon [20] and TEEBench [21] only evaluate one implementation. In contrast, SecPerf covers holistically cHPC hardware, encompassing programmable accelerators, network scalability, distributed applications, memory, NUMA, CPU, and GPU compute overheads.

3 **Extensibility** (§III-A, III-B) A cHPC benchmarking platform needs an extensible and portable methodology, as each heterogeneous HPC hardware solution might require unique benchmarks and architectures. None of the works provides an explicit extension flow. Even though lacking any guidance, some studies [21], [25]–[27], [31], [34], [36] could potentially be expanded to other hardware of the same type as their code is available (e.g., from one to another VM-based TEE). While TEEMon [20] is the only one using containers, its code is not open. We ensure expandability by outlining an explicit expansion flow and portability by leveraging Docker.

4 **Relevance** (§III-B, III-C) As a branch of HPC, cHPC benchmarking must involve its unique workloads and setups. While educational, focusing solely on single-node or single-GPU workloads [23], [24], [28]–[34], [36], [37] and traditional data center applications [20]–[22], [27], such as Redis or NGINX, is not typical for HPC. Only a few works show HPC workloads [25], [26], [35]. While SecPerf can integrate workloads from other domains, we present setups relevant to cHPC, such as distributed computation, and investigate resource costs, such as energy and core usage, which are

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

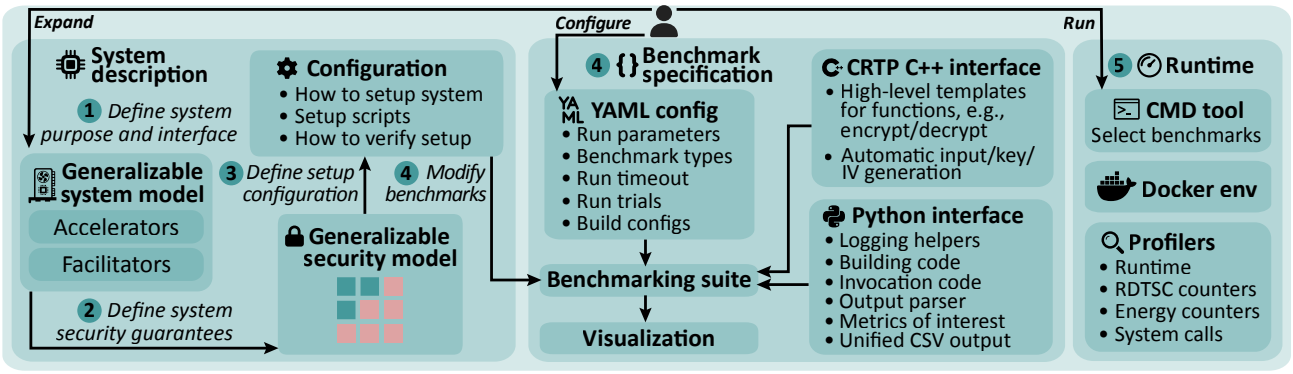

Fig. 2: SecPerf methodology relying on three pillars: system description, benchmark specification, and runtime.

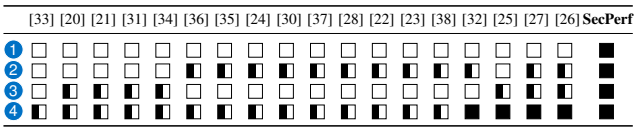

TABLE I: Comparison of secure system performance benchmarking studies. ■ good, ◧ partial, and □ no support.

critical in large-scale HPC deployments.

III. SecPerf design

We describe SecPerf's key components, shown in Figure 2: the *system description*, which configures the secure environment; the *benchmark specification*, which defines benchmarks; and the *runtime*, which provides a measurement environment.

A. System description

System descriptions enable users evaluating solution performance to understand their guarantees, limitations, and interfaces from manufacturer- or designer-prepared documentation. It involves four steps (1-4 in Figure 2) where SecPerf introduces generalizable security and system categorizations.

1 System taxonomy: The primary goal of the system taxonomy is to inform about hardware interfaces and purpose. It also paints the landscape of cHPC hardware, allowing us to incorporate the most relevant solutions into SecPerf. Existing classifications [29], [39]–[43] either focus on low-level details

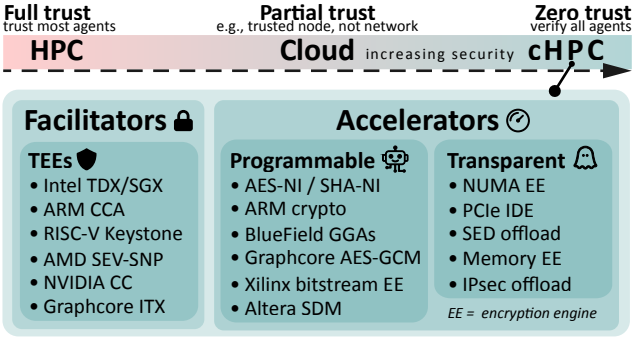

Fig. 3: The categorization of secure systems into facilitators, accelerators, and their corresponding subtypes.

that are too fine-grained to guide a benchmarking suite or concentrate on specific mechanisms (e.g., TEEs), typically not covering hardware such as transparent accelerators, which are central to heterogeneous cHPC nodes and secure storage and network protocols (Section IV-C). Our taxonomy is designed to be directly usable when selecting and comparing secure hardware for cHPC.

We classify secure systems into two generalizable classes based on their functionality: *facilitators* and *accelerators*. Facilitators enable foundational security features (e.g., trust, Section III-A), which are frequently infeasible to achieve with software alone. Facilitators usually lack unified interfaces and are tailored to the specific guarantees provided. Accelerators enhance the performance of security-critical operations and are often functionally replicable in software, albeit at a performance cost (Section IV-B). Typical existing accelerators focus on protecting data flow between facilitators that provide trust. Figure 3 shows examples of predominantly hardware-based systems. These can also be purely software-based or use lower-level hardware, such as IPsec leveraging AES-NI for acceleration.

TEEs are instances of a facilitator and are formally defined as secure environments that guarantee the authenticity, integrity, and confidentiality of user data, programs, and runtime states (e.g., CPU registers and memory) [43]. In addition, TEEs must be attestable by remote verifiers [44], [45]. Attestation is a process in which hardware provides verifiable evidence [46]–[48] that its state is as expected, thereby establishing trust and securing communication [44]. The evidence provided by TEEs can be verified as authentic by leveraging vendor-specific services [46]–[48], and usually includes a public key used to open a secure channel. Although exact implementations vary [44], attestation establishes trust and secure communication before sending sensitive data or keys. Major manufacturers and architectures offer different TEEs [12], [17]–[19], [49]–[51]. TEEs have been used to protect sensitive workloads such as data analytics on untrusted infrastructure [52], [53], confidential ML training/inference [54], [55], blockchain [56], and others [57].

We provide TEEs as a representative facilitator, as they

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

form a foundation of cHPC systems and their performance. However, this category generalizes to a broader set of hardware. Examples include roots-of-trust (RoTs) [39], which are leveraged as building blocks of TEEs to ensure secure boot and attestation measurements. Example RoTs are AMD PSP [58], Intel SEAM [17], ARM RME [59], NVIDIA On-Die RoT [12], and Graphcore CCU [50]. Modern CPUs may also provide hardware-secured, fused key storage [60] and physical unclonable functions (PUFs) [61], which enable hardware-backed identity and key derivation. Although these systems are included in facilitators, members other than TEEs are typically not critical to cHPC performance. For optimal performance, expensive facilitator operations, e.g., attestation or secure key exchange involving RoTs or PUFs, should be performed only once at the beginning of typically long HPC jobs [9], [62].

While TEE guarantees differ, their security is achieved in a similar way [39], frequently through *transparent accelerators*, which form another type in our categorization. These provide functionality without a user interface and typically operate at the system level. Because all data leaving a TEE needs to be protected, these accelerators are crucial to the performance of TEEs. For example, they encrypt all TEE memory, which is achieved through encryption engines (EEs) at the socket interconnect [63], [64] and memory controllers [17], [19]. Similarly, any traffic flowing over PCIe (e.g., between CPU and GPU TEEs) needs to be protected. Currently, it is achieved in software, leveraging expensive bounce buffers (Section IV-E). While still in development in upcoming products, such as Intel's TDXIO [17], [65] and AMD's SEV-TIO [66], PCIe IDE and TDISP aim to solve this bottleneck. Anything injected into the network must also be protected. This can be achieved by leveraging protocols such as IPsec [67] and WireGuard [68], which can be run either in the kernel employing programmable accelerators, or offloaded to the network interface cards (NICs) (Section IV-C). Finally, protections for disk storage can be added either in the kernel using DM-crypt or DM-integrity [69] or offloaded using solutions such as self-encrypting drives (SED) [70].

We also differentiate *programmable accelerators*. In contrast to transparent accelerators, these have a direct interface that specifies operations, inputs, and outputs. Well-known programmable accelerators include synchronous AES-NI [71], SHA-NI [72], and ARM cryptography extensions [73]. They can also be asynchronous, like in the case of BlueField GGAs [74] or Graphcore's AES-GCM [50]. FPGAs have their equivalent bitstream encryption engines, such as those found in Xilinx [75] or Altera's SDM [76]. All operate on the user level and accelerate basic security primitives, typically AES and SHA-based algorithms (Section IV-B). Example applications include homomorphic encryption [9], secure networking [77], differential privacy [62], and secure storage [78].

2 Security taxonomy: The main part of system descriptions is the security guarantees. This is critical as performance is typically in trade-off with more guarantees. For example, an alternative to TEEs, homomorphic encryption (HE) enables computations conducted on encrypted data without decryption [79]. Although providing strong guarantees, state-of-the-art HE reduces performance up to 10,000x due to the size of the keys, ciphertext expansions, and operation overheads [9], [79], making its applicability narrow [9], [79]–[81]. We aimed for a taxonomy that is generalizable enough to model all systems and simple enough not to require a deep understanding of security. Thus, we leveraged terms known in security research. We inspire our approach with the hardware landscape we explored (Section III-A), insights from security research [10], [82], [83], and our review of past attacks [79], [84]–[93] that can potentially compromise the whole system. Manufacturers and users can use SecPerf to also provide information about known vulnerabilities alongside guarantees of a given solution. This is beneficial, as currently, such aggregation of vulnerabilities does not exist.

	Data-in-use	Data-in-motion	Data-at-rest
Confidentiality	TDX, SEV-SNP AES-XTS	TLS ChaCha20	DM-Crypt AES-XTS
Integrity	TDX ECC fields SHA-3	WireGuard Poly1305	DM-Integrity AES-GCM
Freshness	SGX v1 Merkle tree	IPsec PSN window	DM-x Merkle tree

TABLE II: Guarantees matrix for data protection, with example systems and their primitive algorithms.

We define the taxonomy in two key areas: *data protection* and *trust*. These areas also map directly to our system taxonomy, where data protection is conducted on accelerators while trust is placed in facilitators. Table II summarizes the data protections from our categorization in a matrix that outlines corresponding example protocols and products, along with their underlying primitive algorithms. The columns correspond to the states of data. Data-in-use refers to the processing of data, data-in-motion refers to the transfer of data between entities, and data-at-rest refers to the passive storage of data that does not involve computation or data movement. The rows connect directly to potential threats:

- *Confidentiality* ensures a malicious agent cannot comprehend the data without knowing a secret key.
- *Integrity* ensures a malicious agent cannot modify the data without detection.
- *Freshness* ensures a malicious agent cannot replace data with stale valid data (similar to the ABA [94] problem).

While we show different solutions in Table II for each matrix element, some can cover more than one (e.g., TLS [95] can cover confidentiality, integrity, and freshness). A fully secure end-to-end cHPC system should encompass a technology stack that covers all elements of the security matrix. As security follows the weakest-link paradigm [96], the lack or breach of any of them may compromise the system's overall security. For example, freshness is ignored in most TEEs due to the significant performance cost of a Merkle Tree spanning memory, as apparent in the first implementation of SGX [18]. Yet, the lack of freshness indirectly led to an inexpensive attack compromising AMD SEV-SNP [93].

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

System	Guarantees (use, motion, store)				Type	Hardware	Primitives	Frontends	Benchmark	Section
	Confidentiality	Integrity	Freshness	Trust						
Encryption primitives	■■■	□□□	□□□	□		AES-NI, SIMD	AES CTR256, AES-XTS256, ChaCha20	CryptoPP, libsodium, OpenSSL	Own	IV-B
Hashing primitives	□□□	■■■	□□□	□		SHA-NI, SIMD, BF-2 GGAs	SHA-256, SHA-512, BLAKE2, BLAKE3	CryptoPP, libsodium, OpenSSL, BLAKE3, DOCA	Own DOCA Bench	IV-B
AEAD primitives	■■■	■■■	□□□	□		AES-NI, SIMD	AES-GCM, AEGIS128L, ChaCha20-Poly1305	CryptoPP, libsodium, OpenSSL	Own	IV-B
IPsec	□■□	□■□	□■□	□		AES-NI, SIMD BF-2 NIC	AES-GCM	WireGuard, OpenSwan NIC offload	UCX, OSU, Graph500 LULESH, ML proxy	IV-C
Intel MKTME	■□□	■□□	□□□	□		MKTME EE	AES-XTS, SHA-3	–		
AMD SME	■□□	□□□	□□□	□		SME EE	AES-XEX	–	MLC	IV-D
NUMA encryption	□■□	□■□	□□□	□		?	?	–		
AMD SEV/ES	■□□	□□□	□□□	■		AMD SEV/ES	–	–		
AMD SEV-SNP	■□□	◧□□	□□□	■		AMD SEV-SNP	–	–	Black Scholes, GEMM, GROMACS, LULESH, HPCG, HPL, MILC, NPB, SPEC CPU 2017, TPC-C	IV-D
Intel TDX	■□□	■□□	□□□	■		Intel TDX	–	–		
Intel SGX (classic)	■□□	■□□	■□□	■		Intel SGX	–	–		
Intel SGX (scalable)	■□□	■□□	□□□	■		Intel SGX	–	–		
Accelerator PCIe encryption	□■□	□■□	□□□	□		NVIDIA bounce buffer EE	AES-GCM	–	NVbandwidth	IV-E
NVIDIA H100 Confidential GPU	□□□	◧□□	□□□	■		NVIDIA bounce buffer EE	–	–	TorchBench	IV-E

TABLE III: Overview of systems categorized in SecPerf, with guarantees on trust and data (use, motion, and store, as in Table II, and squares, as in Table I), system types (symbols from Figure 3), hardware and primitives employed, and frontends and benchmarks that SecPerf supports. "Own" refers to our C++ benchmarking; "?" indicates no public information available.

Trust is the second pillar of the taxonomy. It describes how much tenants believe that no malicious agent operates on the cluster. Trust can be measured by the size of the trusted computing base (TCB), which represents all components that provide a secure environment. A smaller TCB reduces the attack surface and increases system assurance. cHPC hardware should operate in a zero-trust manner, which implies no trust unless a formal verification via attestation is conducted [97]. While theoretically sound, zero trust in practice only minimizes the risk of malicious agents rather than eliminating them. For example, attestation in TEEs requires confirmation from the vendor that the underlying hardware is a legal product in a given state. Thus, tenants trust that the hardware vendor will not mislead them or collude with the data center operator, who might want to steal their data. CC and TEEs are only approximations of zero-trust; nonetheless, the most practical ones the state-of-the-art offers for computing [36].

The above properties are the minimum fundamental distinctions we identified across systems and protocols that have been proven critical through attacks, while also providing helpful information, especially for non-security experts. Table III shows how both our security and hardware taxonomies can be easily used to differentiate between solutions. The shown guarantees are offered by default by the listed solutions, and additional guarantees can be introduced by combining systems (e.g., TEEs do not provide network protection by default and require additional solutions, such as IPsec, to enable them). Note, however, that the primary goal of the categorizations is not to provide security blocks that can be easily combined (e.g., Intel's MKTME cannot be combined with AMD's SEV), but rather to facilitate comparisons between systems. For example, Intel's MKTME supports cryptographic integrity protection by reusing ECC fields of ECC-capable DRAM to store a 28-bit SHA-3-based message-authentication code (MAC) for each 64B cache line [18]. In contrast, AMD's SME does not offer such a solution. This results in Intel's TDX and SGX supporting not only logical integrity isolation between tenants, the host OS, and TEEs, but also cryptographic integrity at the memory level. In comparison, AMD's SEV-SNP provides only logical integrity protection via its Reverse Map Table (RMP), whereas its SEV/ES offers none. The old version of SGX (classic) [18] also supported a Merkle Tree over the memory to provide freshness, which has been removed in the newer SGX versions. Additionally, we know that in all these TEEs, socket interconnects are encrypted via dedicated encryption engines [63], [64]; however, the exact details of the primitives used in these engines are not publicly disclosed. While GPU TEEs, such as NVIDIA's H100s, offer logical isolation between tenants by allocating the entire GPU to a single tenant, they do not protect their HBM memory at all. The next generation B100s offer such protections; however, they are not yet widely available in their CC configuration.

3 **Configuration:** defines the steps necessary to enable secure system functionality and verify it is working, gathering the information required to facilitate easy deployment. While SecPerf strives to automate configuration using scripts as much as possible, for specific steps, such as BIOS setup, this is not feasible, and a textual description of the steps is required. Alternatively, a link to external manufacturer documentation may be provided. Note that while many secure solutions require hardware (facilitators) or achieve best performance with hardware accelerators, some need minimal or no configuration (e.g., AES-NI). Our approach also generalizes to them.

B. Benchmark specification **4**

Each system type has different workload needs. For example, programmable accelerators can be unified with standard interfaces, while transparent accelerators require proxy benchmarks that indirectly measure their overheads. To address this, we expose two types of interfaces to the user. First, we use the curiously recurring template pattern (CRTP) [98] to define C++ skeletons for systems that permit the unification of their interfaces. For example, encryption accelerators have unifiable functions such as `encrypt` and `decrypt`. For

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

these cases, we include automatic input, output, and key generation routines. The only step necessary to expand such an interface is to implement the functions specified for a given primitive. Second, we define a Python interface that is more versatile, enabling the use of existing benchmarks and the implementation of new ones at a higher level. Each specification in this interface contains the code to build and invoke the benchmark. It also includes output parsers that extract metrics of interest (such as throughput, latency, queries per second, etc.) and output them to a unified CSV. The state for this interface is defined in YAML files, which specify details such as build versions, runtime parameters, and run timeouts. Leveraging this interface enables SecPerf to be efficient by using standard tools developed by the community or vendors (e.g., DocaBench [99]) without reinventing the wheel. Table III also provides an overview of the 19 currently supported benchmarks, each with multiple metrics and applications, resulting in 29 measured workload variants. Whenever possible, SecPerf exposes multiple frontends to the same accelerator (e.g., AES-NI), thereby increasing the total number of primitive implementations to 25. As shown in Section IV-B, these can have a significant impact on performance, especially across workload requirements (e.g., encryption buffer size).

C. Runtime 5

We offer our main runtime as a Docker container to enable portability across environments. We provide command-line scripts to enable easy benchmark runs. To measure performance, we provide profiler scripts for users. These include not only basic performance measurements leveraging coarse-grained wall-clock time and fine-grained timestamp counters (RDTSC), but also energy consumption and core utilization. These form the basis of understanding the resource cost, yielding the performance-adjusted energy usage (e.g., queries per watt). Additionally, we enable users to gather statistics on system calls, which incur costs in TEEs.

IV. Evaluation

We leverage SecPerf to study the following research questions:

1) How performant are programmable accelerators?
2) What is the cost of end-to-end cHPC networking?
3) What are the application overheads of CPU TEEs?
4) How costly is GPU acceleration on cHPC applications?

In our evaluation, we focus on novel insights, avoiding properties that are not critical to overall performance, such as one-off startup [100] or attestation [26].

A. Experimental setup

For evaluation, we follow the standard benchmarking practice [101] and use systems outlined in Table IV. Our virtual machines (VMs) occupied an entire socket (VM threads pinned to physical cores) and ran Ubuntu 24.04 LTS. For SGX, we used Gramine [102]. We set the frequency governor to performance.

B. Programmable accelerators

The performance of programmable accelerators is crucial to evaluate, as it serves as an upper bound for our analysis of other secure systems that leverage them. In our implementation, we focused on algorithms that either significantly outperform competitors or have been standardized and recommended by institutions such as the National Institute of Standards and Technology (NIST). We implement CryptoPP, libsodium, OpenSSL, BLAKE3 [103], [104], NVIDIA DOCA, and AEGIS128L [105] accelerator frontends in these experiments. Absent datapoints are due to the given backend not supporting sub-block operations. We focus on the latency of operation and setup (including defining all necessary structures and configuring accelerators), single-core throughput, and scalability with increased core counts across different buffer sizes. Time can be converted to cycles using our TSC clock, which runs at a constant 2 GHz.

Section	#N/#S	CPU	Memory	OS / SW	Network / Accelerator
IV-B IV-D	1/2	Intel Xeon Platinum 8580	512 GB, DDR5 4800 MT/s	Ubuntu 24.04	–
IV-B (DOCA)	1/1	NVIDIA BF-2 DPU	16 GB DDR	Ubuntu 22.04 DOCA 2.10.0	NVIDIA BF-2
IV-C (P2P)	2/1	Intel Xeon Gold 6130	196 GB, DDR4 2666 MT/s	Ubuntu 24.04 DOCA 2.10.0	100 Gbps IB CX6 NICs
IV-C (Others)	7/2	Intel Xeon Gold 6140	196 GB, DDR4 2666 MT/s	Rocky 9.4	100 Gbps IB CX5 NICs
IV-E	1/2	Intel Xeon Gold 6530	1 TiB, DDR5 5600 MT/s	Ubuntu 24.04	4xH100 SXM5 80 GB
IV-D (Compute)	1/2	AMD EPYC 7773X	1 TiB, DDR4 3200 MT/s	Ubuntu 24.04	–

TABLE IV: Details of six systems we used in evaluation, with N the number of nodes, and S the number of sockets per node.

Confidentiality: Figure 4 shows the performance of accelerator frontends for providing confidentiality to buffers of different sizes. AES implementations are accelerated using AES-NI, while ChaCha20 ones leverage SIMD vector instructions. The more software-reliant ChaCha20 is penalized in terms of throughput and latency. The fastest AES CTR achieves 30 ns of latency and 8.4 GB/s of throughput per core. While OpenSSL has the longest setup time, it performs best overall. We attribute this to its highly optimized code, which is partially written in manual assembly. The performance difference between the best and worst implementations of a given algorithm can reach 8.3x. The setup time required whenever a key or initialization vector is modified determines which algorithm is the fastest for small sizes. libsodium's ChaCha20 has the highest throughput up to 1 KiB, including setup time. Our evaluation also examines the scaling of AES CTR, which performed best. We find that the encryption scales with a factor of 0.71, reaching 357 GB/s. This throughput suffices for most applications and stems from AES CTR being embarrassingly parallel when operating above its block size. However, the resources used by synchronous encryption scale linearly, which can deteriorate application performance (Section IV-C).

Insight 1: Programmable accelerators show a large variance in performance across buffer sizes and frontends.

Recommendation 1: Verify the size of buffers used by your application to select the optimal algorithm.

Integrity: Figure 5 shows the performance of frontends that provide integrity via hashing. Apart from standardized algorithms, we also benchmark BLAKE [103], [104], designed

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

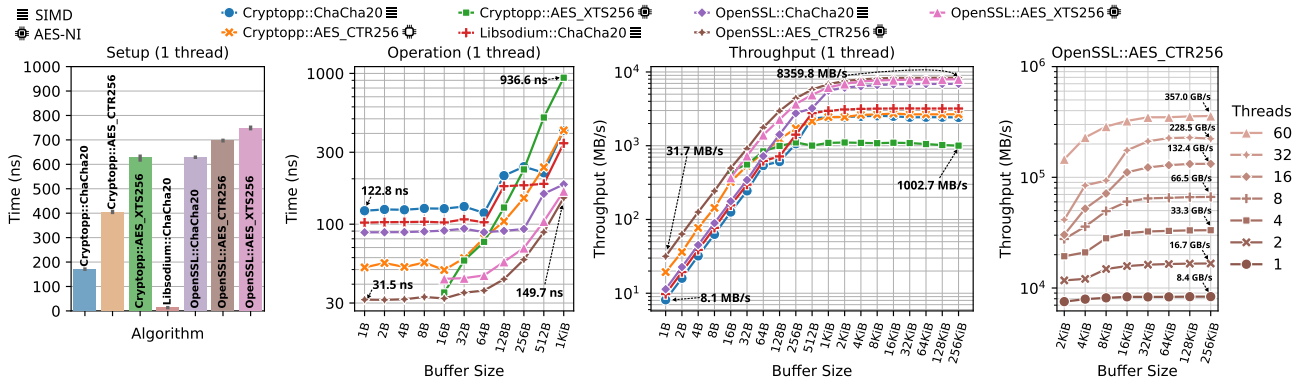

Fig. 4: Setup and operation latency, throughput, and scaling for encryption leveraging different algorithm backends.

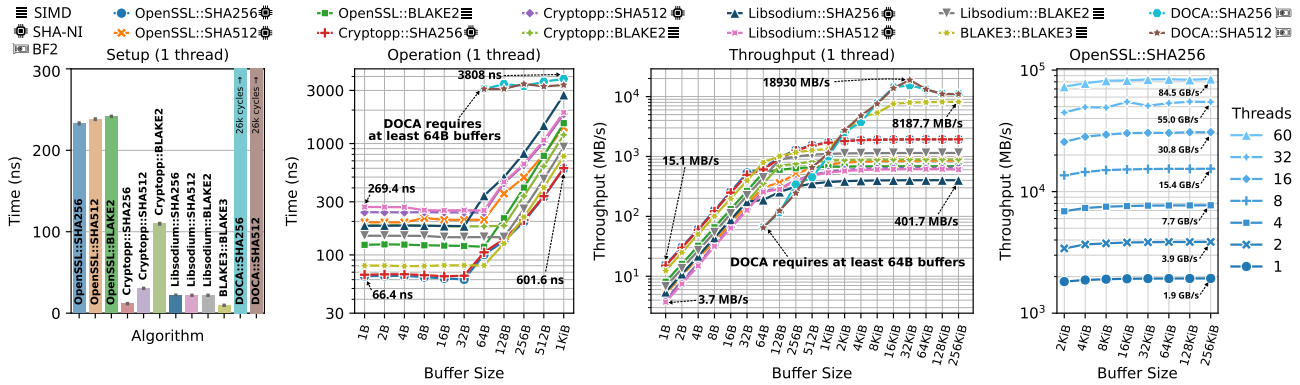

Fig. 5: Setup and operation latency, throughput, and scaling for integrity protections on buffers of different sizes.

as a fast hashing routine, and asynchronous BlueField-2's DOCA GGAs. Compared to encryption, integrity throughput and latency are considerably worse. SHA-256 implementation in OpenSSL has the lowest latency of 66 ns in operation for small buffers. DOCA's GGAs achieve the best throughput but exhibit the worst latency due to the typical microsecond-scale invocation latency of the engine (Section IV-E). GGAs also scale poorly, reaching only 11 GB/s, compared to OpenSSL's SHA-256, which achieved 84.5 GB/s. Nevertheless, the advantages of GGAs lie in different aspects. First, they are directly located on the critical network path in the NIC. Second, they are asynchronous, utilizing fewer resources on the host CPU.

BLAKE3 starts with slightly higher latency than OpenSSL, but then catches up. BLAKE3's throughput, however, reaches a considerably higher asymptotic rate at 8.1 GB/s per core. It also achieves a higher overall scaling throughput of 358 GB/s, considerably outperforming OpenSSL. BLAKE's performance advantage lies in the underlying hashing mechanism. While SHA-256 processes data sequentially, block by block, BLAKE creates a Merkle tree by chunking data into 1 KiB segments, enabling more effective parallelism. The 1 KiB buffer size, where BLAKE changes its asymptotic behaviour, exactly corresponds to the introduction of another chunk. We show scaling for OpenSSL SHA-256, the fastest standardized algorithm, achieving a throughput of 2 GB/s per core.

Insight 2: While scaling worse than host algorithms, asynchronous hashing offers a resource-efficient alternative.

Recommendation 2: For buffers above 2 KiB and if trusting the NIC is allowed, offload integrity for minimal resource usage. Otherwise, use BLAKE2/3.

AEAD: Integrity is rarely implemented in isolation. Specific algorithms have been developed to combine confidentiality and integrity, resulting in AEAD. Figure 6 shows the performance of various AEAD implementations. We focus on three algorithms in these experiments: AES-NI-accelerated AES-GCM, vectorized ChaCha20 with Poly1305, and AEGIS128L, the winner of a recent CAESAR competition [106]. Combining two algorithms sequentially, like in ChaCha20-Poly1305, incurs a latency penalty that rarely makes sense. AEGIS excels in terms of latency (50 ns) and throughput (16.7 GB/s). While such throughput per core is remarkable, it results from AEGIS being designed from the ground up to achieve high performance on modern CPUs [105]. AEGIS uses AES-NI, SIMD, and linear state updates rather than polynomial multiplication over GF(2^{128}) as in AES-GCM [105], thereby improving its efficiency on modern processors. AEGIS reaches 723 GB/s with 60 cores and a 256 KiB buffer (fitting in the L2 of our

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

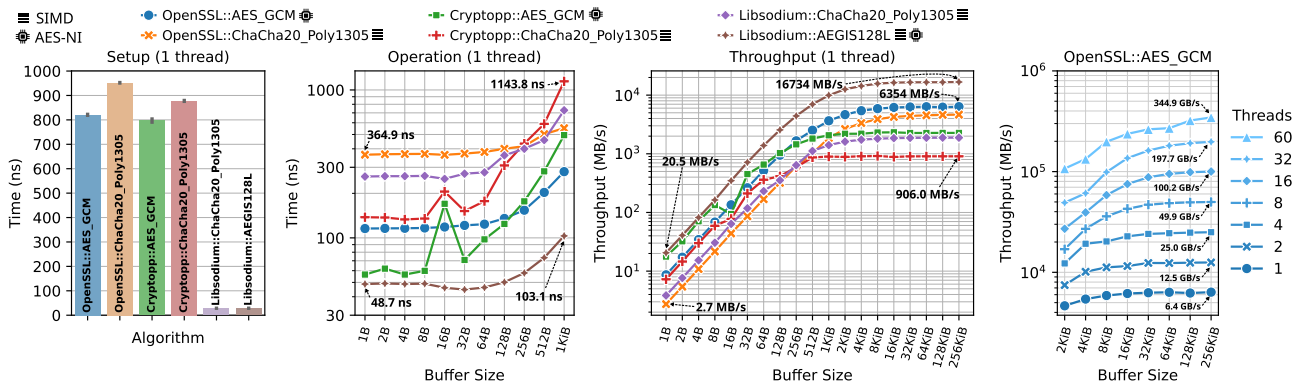

Fig. 6: Setup and operation latency, throughput, and scaling for integrity and confidentiality (AEAD) protections on buffers.

machine). We verified that this throughput is below the L2 throughput. Although these numbers are impressive, care must be taken in practice as AEGIS has not yet been standardized.

Insight 3: Modern primitives reach 16.7 GB/s per core, suitable for demanding workloads (e.g., 400 Gbps networking).

Recommendation 3: If lack of standardization is not an issue, use AEGIS. Otherwise, use OpenSSL's AES-GCM.

C. Networking

Evaluating transparent accelerators, such as network protections, is more challenging, as they lack a direct interface and require proxy benchmarking, such as UCX [107], OSU [108], and distributed applications [109]–[111]. For network protection, we initially selected the TLS, IPsec, and WireGuard protocols, which provide all three data security guarantees without requiring application modifications. However, we found that TLS presents challenges in the context of cHPC, as it securely binds TCP/IP *ports* of two endpoints. Predetermining which ports are selected from ephemeral Linux pools is non-trivial. In contrast, WireGuard and IPsec create secure tunnels between IP *addresses*, enabling easy deployment.

We begin by evaluating the performance of common UCX primitives in a point-to-point, 100 Gbps IP-over-InfiniBand system. We use it as a reference for the performance of primitives leveraged in, e.g., Message Passing Interface (MPI) implementations. Figure 7 presents the results showing eight systems. Remote Direct Memory Access (RDMA) [112]–[115] and TCP are baselines that are not protected. IPsec and WireGuard are implemented in the kernel. IPsec (pcrypt) is leveraging the `pcrypt` kernel module, the only existing way to parallelize IPsec. We also include results leveraging NIC cryptographic offloading (cryptographic operations) and full offloading (cryptographic operations with packet processing).

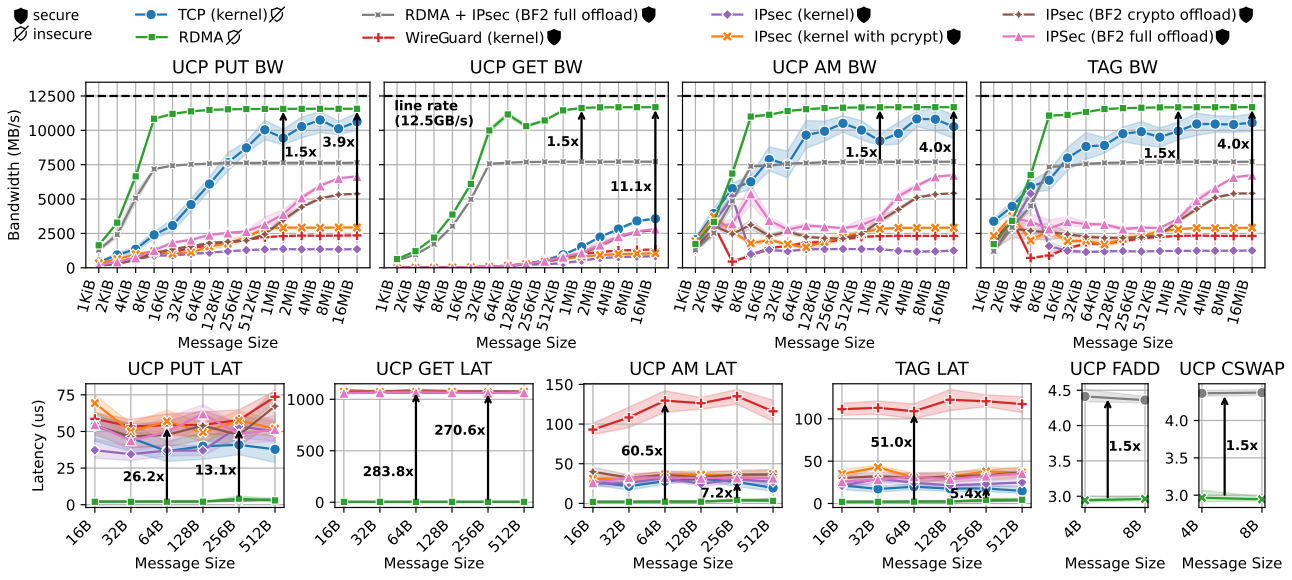

Fig. 7: Performance of UCX benchmarks with different protection methods on a 100 Gbps point-to-point IPoIB link.

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

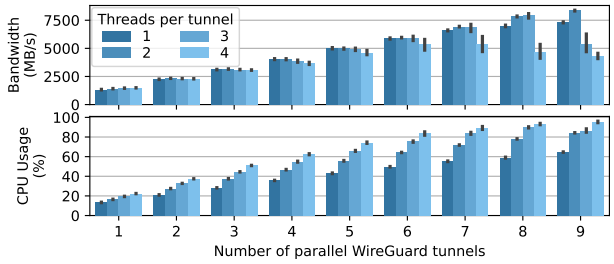

Fig. 8: UCP PUT BW vs. # of WireGuard tunnels (12.5 GB/s link).

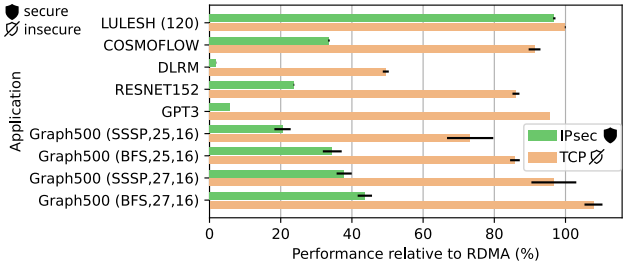

Fig. 9: End-to-end IPsec overhead on different apps.

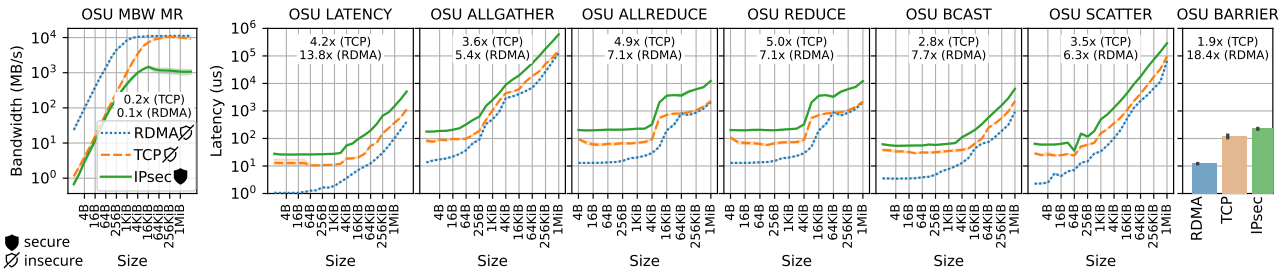

Fig. 10: OSU microbenchmarks: IPsec over TCP vs. RDMA.

We observe that the throughput of a raw kernel AES-GCM IPsec is 7.4x worse than the performance of RDMA and 7x worse than raw TCP in the put bandwidth test. Such overheads are repeated throughout the bandwidth tests. We investigated the cause of such poor performance and identified the in-kernel IPsec design as the primary factor. Modern kernels assign each IPsec tunnel only one thread, creating a point of contention. While `pcrypt` improves performance to 2.7 GB/s, it fails to address the main issue, as it only parallelizes the cryptographic part of the operations. Notably, after evaluating AES-GCM AEAD performance using SecPerf, we observed that it achieves 3.6 GB/s per core, considerably higher than the 1.3 GB/s we measured for UCP PUT BW. This suggests that contention is the primary source of overhead, a software design limitation rather than a CPU hardware bottleneck. We also found that the alternative WireGuard frontend (ChaCha20-Poly1305) did not perform considerably better.

Insight 4: Raw IPsec performance is limited by the single-threaded in-kernel implementation.

An optimization that significantly improved performance was the use of CX6 NIC offloading. We found that offloading cryptographic operations increased throughput to 5.4 GB/s and full offload to 6.7 GB/s. Changing the transport and enabling IPsec on RDMA further increased performance to 7.7 GB/s, reaching 70% of the maximum RDMA bandwidth. We observed that performance plateaued at this point due to saturation of the NIC cryptographic engine [99]. Despite these improvements, NIC offloading presents a fundamental challenge detected by our security taxonomy: the lack of trust in the NIC and protection at the PCIe link layer. While IDE and TDISP are in development, no secure method currently exists to integrate NICs into the TCB. Even as these technologies reach the market, ensuring availability across all heterogeneous cHPC nodes remains a significant challenge.

Insight 5: While increasing the performance to 7.7 GB/s, IPsec offloaded to NICs currently does not provide trust.

Recommendation 4: When lack of trust in the NIC is allowed, offload network protections, preferentially with RDMA. Otherwise, run IPsec with the pcrypt module.

We evaluated UCP PUT BW while scaling the number of tunnels between two nodes, as a potential optimization that also simulates multi-tenant traffic. Figure 8 confirms our hypothesis that single-thread contention harms performance. We observe an almost linear throughput scaling with the number of tunnels until computational resources become the bottleneck at about six parallel tunnels. While we achieved throughput exceeding 8 GB/s, beating even offloading approaches, it came at a significant resource cost. For example, with 7 cores and 3 driving threads per core, we used 80% of the CPU: 58% for communication and 22% for cryptographic operations.

As scaling with the number of tunnels increases linearly, these findings suggest that IPsec may be better suited for collectives in which multiple ranks communicate in parallel over different IP addresses (and thus different tunnels). To verify this, we run OSU microbenchmarks on a 252-rank setup with a full point-to-point mesh of tunnels between the nodes' IP addresses. As shown in Figure 10, the results indicate that IPsec performance overheads remain comparable to or worse than those of point-to-point communication, with some collectives, such as scatter and gather, performing better than others. Despite attempts to optimize collective performance, such as Allreduce, using different algorithms, we observed no substantial improvements. We found that the aforementioned CPU communication and security contention contribute to the poor performance, especially at high rank counts per node. These findings show that hybrid systems combining MPI

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

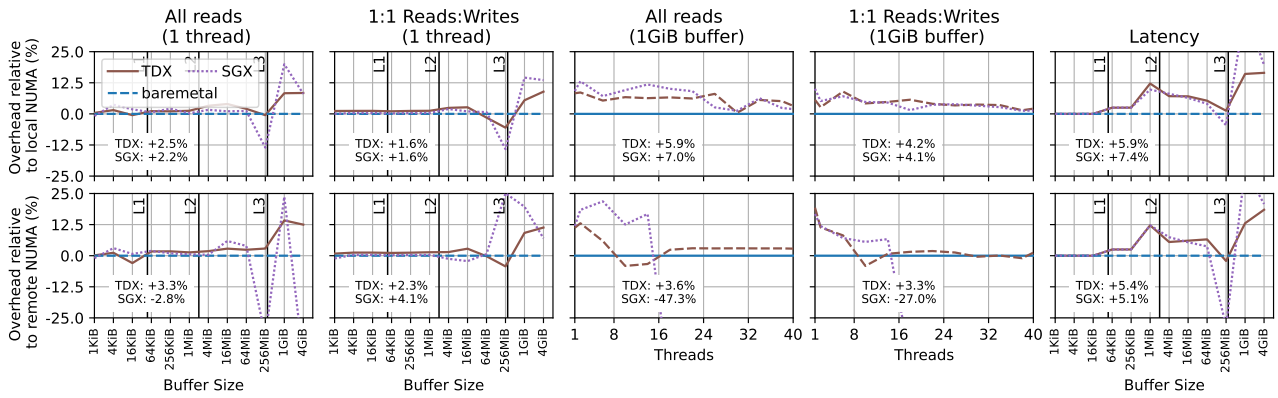

Fig. 11: Memory access performance with negative remote socket results for SGX caused by not adhering to NUMA allocations.

with other frameworks, such as OpenMP, can achieve better performance by minimizing the number of communicating processes. Another important observation is that even if we optimize the kernel's IPsec, performance will still be limited to TCP levels that are considerably worse than RDMA.

Insight 6: Although increasing the number of parallel IPsec tunnels can achieve 8 GB/s, this comes at a considerable resource cost, thereby impacting collective performance.

Recommendation 5: If your application is not saturating the CPU, leverage multiple IPsec tunnels, each with one thread.

While microbenchmarks provide valuable insights into communication overheads, they do not fully reflect an application's overall performance. To gain a more comprehensive understanding, we evaluated a set of distributed applications [109], [110], [116] shown in Figure 9. Unlike in the point-to-point and collective benchmarks, we observed that TCP delivers satisfactory performance for all applications except DLRM training. However, IPsec's protection introduced considerable overhead, reducing application performance to only 35% of RDMA, except for LULESH [109], which has a small communication-to-computation ratio. In the worst case, IPsec introduces 98% overhead for DLRM training. DLRM training is latency-sensitive because it involves small compute kernels and peer-to-peer communication. Latency introduced by TCP already reduces performance by half, and IPsec adds an additional order of magnitude of latency overhead (Figure 10).

Insight 7: cHPC networking is a major source of overhead with performance reduced by an average of 35% (2-98%).

D. CPU TEEs

Memory and NUMA: We also examine the performance of transparent accelerators in memory and socket interconnect, which are significant sources of overhead for TEEs. Figure 11 shows the throughput and latency of memory transfers leveraging Intel's Memory Latency Checker (MLC) [117]. We tested two patterns, all reads, and 1:1 reads:writes, across different buffer sizes and thread counts. Starting with the former, we observe that performance drops remain small below the L3 cache. This is expected as the cache layer remains unencrypted and is only logically protected at the address level. As we reach L3, TDX overheads jump considerably to 8.4-8.9%. A similar trend is visible on the remote socket with a larger overhead, adding 2.4-4%, to a total of up to 12.4%. The average overheads across all buffer sizes remain within 2-4%. Noise in TDX can be attributed to VM-based TEEs from Intel and AMD that do not use pre-allocated hugepages and default to 2MiB transparent pages. For SGX, we observe noise and overheads that can reach 25%, which is considerably higher than for TDX. The noise and negative overheads for SGX are caused by failing to adhere to the provided NUMA constraints and by using Enclave Page Cache (EPC). EPC constitutes memory pages allocated during boot and used as a cache for the TEE, which is crucial for SGX performance. In our experiments, we observed that SGX performance dropped by more than 99% with a small EPC. However, EPC allows SGX to ignore our NUMA constraints entirely and allocate memory on any node. That means performance can be better or worse than the baseline, depending on whether SGX allocates thread memory on a remote or local node, regardless of our imposed constraints. SGX shows similar noise on remote nodes.

We then selected 1 GiB as a representative buffer size at which the encryption engine is activated and scaled the accesses across different numbers of threads. We observed that as the raw throughput plateaus, the overheads decrease from 10-12% to 2-3% for both SGX and TDX. This occurs when the encryption engines become saturated. Surprisingly, the overheads for remote socket accesses drop to almost negligible levels, suggesting that once the link between sockets saturates, the encryption engines no longer become a bottleneck. Finally, latency remains sensitive and exhibits larger overheads reaching 16.5%. As these cannot be hidden like in the bandwidth case, such behavior is expected.

Insight 8: Memory overheads depend on the buffer size and thread count, on average reaching less than 10%, with socket interconnect encryption adding up to 4% overhead.

Recommendation 6: Minimize data movement between caches, and main memory/sockets (e.g., using prefetchers).

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

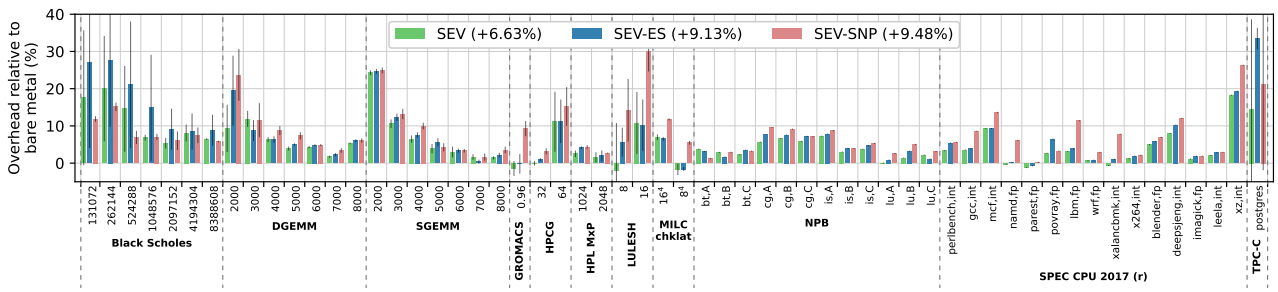

Fig. 12: End-to-end application performance across the AMD TEEs displaying consistent overheads of less than 30%. For clarity, we show the subset of interesting results but indicate average overheads across the entire measured dataset.

We also investigated the energy usage to understand the resource cost. We found that as the buffer size scales, the average power usage for TDX remains constant (local socket) and increases up to 1.99% (remote socket). SGX experiences similar overheads. TDX measured the same power usage for thread scaling, while SGX increased the power usage by 11.6-11.9% on average. As overheads grew with the number of threads, we associate this with control-path overheads within SGX and Gramine. For the latency experiments, we noticed 5-6% (TDX) and 3.5-7.16% (SGX) higher usage. These higher numbers indicate the overhead of running the control path for TDX and SGX, as power usage for latency is not dominated by memory transfers, unlike in batch and thread scaling. The above results do not account for the lost performance. The average performance-adjusted energy overheads across all experiments are 6.18% (TDX local), 4.16% (TDX remote), 3.5% (SGX local), and -0.61% (SGX remote). Maximum performance-adjusted overheads reach 9.85% (TDX local), 13.27% (TDX remote), 24.57% (SGX local), and 23.39% (SGX remote). SGX's energy usage is also affected by its startup time, which exceeds 2 minutes for our 256 GB enclave. The within-noise negative performance-adjusted average energy overhead for SGX remote is caused by the aforementioned lack of NUMA support, which enables better performance than the baseline. However, the maximum is still positive, as expected.

Insight 9: Memory access resource costs are on average less than 12% (raw), and less than 7% (performance-adjusted).

Compute overheads: To evaluate the end-to-end performance of CPU TEEs, we assembled a diverse suite of relevant applications and benchmarks spanning multiple domains. Our selection includes finance-oriented workloads such as the Black-Scholes benchmark [118], scientific computing applications such as GROMACS [119], HPCG [120], HPL-MXP [121], GEMM [122], and MILC su3_rmd [123], [124], database TPC-C [125] running on PostgreSQL, as well as industry-standard benchmarks like the NAS Parallel Benchmarks (NPB) [126] and SPEC CPU 2017 [127]. This comprehensive approach ensures a rigorous and realistic evaluation.

Figure 12 shows overheads within AMD's TEEs. We observe overheads below 30%. SEV-SNP reaches the highest average overhead of 9.48%, SEV-ES follows with 9.13%, and SEV with 6.63%. As the guarantees given by each system increase, such incremental overheads are to be expected. The noise visible in memory measurements also manifests in application performance. The noise and overheads fall as the workload becomes more compute-bound. For example, in Black Scholes, GEMM, and NPB, the noise can reach 20% for smaller inputs and decline to 5% as the sizes increase. The overhead follows a similar trajectory, starting at 20% and decreasing to less than 10%. Because the computation itself has zero overhead in TEEs, with costs arising only from data movement and system interaction (system calls), a focus on compute results in lower overhead. This also implies that computationally intensive applications, such as those in HPC, will perform better, and that typical performance optimizations, such as minimizing data movement [128] to saturate compute, are even more applicable for cHPC. Energy-wise, we found that these systems average between 20-24% overhead, with the majority due to virtualization rather than security, which we attribute to the older AMD CPUs used.

Insight 10: CPU TEE overheads are below 30% with average of 10%, and drop for compute-bound workloads, making traditional optimizations even more important in cHPC.

Recommendation 7: Increase arithmetic intensity and optimize data locality to keep execution compute-bound.

E. GPU TEEs

We evaluated GPU performance within GPU TEEs across three key aspects: data movement, control path, and end-to-end performance on real-world applications. This comprehensive approach provided us with a holistic view of GPU TEE performance across microbenchmarks and representative workloads.

Memory transfers and kernel invocations: To understand the GPU TEE data movement overheads due to the bounce-buffer transparent accelerator, we utilized nvbandwidth [129] and analyzed transfers between the host and device. To determine the penalty on the control path, we measured kernel invocation costs through launching empty kernels in loops and capturing CPU time after device synchronization. Figure 13 shows how the VM performance is nearly identical to bare metal across all `memcpy` operations (host-to-device, device-to-host, bidirectional) and host device latency, with differences

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

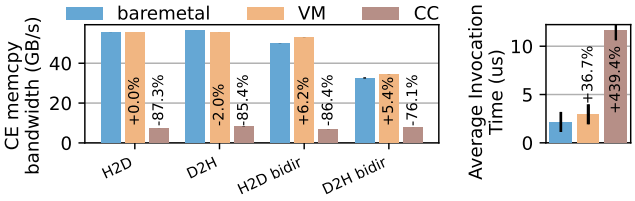

Fig. 13: Control and memory operations on the confidential GPU with copy engine (CE) and for uni- and bidirectional transfers from host to device (H2D) and device to host (D2H).

ranging from 6.2% to -2.0%. However, CC suffers from severely degraded memory bandwidth, reduced by 76.1% to 87.3% compared to bare metal. This suggests that security is the primary source of overhead, rather than virtualization. The bounce buffer explains the poor performance of `memcpy`, since copying and encrypting data rather than sending it directly to the GPU is expensive. When invoking kernels, the VM incurs moderate overhead (36.7%) compared to bare metal. TEEs exhibit a considerably elevated invocation time, which is more than five times slower than bare metal (439.4%).

Insight 11: Confidential GPUs suffer from bandwidth overheads of over 80% and invocation latency of over 400%.

Recommendation 8: Minimize the number of kernel invocations and data transfers for optimal cGPU performance.

End-to-end overheads: To understand the end-to-end cost of confidential GPUs, we evaluated them on real-world applications. We employed the PyTorch benchmark suite TorchBench [130], which features diverse models, including GNNs, CNNs such as AlexNet and ResNet-152, transformers like BERT and GPT-2, and specialized models like DLRM and DRQ. Figure 14 presents the performance of models in both training (TRAIN) and inference (EVAL) modes. While the impact of virtualization (VM) is again minimal, the TEE exhibits significant performance degradation. In the EVAL mode, overheads range from 5.5% for AlexNet up to 79.5% for ResNet152 and 100.8% for TTS angular. In TRAIN mode, the CC overhead is often lower, ranging from a single-digit change (e.g., 3.9% for AlexNet, 9.1% for DRQ) to 86.0% in DCGAN and 84.9% in TTS Angular. The substantial performance degradation in the TEE can be primarily attributed to two factors revealed by the microbenchmarks: reduced memory

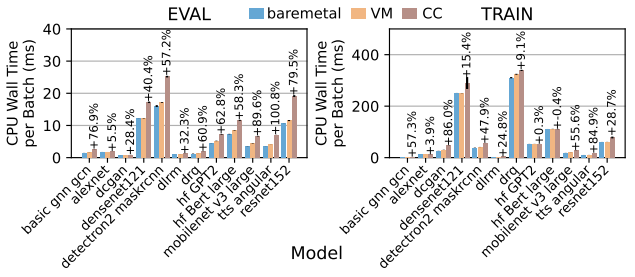

Fig. 14: Confidential GPU performance on modern models.

bandwidth due to the bounce buffer, which severely impacts models that require frequent or large data transfers between the host CPU and the accelerator; and high kernel invocation time caused by secure control path, which penalizes workloads involving frequent kernel launches or other interactions with the device driver. These explain the higher relative overhead in shorter EVAL tasks compared to longer TRAIN tasks.

Insight 12: Bandwidth and control path overheads result in final application performance degradation of 0.3-100%, depending on data transfer and number of kernel invocations.

V. Use cases

System design: a major space where SecPerf can be leveraged is in designing secure systems. For example, when building a secure storage pipeline with a Merkle tree [78], one needs to select an underlying hashing algorithm, for which SecPerf provides a ready suite.

Bottleneck detection: by porting a hardware solution to SecPerf, it is tested through multiple benchmarks, potentially displaying existing limitations. For example, in our study, we found that networking and IPsec had high costs.

Performance estimation and decision making: by providing performance for major systems, SecPerf enables rough estimation of performance. For example, our CPU compute-heavy results align well with previous results on similar workloads [36]. Such a system comparison enables informed, early decision-making without implementing multiple versions of the same application or testing different hardware types.

VI. Conclusion

We introduced SecPerf, a first-of-a-kind benchmarking platform for secure systems. SecPerf is expandable, comprehensive, and generalizable through its hardware and security taxonomies. Our framework enables practitioners and designers to compare the cost and guarantees provided by different solutions. We support more than 30 relevant cHPC workload variants within our implementation. We evaluated SecPerf on six systems, showing insights into the state-of-the-art performance of primitives, offloading, networking, CPU TEE compute, and confidential GPUs. These offer future venues for research and development, while revealing the overheads. We believe SecPerf lays the cornerstone for the broader deployment and development of cHPC solutions.

Acknowledgments

This research was conducted as part of the "UrbanTwin: An urban digital twin for climate action: Assessing policies and solutions for energy, water and infrastructure" project, funded by the ETH-Domain Joint Initiative program in the Strategic Area Energy, Climate and Sustainable Environment, with additional support from Intel Corporation. We thank the Swiss National Supercomputing Center (CSCS), Intel, and E4 Computer Engineering for providing hardware resources, and Madlen Koblinger for assisting with figure design. Mikhail Khalilov is supported by a CAFAmerica grant.

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] M. Sallam, "ChatGPT Utility in Healthcare Education, Research, and Practice: Systematic Review on the Promising Perspectives and Valid Concerns," *Healthcare*, vol. 11, no. 6, p. 887, Jan. 2023.

[2] S. Wu, O. Irsoy, S. Lu, V. Dabravolski, M. Dredze, S. Gehrmann, P. Kambadur, D. Rosenberg, and G. Mann, "BloombergGPT: A Large Language Model for Finance," Dec. 2023.

[3] J. Cui, Z. Li, Y. Yan, B. Chen, and L. Yuan, "ChatLaw: Open-Source Legal Large Language Model with Integrated External Knowledge Bases," Jun. 2023.

[4] T. Kocmi and C. Federmann, "Large Language Models Are State-of-the-Art Evaluators of Translation Quality," May 2023.

[5] O. Sharir, B. Peleg, and Y. Shoham, "The Cost of Training NLP Models: A Concise Overview," Apr. 2020.

[6] W. G. Voss, "European Union Data Privacy Law Reform: General Data Protection Regulation, Privacy Shield, and the Right to Delisting," Rochester, NY, Jan. 2017.

[7] E. Illman and P. Temple, "California Consumer Privacy Act: What Companies Need to Know," *The Business Lawyer*, vol. 75, no. 1, pp. 1637–1646, 2019.

[8] A. Goldfarb and C. Tucker, "Shifts in Privacy Concerns," *American Economic Review*, vol. 102, no. 3, pp. 349–353, May 2012.

[9] M. Chrapek, M. Khalilov, and T. Hoefler, "HEAR: Homomorphically Encrypted Allreduce," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '23. New York, NY, USA: Association for Computing Machinery, Nov. 2023, pp. 1–17.

[10] D. P. Mulligan, G. Petri, N. Spinale, G. Stockwell, and H. J. M. Vincent, "Confidential Computing—a brave new world," in *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*, Sep. 2021, pp. 132–138.

[11] M. Russinovich, "Confidential Computing: Elevating Cloud Security and Privacy: Working toward a more secure and innovative future," *Queue*, vol. 21, no. 4, pp. Pages 10:44–Pages 10:48, Sep. 2023.

[12] G. Dhanuskodi, S. Guha, V. Krishnan, A. Manjunatha, M. O'Connor, R. Nertney, and P. Rogers, "Creating the First Confidential GPUs: The team at NVIDIA brings confidentiality and integrity to user code and data for accelerated computing." *Queue*, vol. 21, no. 4, pp. Pages 40:68–Pages 40:93, Sep. 2023.

[13] PCI-SIG, "TEE device interface security protocol (TDISP)," Aug. 2022.

[14] PCI-SIG (PWG), "Integrity and data encryption (IDE) ECN," Dec. 2020.

[15] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang, G.-Y. Wei, and C.-J. Wu, "MLPerf: An Industry Standard Benchmark Suite for Machine Learning Performance," *IEEE Micro*, vol. 40, no. 2, pp. 8–16, Mar. 2020.

[16] M. Copik, G. Kwasniewski, M. Besta, M. Podstawski, and T. Hoefler, "SeBS: A serverless benchmark suite for function-as-a-service computing," in *Proceedings of the 22nd International Middleware Conference*, ser. Middleware '21. New York, NY, USA: Association for Computing Machinery, Oct. 2021, pp. 64–78.

[17] P.-C. Cheng, W. Ozga, E. Valdez, S. Ahmed, Z. Gu, H. Jamjoom, H. Franke, and J. Bottomley, "Intel TDX Demystified: A Top-Down Approach," *ACM Computing Surveys*, Mar. 2024.

[18] V. Costan and S. Devadas, "Intel SGX Explained," 2016.

[19] D. Kaplan, "AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More," *White paper*, 2020.

[20] R. Krahn, D. Dragoti, F. Gregor, D. L. Quoc, V. Schiavoni, P. Felber, C. Souza, A. Brito, and C. Fetzer, "TEEMon: A continuous performance monitoring framework for TEEs," in *Proceedings of the 21st International Middleware Conference*, ser. Middleware '20. New York, NY, USA: Association for Computing Machinery, Dec. 2020, pp. 178–192.

[21] K. Maliszewski, T. Dietzel, J.-A. Quiané-Ruiz, and V. Markl, "TeeBench: Seamless Benchmarking in Trusted Execution Environments," in *Companion of the 2023 International Conference on Management of Data*, ser. SIGMOD '23. New York, NY, USA: Association for Computing Machinery, Jun. 2023, pp. 163–166.

[22] T. Dinh Ngoc, B. Bui, S. Bitchebe, A. Tchana, V. Schiavoni, P. Felber, and D. Hagimont, "Everything You Should Know About Intel SGX Performance on Virtualized Systems," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 1, pp. 5:1–5:21, Mar. 2019.

[23] J. Amacher and V. Schiavoni, "On the Performance of ARM TrustZone: (Practical Experience Report)," in *Distributed Applications and Interoperable Systems: 19th IFIP WG 6.1 International Conference, DAIS 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17–21, 2019, Proceedings*. Berlin, Heidelberg: Springer-Verlag, Jun. 2019, pp. 133–151.

[24] K. Vaswani, S. Volos, C. Fournet, A. N. Diaz, K. Gordon, B. Vembu, S. Webster, D. Chisnall, S. Kulkarni, G. Cunningham, R. Osborne, and D. Wilkinson, "Confidential Computing within an AI Accelerator," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 501–518.

[25] A. Akram, A. Giannakou, V. Akella, J. Lowe-Power, and S. Peisert, "Performance Analysis of Scientific Computing Workloads on General Purpose TEEs," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2021, pp. 1066–1076.

[26] M. Misono, D. Stavrakakis, N. Santos, and P. Bhatotia, "Confidential VMs Explained: An Empirical Analysis of AMD SEV-SNP and Intel TDX," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 8, no. 3, pp. 36:1–36:42, Dec. 2024.

[27] C. Göttel, R. Pires, I. Rocha, S. Vaucher, P. Felber, M. Pasin, and V. Schiavoni, "Security, Performance and Energy Trade-Offs of Hardware-Assisted Memory Protection Mechanisms," in *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, Oct. 2018, pp. 133–142.

[28] K. Suzaki, K. Nakajima, T. Oi, and A. Tsukamoto, "TS-Perf: General Performance Measurement of Trusted Execution Environment and Rich Execution Environment on Intel SGX, Arm TrustZone, and RISC-V Keystone," *IEEE Access*, vol. 9, pp. 133 520–133 530, 2021.

[29] L. Coppolino, S. D'Antonio, and G. Mazzeo, "A Comprehensive Survey of Hardware-assisted Security: From the Edge to the Cloud," *Internet of Things*, vol. 6, p. 100055, Apr. 2019.

[30] S. Mofrad, F. Zhang, S. Lu, and W. Shi, "A comparison study of intel SGX and AMD memory encryption technology," in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '18. New York, NY, USA: Association for Computing Machinery, Jun. 2018, pp. 1–8.

[31] B. Dong and Q. Wang, "Evaluating the Performance of the DeepSeek Model in Confidential Computing Environment," Feb. 2025.

[32] A. Mohan, M. Ye, H. Franke, M. Srivatsa, Z. Liu, and N. M. Gonzalez, "Securing AI Inference in the Cloud: Is CPU-GPU Confidential Computing Ready?" in *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*. IEEE Computer Society, Jul. 2024, pp. 164–175.

[33] J. Zhu, H. Yin, P. Deng, A. Almeida, and S. Zhou, "Confidential Computing on NVIDIA Hopper GPUs: A Performance Benchmark Study," Nov. 2024.

[34] A. M. Ibarra, J. J. Stephen, A. G. Vidal, K. R. Jayaram, and A. F. S. Gómez, "Performance of Confidential Computing GPUs," May 2025.

[35] Y. Yang, M. Sonji, and A. Jog, "Dissecting Performance Overheads of Confidential Computing on GPU-based Systems," in *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, May 2025, pp. 1–16.

[36] M. Chrapek, M. Copik, E. Mettaz, and T. Hoefler, "Confidential LLM Inference: Performance and Cost Across CPU and GPU TEEs," in *2025 IEEE International Symposium on Workload Characterization (IISWC)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2025, pp. 84–98. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IISWC66894.2025.00017

[37] Y. Tan and Z. Mi, "Performance Analysis and Optimization of Nvidia H100 Confidential Computing for AI Workloads," in *2024 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, Oct. 2024, pp. 1426–1432.

[38] L. Coppolino, S. D'Antonio, G. Mazzeo, and L. Romano, "An experimental evaluation of TEE technology: Benchmarking transparent approaches based on SGX, SEV, and TDX," *Computers & Security*, vol. 154, p. 104457, Jul. 2025.

[39] M. Schneider, R. J. Masti, S. Shinde, S. Capkun, and R. Perez, "SoK: Hardware-supported Trusted Execution Environments," May 2022.

[40] A. Muñoz, R. Ríos, R. Román, and J. López, "A survey on the (in)security of trusted execution environments," *Computers & Security*, vol. 129, p. 103180, Jun. 2023.

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

[41] P. Jauernig, A.-R. Sadeghi, and E. Stapf, "Trusted Execution Environments: Properties, Applications, and Challenges," *IEEE Security & Privacy*, vol. 18, no. 2, pp. 56–60, Mar. 2020.

[42] X. Li, B. Zhao, G. Yang, T. Xiang, J. Weng, and R. H. Deng, "A Survey of Secure Computation Using Trusted Execution Environments," Feb. 2023.

[43] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted Execution Environment: What It is, and What It is Not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, Aug. 2015, pp. 57–64.

[44] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan, "Remote ATtestation procedureS (RATS) Architecture," Internet Engineering Task Force, Request for Comments RFC 9334, Jan. 2023.

[45] R. Walther, C. Weinhold, and M. Roitzsch, "RATLS: Integrating Transport Layer Security with Remote Attestation," in *Applied Cryptography and Network Security Workshops: ACNS 2022 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, SiMLA, Rome, Italy, June 20–23, 2022, Proceedings*. Berlin, Heidelberg: Springer-Verlag, Jun. 2022, pp. 361–379.

[46] Intel Corporation, "Intel SGX data center attestation primitives (intel SGX DCAP): Orientation guide," n.d.

[47] Advanced Micro Devices, "Versioned chip endorsement key (VCEK) certificate and KDS interface specification," Jan. 2025.

[48] NVIDIA Corporation, "NVIDIA device attestation and CoRIM-based reference measurement sharing v2.0," Jan. 2025.

[49] H. Huang, F. Zhang, S. Yan, T. Wei, and Z. He, "SoK: A Comparison Study of Arm TrustZone and CCA," in *2024 International Symposium on Secure and Private Execution Environment Design (SEED)*, May 2024, pp. 107–118.

[50] K. Vaswani, S. Volos, C. Fournet, A. N. Diaz, K. Gordon, B. Vembu, S. Webster, D. Chisnall, S. Kulkarni, G. Cunningham, R. Osborne, and D. Wilkinson, "Confidential Machine Learning within Graphcore IPUs," May 2022.

[51] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20. New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 1–16.

[52] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "Vc3: Trustworthy data analytics in the cloud using sgx," in *2015 IEEE symposium on security and privacy*. IEEE, 2015, pp. 38–54.

[53] F. Shaon, M. Kantarcioglu, Z. Lin, and L. Khan, "Sgx-bigmatrix: A practical encrypted data analytic framework with trusted processors," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1211–1228. [Online]. Available: https://doi.org/10.1145/3133956.3134095

[54] F. Tramèr and D. Boneh, "Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware," Feb. 2019.

[55] T. Hunt, Z. Jia, V. Miller, A. Szekely, Y. Hu, C. J. Rossbach, and E. Witchel, "Telekine: secure computing with cloud gpus," in *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation*, ser. NSDI'20. USA: USENIX Association, 2020, p. 817–834.

[56] C. Cai, L. Xu, A. Zhou, and C. Wang, "Toward a secure, rich, and fair query service for light clients on public blockchains," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 3640–3655, 2022.

[57] A. Paju, M. O. Javed, J. Nurmi, J. Savimäki, B. McGillion, and B. B. Brumley, "Sok: A systematic review of tee usage for developing trusted applications," in *Proceedings of the 18th International Conference on Availability, Reliability and Security*, ser. ARES '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3600160.3600169

[58] R. Buhren, C. Werling, and J.-P. Seifert, "Insecure Until Proven Updated: Analyzing AMD SEV's Remote Attestation," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 1087–1099.

[59] A. C. J. Fox, G. Stockwell, S. Xiong, H. Becker, D. P. Mulligan, G. Petri, and N. Chong, "A Verification Methodology for the Arm® Confidential Computing Architecture: From a Secure Specification to Safe Implementations," *Proc. ACM Program. Lang.*, vol. 7, no. OOPSLA1, pp. 88:376–88:405, Apr. 2023.

[60] Intel Corporation, "Intel key locker specification," Intel Corporation, Tech. Rep. 343965-001US, Sep. 2020.

[61] Y. Gao, S. F. Al-Sarawi, and D. Abbott, "Physical unclonable functions," *Nature Electronics*, vol. 3, no. 2, pp. 81–91, Feb. 2020.

[62] H. Qi, W. Chen, C. Wang, and X. Lu, "Dpar: High-performance, secure, and scalable differential privacy-based allreduce," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 1917–1934. [Online]. Available: https://doi.org/10.1145/3712285.3759828

[63] S. Johnson, R. Makaram, A. Santoni, and V. Scarlata, "Supporting intel® SGX on multi-socket platforms," Intel Corporation, Tech. Rep. 843058, Dec. 2024.

[64] "5th gen AMD EPYC™ processor architecture," AMD, Tech. Rep., Mar. 2025.

[65] Intel Corporation, "Software enabling for intel® TDX in support of TEE-i/O," Intel Corporation, Tech. Rep. Revision 1.00, Sep. 2022.

[66] D. Kaplan, "Hardware VM Isolation in the Cloud: Enabling confidential computing with AMD SEV-SNP technology," *Queue*, vol. 21, no. 4, pp. Pages 20:49–Pages 20:67, Sep. 2023.

[67] N. Ferguson and B. Schneier, "A cryptographic evaluation of IPsec," 1999.

[68] "WireGuard: Next Generation Kernel Network Tunnel."

[69] A. Chakraborti, B. Jain, J. Kasiak, T. Zhang, D. Porter, and R. Sion, "Dm-x: Protecting Volume-level Integrity for Cloud Volumes and Local Block Devices," in *Proceedings of the 8th Asia-Pacific Workshop on Systems*, ser. APSys '17. New York, NY, USA: Association for Computing Machinery, Sep. 2017, pp. 1–7.

[70] C. Meijer and B. van Gastel, "Self-Encrypting Deception: Weaknesses in the Encryption of Solid State Drives," in *2019 IEEE Symposium on Security and Privacy (SP)*, May 2019, pp. 72–87.

[71] E. G. AbdAllah, Y. R. Kuang, and C. Huang, "Advanced Encryption Standard New Instructions (AES-NI) Analysis: Security, Performance, and Power Consumption," in *Proceedings of the 2020 12th International Conference on Computer and Automation Engineering*, ser. ICCAE 2020. New York, NY, USA: Association for Computing Machinery, May 2020, pp. 167–172.

[72] S. Gulley, V. Gopal, K. Yap, W. Feghali, J. Guilford, and G. Wolrich, "Intel SHA extensions: New instructions supporting the secure hash algorithm on intel architecture processors," Intel Corporation, White Paper, Jul. 2013.

[73] H. Fujii, F. C. Rodrigues, and J. López, "Fast AES Implementation Using ARMv8 ASIMD Without Cryptography Extension," in *Information Security and Cryptology – ICISC 2019*, J. H. Seo, Ed. Cham: Springer International Publishing, 2020, pp. 84–101.

[74] NVIDIA Corporation, "NVIDIA BlueField-3 networking platform Datasheet," 2023.

[75] M. Ender, A. Moradi, and C. Paar, "The Unpatchable Silicon: A Full Break of the Bitstream Encryption of Xilinx 7-Series {FPGAs}," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1803–1819.

[76] P. Swierczynski, A. Moradi, D. Oswald, and C. Paar, "Physical Security Evaluation of the Bitstream Encryption Mechanism of Altera Stratix II and Stratix III FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 4, pp. 34:1–34:23, Dec. 2014.

[77] M. Gavahi, A. Naser, C. Wu, M. S. Lahijani, Z. Wang, and X. Yuan, "Encrypted all-reduce on multi-core clusters," in *2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*. IEEE, 2021, pp. 1–7.

[78] M. Chrapek, M. Orenbach, A. Atamli, M. Copik, F. Alder, and T. Hoefler, "snvme-of: Secure and efficient disaggregated storage," *arXiv preprint arXiv:2510.18756*, 2025.

[79] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A Survey on Homomorphic Encryption Schemes: Theory and Implementation," *ACM Computing Surveys*, vol. 51, no. 4, pp. 79:1–79:35, Jul. 2018.

[80] S. Savvides, D. Khandelwal, and P. Eugster, "Efficient confidentiality-preserving data analytics over symmetrically encrypted datasets," *Proc. VLDB Endow.*, vol. 13, no. 8, pp. 1290–1303, Apr. 2020.

[81] L. Burkhalter, A. Hithnawi, A. Viand, H. Shafagh, and S. Ratnasamy, "{TimeCrypt}: Encrypted Data Stream Processing at Scale with Cryptographic Access Control," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 835–850.

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

[82] J. T. Force, "Security and Privacy Controls for Information Systems and Organizations," National Institute of Standards and Technology, Tech. Rep. NIST Special Publication (SP) 800-53 Rev. 5, Dec. 2020.

[83] M. Nieles, K. Dempsey, and V. Pillitteri, "An Introduction to Information Security," National Institute of Standards and Technology, Tech. Rep. NIST Special Publication (SP) 800-12 Rev. 1, Jun. 2017.

[84] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading Kernel Memory from User Space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 973–990.

[85] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest We Remember: Cold Boot Attacks on Encryption Keys," in *17th USENIX Security Symposium (USENIX Security 08)*, 2008.

[86] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the Keys to the Intel {SGX} Kingdom with Transient {Out-of-Order} Execution," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, p. 991.

[87] N. Borisov, I. Goldberg, and D. Wagner, "Intercepting mobile communications: The insecurity of 802.11," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '01. New York, NY, USA: Association for Computing Machinery, Jul. 2001, pp. 180–189.

[88] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," *SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 361–372, Jun. 2014.

[89] M. Morbitzer, M. Huber, J. Horsch, and S. Wessel, "SEVered: Subverting AMD's Virtual Machine Encryption," in *Proceedings of the 11th European Workshop on Systems Security*, ser. EuroSec'18. New York, NY, USA: Association for Computing Machinery, Apr. 2018, pp. 1–6.

[90] E. Barker, "Guideline for Using Cryptographic Standards in the Federal Government: Cryptographic Mechanisms," National Institute of Standards and Technology, Tech. Rep. NIST Special Publication (SP) 800-175B Rev. 1, Mar. 2020.

[91] M. Vanhoef and F. Piessens, "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 1313–1328.

[92] G. Lowe, "Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR," in *Tools and Algorithms for the Construction and Analysis of Systems*, T. Margaria and B. Steffen, Eds. Berlin, Heidelberg: Springer, 1996, pp. 147–166.

[93] J. D. Meulemeester, L. Wilke, D. Oswald, T. Eisenbarth, I. Verbauwhede, and J. V. Bulck, "BadRAM: Practical Memory Aliasing Attacks on Trusted Execution Environments," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Dec. 2024, pp. 103–103.

[94] D. Dechev, P. Pirkelbauer, and B. Stroustrup, "Understanding and Effectively Preventing the ABA Problem in Descriptor-Based Lock-Free Designs," in *2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, May 2010, pp. 185–192.

[95] H. Krawczyk, K. G. Paterson, and H. Wee, "On the Security of the TLS Protocol: A Systematic Analysis," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds. Berlin, Heidelberg: Springer, 2013, pp. 429–448.

[96] I. Arce, "The weakest link revisited [information security]," *IEEE Security & Privacy*, vol. 1, no. 2, pp. 72–76, Mar. 2003.

[97] N. F. Syed, S. W. Shah, A. Shaghaghi, A. Anwar, Z. Baig, and R. Doss, "Zero Trust Architecture (ZTA): A Comprehensive Survey," *IEEE Access*, vol. 10, pp. 57 143–57 179, 2022.

[98] J. O. Coplien, "Curiously recurring template patterns," *C++ Rep.*, vol. 7, no. 2, pp. 24–27, Feb. 1995.

[99] NVIDIA Corporation, *DOCA Documentation v2.10.0*, Jan. 2025.

[100] B. Holmes, J. Waterman, and D. Williams, "SEVeriFast: Minimizing the root of trust for fast startup of SEV microVMs," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '24, vol. 2. New York, NY, USA: Association for Computing Machinery, Apr. 2024, pp. 1045–1060.

[101] T. Hoefler and R. Belli, "Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: Association for Computing Machinery, Nov. 2015, pp. 1–12.

[102] C.-C. Tsai, D. E. Porter, and M. Vij, "{Graphene-SGX}: A Practical Library {OS} for Unmodified Applications on {SGX}," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 645–658.

[103] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "BLAKE2: Simpler, Smaller, Fast as MD5," in *Applied Cryptography and Network Security*, M. Jacobson, M. Locasto, P. Mohassel, and R. Safavi-Naini, Eds. Berlin, Heidelberg: Springer, 2013, pp. 119–135.

[104] J. O'Connor, J.-P. Aumasson, S. Neves, and Z. Wilcox-O'Hearn, "Blake3: One function, fast everywhere," *https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf*, 2021.

[105] H. Wu and B. Preneel, "AEGIS: A Fast Authenticated Encryption Algorithm," in *Revised Selected Papers on Selected Areas in Cryptography – SAC 2013 - Volume 8282*. Berlin, Heidelberg: Springer-Verlag, Aug. 2013, pp. 185–201.

[106] CAESAR Committee, "Final portfolio of the CAESAR competition," Mar. 2019.

[107] P. Shamis, M. G. Venkata, M. G. Lopez, M. B. Baker, O. Hernandez, Y. Itigin, M. Dubman, G. Shainer, R. L. Graham, L. Liss, Y. Shahar, S. Potluri, D. Rossetti, D. Becker, D. Poole, C. Lamb, S. Kumar, C. Stunkel, G. Bosilca, and A. Bouteiller, "UCX: An Open Source Framework for HPC Network APIs and Beyond," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, Aug. 2015, pp. 40–43.

[108] "MVAPICH :: Benchmarks," https://mvapich.cse.ohio-state.edu/benchmarks/.

[109] I. Karlin, J. Keasler, and J. R. Neely, "Lulesh 2.0 updates and changes," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2013.

[110] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the graph 500," *Cray Users Group (CUG)*, vol. 19, no. 45-74, p. 22, 2010.

[111] T. Hoefler, T. Bonato, D. De Sensi, S. Di Girolamo, S. Li, M. Heddes, J. Belk, D. Goel, M. Castro, and S. Scott, "HammingMesh: A network topology for large-scale deep learning," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '22. Dallas, Texas: IEEE Press, Nov. 2022, pp. 1–18.

[112] R. Gerstenberger, M. Besta, and T. Hoefler, "Enabling highly-scalable remote memory access programming with MPI-3 one sided," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: Association for Computing Machinery, Nov. 2013, pp. 1–12.

[113] M. Besta and T. Hoefler, "Fault tolerance for remote memory access programming models," in *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, ser. HPDC '14. New York, NY, USA: Association for Computing Machinery, Jun. 2014, pp. 37–48.

[114] S. Di Girolamo, K. Taranov, A. Kurth, M. Schaffner, T. Schneider, J. Beránek, M. Besta, L. Benini, D. Roweth, and T. Hoefler, "Network-accelerated non-contiguous memory transfers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 1–14.

[115] S. Di Girolamo, D. De Sensi, K. Taranov, M. Malesevic, M. Besta, T. Schneider, S. Kistler, and T. Hoefler, "Building blocks for network-accelerated distributed file systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '22. Dallas, Texas: IEEE Press, Nov. 2022, pp. 1–14.

[116] "Spcl/DNN-cpp-proxies: C++/MPI proxies for distributed training of deep neural networks." https://github.com/spcl/DNN-cpp-proxies.

[117] K. Viswanathan, V. Viswanathan, K. Kumar, T. Willhalm, S. Sakthivelu, and S. Srikanthan, "Intel® memory latency checker v3.12," Oct. 2025.

[118] F. Black and M. Scholes, "The pricing of options and corporate

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

liabilities," *Journal of Political Economy*, vol. 81, no. 3, pp. 637–654, 1973.
[119] H. Bekker, H. J. C. Berendsen, E. J. Dijkstra, S. Achterop, R. van Drunen, D. van der Spoel, A. Sijbers, H. Keegstra, and M. K. R. Renardus, "Gromacs - a parallel computer for molecular-dynamics simulations," in *Physics Computing '92*, RA. DeGroot and J. Nadrchal, Eds. World Scientific Publishing, 1993, pp. 252–256.
[120] M. A. Heroux, J. Dongarra, and P. Luszczek, "HPCG benchmark technical specification," Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2013.
[121] J. Dongarra and P. Luszczek, "HPL-MxP Benchmark: Mixed-Precision Algorithms, Iterative Refinement, and Scalable Data Generation," *The International Journal of High Performance Computing Applications*, p. 10943420251382476, Sep. 2025.
[122] Intel Corporation, "SGEMM/DGEMM benchmark (direct-download archive)."
[123] C. Bernard, M. C. Ogilvie, T. A. Degrand, C. E. Detar, S. A. Gottlieb, A. Krasnitz, R. Sugar, and D. Toussaint, "Studying quarks and gluons on mimd parallel computers," *International Journal of High Performance Computing Applications*, vol. 5, no. 4, pp. 61–70, Dec. 1991.
[124] G. Bauer, S. Gottlieb, and T. Hoefler, "Performance modeling and comparative analysis of the MILC lattice QCD application su3_rmd," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, 2012, pp. 652–659.
[125] S. T. Leutenegger and D. Dias, "A modeling study of the TPC-C benchmark," *SIGMOD Rec.*, vol. 22, no. 2, pp. 22–31, Jun. 1993.
[126] D. H. Bailey, "NAS parallel benchmarks," in *Encyclopedia of Parallel Computing*, D. Padua, Ed. Boston, MA: Springer US, 2011, pp. 1254–1259.
[127] J. Bucek, K.-D. Lange, and J. v. Kistowski, "SPEC CPU2017: Next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. Icpe '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 41–42.
[128] T. Ben-Nun, J. de Fine Licht, A. N. Ziogas, T. Schneider, and T. Hoefler, "Stateful dataflow multigraphs: A data-centric model for performance portability on heterogeneous architectures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 1–14.
[129] NVIDIA. Corporation, "Nvbandwidth," 2022.
[130] W. Constable, X. Zhao, V. Bittorf, E. Christoffersen, T. Robie, E. Han, P. Wu, N. Korovaiko, J. Ansel, O. Reblitz-Richardson, and S. Chintala, "TorchBench: A collection of open source benchmarks for PyTorch performance and usability evaluation," Sep. 2020.

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

Appendix: Artifact Description/Artifact Evaluation

Artifact Description (AD)

I. Overview of Contributions and Artifacts

A. Paper's Main Contributions

- C_1 We introduce SecPerf, the first end-to-end benchmarking platform for continuous evaluation, analysis, and comparison of secure system performance and cost.
- C_2 We propose generalizable security and system taxonomies that contrast security guarantees with performance, enabling expandability and facilitating system comparisons.
- C_3 We implement and open-source (github.com/spcl/secperf) a reference version of SecPerf ready for containerized evaluation.
- C_4 We conduct an extensive evaluation of SecPerf across six cHPC systems, covering 25 cryptographic primitive implementations, secure networking, offloading, and widely available CPU and GPU TEEs, sharing 11 novel key insights and 8 design recommendations.

B. Computational Artifacts

- A_1 https://doi.org/10.5281/zenodo.18712654 with the newest version available at https://github.com/spcl/secperf

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_3	N/A
A_1	C_4	Figures 3-13

II. Artifact Identification

A. Computational Artifact A_1

Relation To Contributions

Artifact A_1 features a reference implementation of SecPerf with hardware configuration scripts and Dockerfiles (C_3). It also contains the benchmarking source code and scripts to evaluate different secure hardware, also in the confidential compute (CC) mode (C_4).

Expected Results

Depending on the section of the benchmarking suite:

- Programmable accelerators for encryption, hashing, and AEAD should reach high throughput (well into GB/s) per core and low latency (less than or around 100ns).
- On a 100 Gbit network, the performance overhead of in-kernel protections for point-to-point UCX benchmarks should be considerable, with offloaded protections obtaining better performance.
- On a 100 Gbit network, throughput should scale with the number of parallel secure tunnels, with resource usage following a similar trend.
- On a 100 Gbit network, both latency and throughput should have a significant overhead for collectives.
- On a 100 Gbit network, the distributed application overheads should be considerably lower for TCP than for IPsec.
- SGX should achieve, on average, lower memory overheads than TDX, and local NUMA should achieve, on average, lower overheads than remote NUMA.
- SEV should achieve, on average, better performance for applications than SEV-ES, which should be better than SEV-SNP, with all overheads staying, on average, at less than 20%.
- Kernel invocation latency, and transfers to a GPU should incur considerable overheads on throughput in the CC case, and be low in the VM case.
- Applications should incur considerable performance overheads in the CC case, and not large ones in the VM case.

Expected Reproduction Time (in Minutes)

Building the containers, obtaining dependencies, and preparing the environment is expected to take up to 120 minutes per machine.

Executing can take up to 360 minutes per benchmark, per setup (secure, non-secure, different configurations). However, this varies considerably between benchmark types. For example, encryption, hashing, and AEAD benchmarks typically complete in less than 60 minutes, while memory benchmarks can take 360 minutes to complete per single setup (SGX, TDX, VM, bare metal). The total expected time to evaluate all benchmarks on all setups reaches potentially tens of thousands of minutes.

Parsing the results, including plotting, is expected to take less than 30 minutes.

Artifact Setup (incl. Inputs)

Hardware: Programmable accelerators can be evaluated on most commodity hardware. DOCA benchmarks require a Bluefield 2 and 3. The networking experiments require a fast network, such as a 100 Gbit one, and an offloading-capable NIC (e.g., BF2 in our case). For the collective experiments, more than one node is needed. For CPU, memory, and application experiments, we require CPUs capable of CC (e.g., modern Intel Sapphire or Emerald Rapids; AMD's EPYC starting from the Milan generation). For GPU memory and application experiments, we require a CC capable CPU and GPU setup (e.g., Nvidia H100 or Blackwell GPUs with one of the above CPUs). Note that the hardware must be configured for CC workloads. While we provide pointers on how to achieve this in the artifact using, e.g., BIOS configuration, certain parts of this process cannot be automated.

Software: While we list the software we used, we also provide these dependencies in the artifact either packaged in Docker containers or supplied with instructions on how to

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.

obtain them. For packages without a version, we used the latest master commit as of the paper's submission.

- Docker 28.1.1 (https://docs.docker.com/engine/install/)
- Python 3.10 (https://www.python.org/downloads/)
- Google Benchmark 1.9.2 (https://github.com/google/benchmark.git)
- CryptoPP 8.6.0 (https://github.com/weidai11/cryptopp)
- Libsodium 1.0.20 (https://download.libsodium.org/libsodium/releases/libsodium-1.0.20-stable.tar.gz)
- OpenSSL 3.0.2 (https://www.openssl.org/)
- DOCA 2.10.0 (https://docs.nvidia.com/doca/sdk/index.html)
- BLAKE3 1.8.2 (https://github.com/BLAKE3-team/BLAKE3.git)
- UCX 1.16 (https://github.com/openucx/ucx)
- OpenMPI 5.0.7 (https://www.open-mpi.org/)
- OSU benchmarks 7.5 (https://mvapich.cse.ohio-state.edu/benchmarks/)
- WireGuard 1.0.20210914 (https://git.zx2c4.com/wireguard-tools/)
- Libreswan 4.12 (https://libreswan.org/) or StrongSwan 5.9.13 (https://strongswan.org/)
- DNN proxies (https://github.com/spcl/DNN-cpp-proxies)
- Black Scholes (https://github.com/IntelPython/BlackScholes_bench)
- GROMACS (https://gitlab.com/gromacs/gromacs)
- HPL-MxP (https://bitbucket.org/icl/hpl-ai.git/src)
- Graph500 (https://github.com/graph500/graph500)
- MILC (su3_rmd) (https://github.com/milc-qcd/milc_qcd)
- LULESH (https://github.com/LLNL/LULESH)
- GEMM (https://www.intel.com/content/www/us/en/developer/articles/technical/benchmarking-gemm-with-intel-mkl-and-blis-on-intel-processors.html)
- SPEC cpu2017 (https://www.spec.org/cpu2017/)
- HPCG (https://github.com/hpcg-benchmark/hpcg)
- NPB 3.4.2 (https://www.nas.nasa.gov/assets/npb/NPB3.4.2.tar.gz)
- TPC-C (https://github.com/cmu-db/benchbase)
- MLC 3.11b (https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html)
- Nvbandwidth 0.7 (https://github.com/NVIDIA/nvbandwidth)
- Pytorch benchmark (https://github.com/pytorch/benchmark)

Datasets / Inputs: All required benchmark inputs are included in the artifact; therefore, no external datasets are needed.

Installation and Deployment: For experimental setup, we provide a Docker container that allows the environment to be recreated. For hardware configuration, we provide scripts and pointers to the necessary setup resources for non-automated flows.

Artifact Execution

The main tasks:

- T_1 Build the Docker environment for all benchmarks.
- T_2 Compile the applications in the Docker environment.
- T_3 Run encryption/hashing/AEAD benchmarks.
- T_4 Configure network to the appropriate protection state (no, IPsec, WireGuard, offload) on all nodes involved in the benchmarking.
- T_5 Run point-to-point benchmarks.
- T_6 Run tunnel scaling benchmarks.
- T_7 Run collective benchmarks.
- T_8 Run network application benchmarks.
- T_9 Configure CPU to the appropriate CC state (baremetal, VM, TDX, SGX).
- T_{10} Run memory benchmarks.
- T_{11} Run application benchmarks.
- T_{12} Detach GPU from the CUDA driver and configure it to the appropriate CC state (baremetal, VM, CC).
- T_{13} Run GPU memory and kernel latency benchmarks.
- T_{14} Run application GPU performance.

We provide convenience scripts in the `src/` directory of each benchmark type to automate the execution of some of these benchmarks. The dependencies:

- $T_1 \rightarrow T_2 \rightarrow T_3$ (Figures 3-5)
- $T_1 \rightarrow T_2 \rightarrow T_4 \rightarrow T_5$ (Figure 6)
- $T_1 \rightarrow T_2 \rightarrow T_4 \rightarrow T_6$ (Figure 7)
- $T_1 \rightarrow T_2 \rightarrow T_4 \rightarrow T_7$ (Figure 8)
- $T_1 \rightarrow T_2 \rightarrow T_4 \rightarrow T_8$ (Figure 9)
- $T_1 \rightarrow T_2 \rightarrow T_9 \rightarrow T_{10}$ (Figure 10)
- $T_1 \rightarrow T_2 \rightarrow T_9 \rightarrow T_{11}$ (Figure 11)
- $T_1 \rightarrow T_2 \rightarrow T_9 \rightarrow T_{12} \rightarrow T_{13}$ (Figure 12)
- $T_1 \rightarrow T_2 \rightarrow T_9 \rightarrow T_{12} \rightarrow T_{14}$ (Figure 13)

The figures in the paper present the input parameters that have been modified from their defaults. For benchmarks where runtime did not exceed an hour, we measured at least ten iterations with 10% warmup runs. The scripts provide exact details for each benchmark, including input and output sizes, as well as the number of iterations.

Artifact Analysis (incl. Outputs)

All benchmark runs output respective CSV files in the `plotting/results/` directory, including the benchmark name in the file name. The Python scripts in `plotting/` with the corresponding benchmark name in the file name can be used to parse the data and generate the corresponding plots. We expect some variability in the evaluation results due to differences in hardware. However, the main trends are expected to remain the same.

Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on August 19,2026 at 11:41:10 UTC from IEEE Xplore. Restrictions apply.