SE-PIM: In-Memory Acceleration of Data-Intensive Confidential Computing

Kha Dinh Duy and Hojoon Lee

Abstract—Demand for data-intensive workloads and confidential computing are the prominent research directions shaping the future of cloud computing. Computer architectures are evolving to accommodate the computing of large data. Meanwhile, a plethora of works has explored protecting the confidentiality of the in-cloud computation in the context of hardware-based secure enclaves. However, the approach has faced challenges in achieving efficient large data computation. In this article, we present a novel design, called SE-PIM, that retrofits Processing-In-Memory (PIM) as a data-intensive confidential computing accelerator. PIM-accelerated computation renders large data computation highly efficient by minimizing data movement. Based on our observation that moving computation closer to memory can achieve efficiency of computation and confidentiality of the processed information simultaneously, we study the advantages of confidential computing *inside* memory. We construct our findings into a software-hardware co-design called SE-PIM. Our design illustrates the advantages of PIM-based confidential computing acceleration. We study the challenges in adapting PIM in confidential computing and propose a set of imperative changes, as well as a programming model that can utilize them. Our evaluation shows SE-PIM can provide a side-channel resistant secure computation offloading and run data-intensive applications with negligible performance overhead compared to the baseline PIM model.

Index Terms—Processor-in-memory, confidential computing

1 Introduction

Today's cloud computing is facing two urgent challenges: improving efficiency in data-centric computation and providing confidentiality of sensitive data computation. Data-intensive workloads such as large data analytics and training of neural networks have become the most common use cases of the cloud. Recent advancements and future research directions in processor architectures, accelerators, and memory technology are also centered around this trend. Ensuring the confidentiality of the computation in the cloud is another agenda shaping the future of cloud computing. Many cloud service providers are already providing confidential computing services that adapt security extensions to modern processor architectures [1], [2], [3].

Many previous works have proposed methods for securing data computation in the cloud based on commonly available processor-supported *secure enclaves* [4], [5], [6]. Secure enclaves such as Intel SGX [7] allow the construction of a robust security model in which the data and its computation are protected from possibly malicious cloud service providers and co-tenants. The code and data inside an enclave are only visible when a thread is in enclave mode and protected from any other software, including the OS kernel. Also, enclave-protected contents are encrypted upon leaving the CPU package to memory. Applications can be modified or developed to protect sensitive code and data by placing them inside enclaves. For instance, VC3 [4] retrofits the MapReduce framework to protect processed data, and Occlumency [5] proposes SGX-protected deep learning inference. Secure enclaves in the cloud also have limited memory (e.g., 128 MB in the case of SGXv1) [8]. This is a critical impediment for adapting SGX to data-intensive applications; existing works resorted to splitting workloads into batches due to the limited memory accepting the performance degradation and additional complexity in programming [5], [6].

Additionally, recent works have revealed that upholding a robust security model of secure enclaves amid surrounding threats proved to be much more complicated than initially considered. Studies showed that *side-channel attacks* are sufficient to compromise the confidentiality of the computation encapsulated by secure enclaves [9], [10], [11]. Especially, researchers have shown that even when the memory content of a secure enclave is automatically encrypted by hardware, the *data access pattern* on the bus could still leak sensitive information about the enclave execution [11]. To this end, many works proposed defenses that hide the data access pattern of secure enclaves. Oblivious RAM (ORAM)-based approaches for SGX have been explored by many works [12], [13], [14]. Those approaches transform the protected program such that its memory access pattern appears random across different executions. However, applying ORAM to a program makes its memory access time and throughout several magnitudes slower than native [12], [13]. In response, hardware-based approaches

• *The authors are with the Department of Computer Science and Engineering, Sungkyunkwan University, Seoul 03063, South Korea. E-mail: {khadinh, hojoon.lee}@skku.edu.*

Manuscript received 25 February 2022; revised 5 September 2022; accepted 8 September 2022. Date of publication 30 September 2022; date of current version 6 September 2023.
This work was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) under Grant MSIT 2022-0-00688 (AI Platform to Fully Adapt and Reflect Privacy-Policy Changes), funded by the Korea Government.
(Corresponding author: Hojoon Lee.)
Recommended for acceptance by J. Zhai.
Digital Object Identifier no. 10.1109/TCC.2022.3207145

are proposed that provide ORAM-like security, but without the prohibitive performance overhead [15], [16], [17]. Those approaches establish a secure channel between a host-side memory controller (software or hardware) and external memory with computing capabilities (e.g., 3D-stacked memory or an FPGA device). For instance, Trustore [17] propose using an enclave-protected software memory controller and a trusted FPGA device that serves as secure memory. The approach attains significant performance improvements over ORAM-based approaches, showing its practicality. However, those approaches leave the side-channels incurred from the enclave execution (e.g., control-flow side-channels) out of scope and only protect the access pattern of the memory controller. The design of SE-PIM is based on the following observation: first, moving sensitive computation from the secure enclaves into memory hides the memory access pattern. Second, it also narrows the attack surface of confidential computations against CPU-based side-channels.

Another struggle in the cloud is to overcome the data movement bottleneck, which is often referred to as the *memory wall problem*. The size of the processed data sets is ever-increasing, and computer architectures and the cloud are evolving to become more efficient in computing over large data sets. Today's computer architectures spend more than half of their cycles moving the data. The near-data processing approach seeks to solve the problem by enabling computation inside storage devices to minimize data movement [18], [19], [20]. *Processing-In-Memory (PIM)* is one of the most prominent directions in the trend [18], [21], [22], [23]. In fact, many semiconductor makers have disclosed their plans for processor-embedding memory devices [24], [25]. The idea is to bring the computation inside memory such that the computation enjoys very low-latency access to data residing in memory and, at the same time, minimizes unnecessary data movement through the system bus. The approach is actively being explored by the industry [24], [26] and in academia [21], [22], [27], [28], [29]. PIM-accelerated computation of large data has shown its potential through many works [27], [28], [30], [31], [32], [33], [34], [35]. Notably, a few works [15], [16], [36], [37] have explored the use of PIM for security. For instance, [15], [16] proposed PIM-assisted address side-channel mitigation on the system bus.

This work presents a novel CPU-Memory cooperation model for confidential computing, based on our observation that PIM can simultaneously achieve confidentiality and efficiency in large data computation in the cloud. Moving data to computation (i.e., the main processor) creates a constraint on the memory channel and may create side channels along the way. We argue that bringing computation to memory instead of the opposite significantly reduces the attack surface against confidential computing. However, security requirements and applications of PIM are still underexplored. We explain our study on the security model and programming model for enclave functionality in PIM and generalize them into a software-hardware co-design called **S**ecure *Computation* **E**xentesion *for* **PIM** (SE-PIM). Then, we show the merits of secure acceleration provided by PIM in two aspects. First, SE-PIM can function as a side-channel resistant trusted memory extension for secure enclaves to overcome the memory limitation. Second, SE-PIM can turn a memory module into secure in-memory accelerators to facilitate fast large data computation that is infeasible in the current processor enclave design (i.e., Intel SGX). We summarize the contributions of this work as the following:

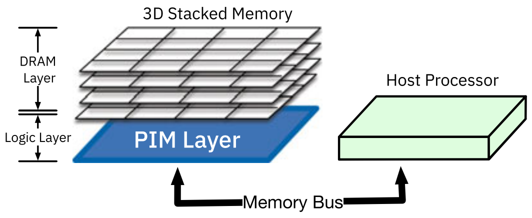


Fig. 1. Processing-In-Memory layer in 3D-stacked memory.

- This is the first work that explores the potential of PIM as a secure computation accelerator and discusses its inherent advantages.
- We present a novel design for confidential computing using PIM by studying the design space and performing a thorough security requirements analysis.
- We outlined the challenges in terms of security and performance when using PIM as an accelerator for confidential computation.
- We propose a set of non-intrusive yet imperative modifications to the PIM architecture for ensuring the confidentiality and integrity of data and computation inside the memory.
- We evaluate our design through a confidential computation of data-intensive workloads on our architecture to show that our proposal for PIM-based confidential computing achieves both efficiency and security.

2 Background and Related Work

This section includes the background and related works of SE-PIM. Our work is primarily motivated by recent developments and the upcoming integrations of PIM architectures described in Section 2.1. Our work introduces a cooperation model for confidential computation that addresses the challenges of CPU-based enclaves (e.g., the lack of memory and side-channels). We explain the background for those challenges and the related work in Section 2.2.

2.1 Processing-In-Memory

Processing-In-Memory (PIM) is a general direction towards embedding processing capability inside memories to overcome the memory bandwidth bottleneck, as today's workload is becoming more and more data-intensive. We briefly explain the existing works on PIM from both the industry and academia.

PIM Hardware Architectures. The progress made in 3D-stacked memory technology enabled the embedding of logic or even processors into memory. Two of the most prominent example of 3D-stacked memory are *Hybrid memory cube (HMC)* [38], and *High Bandwidth Memory (HBM)* [24], [25]. Fig. 1 demonstrates the architecture of 3D-stacked memory and the PIM layer. Stacked memory architectures vertically stack DRAM layers on top of each other and connect the vertical partitions of memory using high-bandwidth through-silicon vias (TSVs). A typical 3D-stacked memory configuration can employ thousands of TSVs [39], which makes its internal memory bandwidth far exceed that of traditional memory systems. At the bottom of the memory stacks is a *PIM layer*, or logic layer, that can host hardware logic that can interact with both the host processor and the DRAM memory. The PIM layer can be special-purpose hardware logic [15], [32] that is embedded into the memory chip, or a general-purpose processor [24], [26], [29], [30], [40] that runs software called the *PIM kernels*, analogous to the kernels loaded onto GPUs. Most 3D-stacked PIM architecture requires changes to the host-side processor [40], but PIM architectures that can integrate with commodity processors also exist [26], [40], [41]. UPMEM [26] introduced its PIM architecture on a traditional DRAM chip. Lee et al. [40] designed and implemented a generalized PIM architecture and programming model to support PIM-accelerated DNN computation on HBM memory. Both of these prototypes have real hardware implementations and are compatible with off-the-shelf systems [40], [41].

2.2 Confidential Computing in Cloud and Challenges

Supporting confidential computing in the untrusted cloud is a common goal shared by the industry and academia, as evidenced by confidential computing services being offered by cloud service providers [1], [2], [3].

Secure Enclaves and the Cloud. Modern processor architectures support secure enclaves for isolated execution of sensitive code and data. Intel SGX [7] is the most commonly available secure enclave in the cloud. SGX provides protected memory pages called *Enclave Page Cache (EPC)* in which the sensitive subset of a program and processed data can be protected. Memory accesses from the processor to the EPC region are automatically encrypted on the memory bus by the memory encryption engine [8]. A remote user can verify the integrity of a deployed enclave through *remote attestation*, a procedure that allows an enclave to *attest* its code and data with a measurement signed by a secure hardware cryptographic key.

Memory Limitations of Secure Enclaves. The limited memory capacity is one of the main factors that hinder the adoption of SGX-based secure enclaves. SGX has a limited EPC capacity of 128MB; large data computation has to be broken down into smaller batches [5], [6]. DNN model-based inference frameworks that are aware of and optimized for the SGX's memory limitations are also proposed [5], [43]. Moreover, it is shown that EPC page swapping is expensive when a large amount of secure memory is used [13] (up to $1000\times$ performance overhead). Designs for external secure storage devices for secure enclaves are also explored by several works to complement SGX's memory limitation [15], [16], [17].

Side-Channel Attacks on Enclaves. Researchers have shown that secure enclaves can fall vulnerable to side-channel attacks. Especially, the microarchitectural side-channels and the data access pattern captured on the memory bus are shown to be exploitable against SGX secure enclaves to undermine the confidentiality of the computed data [9], [10], [11], [44]. Microarchitectural side-channel attacks exploit *microarchitectural resources sharing* to extract sensitive information about the enclave execution [9], [10]. For instance, attackers can infer secret-dependent cache line accesses of an enclave based on the cache timing [10].

The access patterns that incur from memory requests of the secure enclave are also a known side-channel of SGX enclaves [8]. The premise for such a side-channel is that SGX only encrypts the content $data$ of memory packets (Packet $P = (addr, data)$), while the address $addr$ is visible to the attackers that place a physical snooping device on the memory bus. The attack vector is recently realized by the Membuster [11] attack, which demonstrates that the adversaries are able to leak sensitive information from the *address side-channel* of SGX enclaves.

Oblivious RAM and SGX Enclaves. Oblivious RAM (ORAM) [45], [46], [47] algorithms have been accepted as general mitigation for access pattern side-channels. ORAM algorithms transform the original data accesses into sequences of obfuscated accesses that appear uniform to attackers [45], [46]. There have been adaptations of ORAM to SGX enclaves [12], [13] to hide the enclave's access patterns to sensitive data structures. However, the remarkably high performance overhead of ORAM-based approaches remains an obstacle in widespread adoption.

Extending Trust From Enclave to Peripheral Devices. SGX's security model requires the device to actively participate in establishing a trusted I/O path. Many works have proposed secure peripheral devices that complement or cooperate with SGX in confidential cloud computing [15], [16], [17], [42]. These devices support remote attestation and secure communication channel establishment with a host secure enclave while offering varying functionalities.

One of such functionalities is secure hardware acceleration. Graviton [42] proposed a set of GPU hardware modifications to support secure sessions between a host enclave and the GPU and the isolation between GPU contexts. Similarly, Telekine [48] employs *API remoting* to allow remote users to utilize cloud GPUs with Graviton-like features without a host enclave.

Motivated by the high overhead of ORAM algorithms, secure storage designs are proposed as an alternative. These designs offer ORAM-like security guarantees at much lower overheads. TrustOre [17] establishes an encrypted channel between an SGX enclave and the secure FPGA-based device to provide side-channel-free external storage. Invisimem [15] and Obfusmem [16] introduced encryption capabilities to both the CPU-side and the memory-side controllers to build a secure channel on the memory bus. Such designs, however, require modification to the CPU and the memory bus.

Table 1 draws a comparison between SE-PIM and the aforementioned existing works on the secure devices for confidential computing in the cloud. Graviton [42] presented a unique design for supporting confidential computing in GPUs and showed its efficacy through simulation-based evaluation. SE-PIM explores the confidential computing in the new accelerator for large data computation that is PIM. SE-PIM provides both secure storage and computation functionalities. The secure storage design is inspired by TrustOre [17]'s fixed DMA/MMIO channel for secure storage. The existing works [15], [16] equip *smart memory*, which we classify as a type of PIM, with cryptographic primitives

TABLE 1
Comparison Between SE-PIM and Other Proposed Architectures

	Device	Computation	Objective	Method
TrustOre [17]	FPGA	CPU (TEE)	Secure external storage (PCI-E) for CPU TEE, data and addresss confidentiality	Establish secure TEE-FPGA channel with fixed DMA/MMIO address and size
Graviton [42]	GPU	CPU (TEE) +GPU	Confidential computation acceleration using GPU	Introduce GPU supports for secure GPU sessions & GPU contexts isolation
Invisimem [15], Obfusmem [16]	PIM	CPU (TEE)	Secure memory for CPU TEE, data and address confidentiality of bus packets	Extend CPU's memory controller to achive encrypted bus packets with smart memory
SE-PIM	**PIM**	**CPU (TEE) +PIM**	**(1) Secure storage (DRAM), (2) Confidential computation acceleration using PIM**	**(1) DRAM lockdown & controlled data access, (2) Secure PIM sessions**

to the memory device. By doing so, they allow the host CPU enclave and memory to cooperate in eliminating side-channel on the system bus and also facilitate encryption of data stored in memory. SE-PIM, on the other hand, retrofits a more advanced PIM model in which a general-purpose processor inside memory cooperates with the CPU enclave for confidential computation. To this end, SE-PIM identifies and addresses unique attack vectors that arise *during* computation. Another difference is that SE-PIM does not assume host CPU architecture modification.

3 SE-PIM Objectives and Challenges

In this section, we first establish the design objectives of SE-PIM. We then explain the generalized baseline PIM architecture, on which SE-PIM's design proposals construct confidential computing capabilities. Against the threat model, which is also explained in this section, we explain the unique challenges that our design must mitigate.

3.1 Design Objectives of SE-PIM

Our goal is to enable the memory to actively cooperate with a CPU-based enclave to perform confidential computation. Hence, our system functions as a secure memory extension for the host enclave and a secure accelerator for data-intensive computation. We identify the high-level objectives for SE-PIM as follows, while specific security objectives are further discussed in our security analysis in Section 8.1.

SE-PIM *is a Trusted Memory Extension.* One of the SE-PIM's objectives is to function as a secure memory extension that supplements the limited memory capabilities of secure enclaves. The memory limitation of secure enclaves is a substantial problem that impedes the widespread adoption of confidential computing for large data. Many works proposed partitioning large data into batches that fit into the enclave memory space, but at the cost of performance [5], [6]. On the other hand, turning memory into an active entity in secure data storage and processing can solve the problem of limited memory space in cloud enclaves.

SE-PIM *Enables Secure Large Data Computation Acceleration.* PIM-enabled memory devices are on the horizon [24], [25], [26], [40] and their advantages in accelerating data-intensive cloud workloads have been demonstrated [31], [40], [41], [49]. Our objective is to take advantage of the processing capability of PIM by allowing the host enclaves to launch a protected PIM kernel on SE-PIM-enabled memory banks. Furthermore, our hardware modifications must not introduce significant overhead to PIM computation.

We observe that PIM can inherently reduce the attack surface to many side-channel attacks that have been an existential threat to many processor enclaves [9], [44], [50]. Performing computations where the data resides eliminates side-channels from the *runtime* data movement on the memory bus or stored in shared storage, such as the processor caches [9]. We observe that such *zero-copy* data transfer in PIM-assisted computations, inherently minimizing the address side-channels on the memory bus [11].

3.2 Baseline PIM

We first explain a generalized PIM architecture without support for confidential computing and its programming model, named `Baseline-PIM`. Using `Baseline-PIM` architecture, we explain the challenges in SE-PIM's imperative additions to `Baseline-PIM` for confidential computing.

`Baseline-PIM` Architecture. Our baseline PIM model (`Baseline-PIM`) is a generalization of several works that introduces compute capabilities to memory [27], [30], [41]. We assume that in each memory bank of a memory module (e.g., 3D-stacked memory or DIMM memory stick), there is an integration of a programmable processing core (i.e., the *PIM core*). The PIM cores can execute PIM kernels – program binaries compiled to run on a PIM core. As to the processing capacity of these processors, they often lack processor caches [26], [29], [51], and are armed with a low-latency on-chip memory, or *PIM local memory* [26], [29], [52]. The local memory stores the PIM kernel and data a PIM core operates on. The PIM core accesses data inside a memory bank through *direct memory access (DMA)*: it requests the on-chip DMA engine to fetch batches of data into its local memory and process the data there. We do not assume architectural support for cache coherency between the PIM core and the host processor. A cache coherence scheme often requires PIM-aware changes to the host processor architectures [28], [53]. Consequently, our current design does not involve concurrent computation on the same data between the PIM core and the host processor.

Host-Side Access to Memory. The memory bank of `Baseline-PIM` can be used as the normal memory space for the host program. We assume a direct mapping approach for data transfer between the host process and PIM-enabled memory banks: the physical address space of the memory

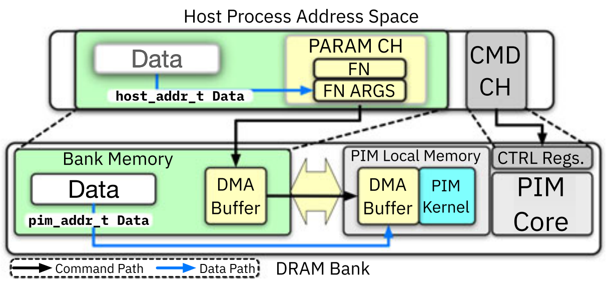


Fig. 2. Communication interfaces between the host and PIM using mappings of memory in host process address space.

bank is mapped by the driver into a contiguous address range in the host process (demonstrated in Fig. 2). The CPU uses the mapped addresses to transfer data into the memory bank directly (the *data channel*). We assume an uncached mapping for the PIM memory banks for data coherency between the host CPU and PIM such that the memory modifications in the host are immediately visible to PIM. `Baseline-PIM` must also support address translation on the PIM core because the host and PIM use different address spaces. To achieve this, the PIM core and the host process share a contiguous memory range with a direct mapping scheme, a common design seen in PIM designs [27]. The PIM core can obtain the in-bank physical address from the host virtual address through a constant offset ($\texttt{pim\_addr} = \texttt{host\_addr} - \texttt{C}$), for all shared addresses. all shared addresses.

Interfaces Between Host and PIM. A host program is typically employed to control the execution of the PIM cores. Similar to most I/O devices, the host process and `Baseline-PIM` communicate using two interfaces, *memory-mapped I/O (MMIO)*, and DMA. For MMIO-based communications, the PIM system exposes registers to the host address space, which are used to send *commands* to the PIM units to control their execution. We refer to this as the *command channel* (CMD CH in Fig. 2). DMA-based communication in `Baseline-PIM` copies data within the memory bank into PIM local memory using the fast PIM bus close to the memory. Hence, PIM-based DMA is different from the DMA communication of other devices that requires data to move across the slow peripheral buses. The host uses the DMA buffer to send *parameters*, a data structure containing arguments for the PIM kernel (e.g., pointers to data inside memory) and commands. We refer to this channel for communication as the *parameter channel* (PARAM CH in Fig. 2). In order to utilize PIM, the host program loads the PIM kernel into PIM local memory (typically via MMIO [29], [41]), writes the parameters that have the function name and its arguments into the parameter channel, and sends a command to the command channel to start executing the PIM kernel. The PIM core then fetches the parameters into its local memory and executes the requested function with the arguments

Zero-Copy Data Exchange. PIM computations typically utilize *zero-copy* data exchanges. As demonstrated in Fig. 2, a PIM kernel receives *pointers* to data already located within memory as the arguments for execution. The PIM kernel then uses the pointers as the source address to fetch data into the local memory for processing with efficient DMA transfers. With zero-copy, data exchanges on the slow memory bus are reduced, and efficiency is achieved in terms of bandwidth utilization and CPU cycles [29], [30], [39].

3.3 Threat Model

We protect the integrity and confidentiality of the data used by SE-PIM and the host enclave. We also guarantee the elimination of observable access patterns to the in-SE-PIM data on the memory bus. We only trust the SE-PIM-enabled memory module and the host CPU enclaves in our threat model. We also assume The physical security of tightly integrated hardware packages. This is a common underlying assumption in secure processor implementation (e.g., Intel SGX [8]) and academic proposals of trusted peripheral devices [15], [17], [42]. PIM hardware packages are often connected with highly integrated connections (e.g.,Through-Silicon Vias (TSV)). For this reason, we assume that physical attacks such as using hardware probes to snoop data contents from the PIM package are infeasible [15], [16].

We assume that the OS can be malicious or compromised by malware. We also assume that the adversaries may have physical access to the machine. More specifically, the adversary may launch a physical probing attack on the system memory bus to leak information about the confidential computation, as such an attack was proven feasible through a research work [11]. Realistically, such physical access can be obtained by, for instance, an employee of the cloud provider who is determined to leak customer secrets out of curiosity or for profit. The above attack model is in line with the security model of SGX [8], [54], SGX-based confidential computation in the cloud [12], [13] , and also secure peripheral devices that support confidential computing [15], [16], [17], [42].

Side-channel attacks that take advantage of *CPU's* microarchitectural characteristics [9], [55] are out of our protection scope. The host enclave can offload sensitive operations to SE-PIM units to reduce the attack surface. Rowhammer attack [56], [57], electromagnetic side-channels [58], power side-channels [59] and similar attacks are not in the scope of our protection. The discovery of such attacks and their mitigation is still an open challenge, and it is rather difficult to formulate a tangible attack model and mitigation specifically for PIM. Hence, we instead discuss orthogonal works that have discussed countermeasures for such attack vectors in Section 8.2.

3.4 Challenges

PIM exhibits many inherent strengths as a side-channel resistant confidential computing accelerator for host enclaves. However, the required design changes for PIM architectures have not been explored to the best of our knowledge. Our contribution lies in identifying the unique challenges and proposing a set of design changes for mitigating them.

We start by introducing rudimentary features that are essential for confidential computation into the aforementioned `Baseline-PIM`. We assume that the attestation and secure channel establishment capabilities are in place, similar to previously proposed secure hardware accelerator architectures [15], [17], [42]. This means that both parties – the host enclave and PIM – communicate through encrypted

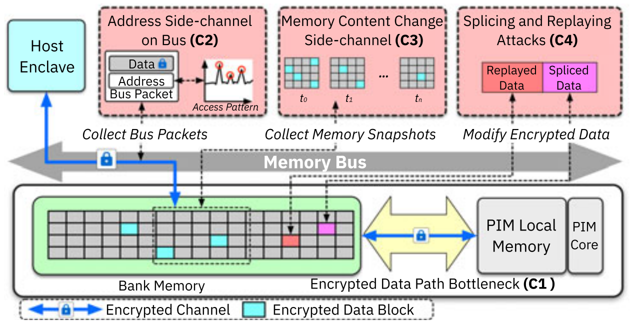


Fig. 3. Challenges faced by `Baseline-PIM`. Encrypted data movement between PIM local memory and the bank memory creates a performance bottleneck (C1). Memory accesses by the host enclave can the *accessed address* on the bus (C2). An adversary can also monitor the memory content changes by quickly taking memory snapshots and comparing differences (C3). Data inside memory is also vulnerable to splicing and replay attacks (C4).

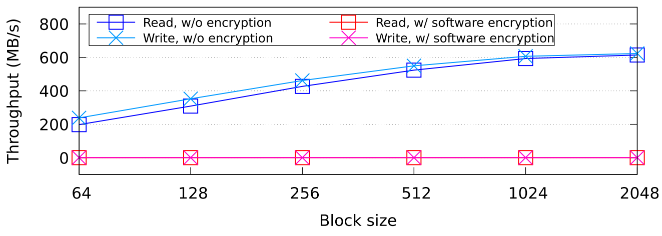


Fig. 4. The memory throughput with and without software AES encryption on UPMEM PIM hardware.

channels to exchange commands and data. The encrypted data is stored in the memory banks and only to be decrypted inside PIM local memory when the PIM core proceeds to process the data.

However, critical challenges remain in securing the workload computation performed in cooperation between the host enclave and PIM. We rigorously inspect all attack vectors that may allow the adversary to leak information about the SE-PIM-assisted confidential computation. Fig. 3 summarizes the challenges that we identify. In the rest of this section, we explain the challenges in depth.

C1: Performance Overhead of End-to-End Encryption. It is essential that the command and data channels between the host and core are encapsulated with an end-to-end encrypted channel. However, we found that this is infeasible without hardware acceleration for encryption. To support our point, we evaluated the performance of working with encrypted data using software-based AES using *mbedtls* on UPMEM's real PIM hardware [26], [41]. The performance of encrypted DMA transfers is measured in both ways; the PIM core encrypts the data before transferring it from its local memory to the memory bank and decrypts the data as the data comes in the reverse direction. We measured the latency of each transfer from the initiation of encryption/decryption to the completion of the DMA transfer.

As shown in Fig. 4, software AES incurs immense performance overhead. The throughput plummeted from 600 MB/s to 0.5 MB/s, around 1200× lower throughput for both read and write. The results convinced us that hardware acceleration for symmetric key cryptography must be included for confidential computing in PIM design. Unlike convention accelerators (e.g., GPU), PIM designs have more constraints in terms of power and area constraints, which will always be their limiting factors [25], [38], [41].

C2: Address Side-Channel on Memory Bus. We observe that when the host access PIM memory banks by writing to the memory mappings of `Baseline-PIM`, an attacker can capture the accessed physical addresses on the memory bus with a snooping device. The attack vector was shown exploitable even on SGX secure enclave, where the memory content is encrypted [11]. Sequential data placement into the memory bank does not generate observable access patterns, but host-side data accesses that are parts of the data processing computation can leak sensitive information about data.

C3: Side-Channels From Memory Content Changes. In `Baseline-PIM`, the memory bank mapping is managed by the untrusted OS. The untrusted OS can directly access the memory content of `Baseline-PIM` through the memory mapping or map the bank memory into a malicious process's address space. In addition, an attacker-controller DMA device on the same system can also access the bank's content. Even though data inside the bank can be encrypted, an attacker can take snapshots of the memory and extract the changes in the memory content made by the PIM kernel. Due to the limited size of the PIM's local memory, the PIM kernel must work with the encrypted data in batches, which is not necessarily sequential. A batch of data will be fetched from the memory bank to be decrypted and processed. Then, depending on the PIM kernel logic, the memory content might be updated with new values. Such operations may inadvertently create recognizable patterns in the eyes of the passive adversary. Research has shown that by only observing the memory change over time, attackers can recover sensitive information about the unencrypted data [60].

C4: Memory Splicing and Replaying Attacks. Using the same access methods described previously, attackers can also modify the content of `Baseline-PIM`'s memory banks. Encrypting data with authenticated encryption algorithms (e.g., AES-GCM) protects encrypted data against direct tampering by attaching an authentication tag to the encrypted data. However, data inside memory is susceptible to *splicing* and *replaying* attacks. The splicing attack replaces data from one location with a valid encrypted data block from another location. The replaying attack reverts a chunk of encrypted data to a previously valid state. Both attacks trick the integrity checking of authenticated encryption algorithms into verifying that maliciously crafted data blocks have not been tampered with.

4 SE-PIM Architecture

The design of SE-PIM specifically mitigates the challenges explained in the previous section. Our design brings imperative changes for `Baseline-PIM` to overcome the unique challenges in achieving its objectives. We explain each architectural change and how they mitigate the challenges in this section. We will further elaborate how a host enclave uses SE-PIM through our programming model in Section 5.2. A detailed security analysis of our design is presented in Section 8.1.

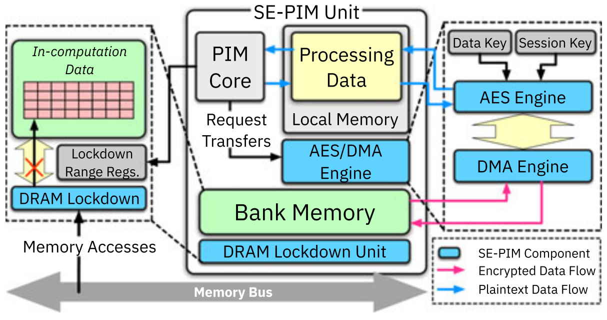


Fig. 5. Overview of SE-PIM hardware architecture.

4.1 Overview

Fig. 5 shows the SE-PIM architecture. SE-PIM hardware design contains modifications to the `Baseline-PIM` architecture mentioned in Section 3.2 to protect the confidentiality and integrity of in-memory computations.

In SE-PIM, an SE-PIM*unit* is included in each *memory bank* present in the memory module. Each SE-PIM unit is equipped with a PIM core, PIM local memory, *DMA/AES engine* and *DRAM lockdown unit*. A memory bank is a unit of secure computation In SE-PIM. Each memory bank is paired with an enclave to provide exclusive and isolated computation acceleration to the enclave. This means that a SE-PIM-enabled memory module can be shared among multiple CPU enclave instances, where each enclave will occupy one or more banks in the module. Also, note that each memory bank, when not used for confidential computation, can be used as a regular memory space for non-enclavized threads in the host.

The PIM core and local memory functions are similar to that of `Baseline-PIM`. The AES/DMA engine introduces hardware acceleration to the DMA engine, significantly accelerating encrypted data transfers. The DRAM lockdown unit enforces *DRAM lockdown*, a mechanism that prevents unauthorized access to the protected data during confidential computation. The DMA/AES engine and DRAM lockdown unit are hardware modules controlled by the PIM core running SE-PIM kernel through MMIO registers. A secure ROM that also functions as a cryptographic key storage for is also introduced, which can be used by all SE-PIM units within a memory module. ROM includes cryptographic primitives (e.g., key pair generation, key signing) and preprogrammed procedures (e.g., attestation) that serve the *commands*.

4.2 Remote Attestation and Secure Communication Channel

Each memory bank is made capable of remote attestation and secure channel establishment through the per-bank SE-PIM unit. More specifically, a secure ROM that (1) hosts code for the attestation and key exchange, (2) functions as secure key storage for the root endorsement key (K_E) (the root of trust) is made available for each memory bank. Such facilities allow host CPU enclaves to attest and exchange a session key with each memory bank.

Remote Attestation. Remote attestation allows a remote challenger (e.g., the CPU-side enclave) to verify the authenticity of the hardware by verifying SE-PIM's certificate with a certificate authority (CA). The host enclave first obtains the SE-PIM unit's certificate with the command `GET_CERT`. The unit first generates an attestation public/private key pair from K_E inside the key storage. It signs the public attestation key with the endorsement key and then writes it to a memory buffer accessible by the enclave. The enclave then verifies the certificate with the CA of SE-PIM (e.g., the manufacturer of SE-PIM). The public key is extracted from a verified certificate and used for key exchange.

Secure Communication Channel Establishment. After remote attestation, the host enclave creates and shares a symmetric session key with a SE-PIM unit to establish an encrypted channel for sending commands and parameters. The session key is encrypted with the public attestation key and sent to SE-PIM as the argument to the command `SET_SESSION_KEY`. The session is used to encrypt all parameters sent to the parameter channel. We will further describe how the channel is initialized and used in Section 5.2. The host also shares a *data encryption key* with SE-PIM using the command `SET_DATA_KEY`. The data encryption key is used to encrypt and decrypt data stored in memory to be processed by SE-PIM. The established communication channel uses authenticated encryption, e.g., AES-GCM, to provide authenticity and integrity. A monotonically increasing counter is also used to achieve resistance against replaying and indistinguishability of messages. Moreover, we employ fix-sized communication messages on the channel to prevent side-channels.

Commands Authorization. The use of authenticated encryption on the parameter channel also prevents unauthorized entities from issuing commands to SE-PIM. When a command does not require parameters (e.g., `LOCK_MEM`) is used, the host enclave encrypts and sends a nonce to the parameter channel to be used by SE-PIM to authenticate the sender of the command. The `GET_CERT` and `DESTROY` commands (described in Table 2) do not require authentication. `GET_-CERT` is used before the secure communication channel is established. An unauthorized `DESTROY` command counts as a denial-of-service and does not compromise the integrity and confidentiality of SE-PIM's execution.

4.3 Hardware Acceleration of Encrypted Data Flow

Since SE-PIM stores encrypted data inside the memory, encrypted data flow is fundamental to our computation model. However, the PIM core is a general-purposed processing core; it must operate on plaintext data. Hence, after loading them into local memory, data from the memory banks must be decrypted before processing. Furthermore, the SE-PIM kernel might need to replace the old in-memory data block with new data after computation, which requires encryption. As we explained in Section 3.4, software-based encrypted data movement between PIM and memory creates a substantial bottleneck in an evaluation on real PIM hardware. Hence, we deem the inclusion of AES acceleration essential for our system to function as an accelerator for large data computation.

AES-Enabled DMA Engine. We propose the use of an AES-acceleration-capable DMA engine. An ISA extension for AES acceleration (e.g., x86 `AES-NI`) can improve the encrypted data transfer performance. However, we argue that it would consume a significant portion of the PIM core cycles that could have been used for data computation.

TABLE 2
Commands Supported by SE-PIM, Their Parameters, Results, and Descriptions

Command	Parameters	Results	Description
`GET_CERT`†	-	`PIM_CERT`	Get SE-PIM's certificate
`SET_SESSION_KEY`	`SESSION_KEY`	-	Send encrypted session key to PIM
`SET_DATA_KEY`	`DATA_KEY`	-	Send data decryption key to PIM
`LOAD_KERNEL`	`KERNEL_BINARIES`	`LOCAL_MEM_MEASUREMENT`	Load PIM kernel
`EXECUTE`*	`FN_NAME, FN_ARGS`	`RESULTS`	Command PIM to execute the kernel
`LOCK_MEM`	-	`BANK_MEASUREMENT`	Enable DRAM lockdown
`MEMCPY`	`DST_ADDR, SRC_ADDR, SZ`	-	Move data within the memory bank during lockdown mode
`ACCESS_DATA`	`DATA, ADDR, SZ, OP`	`DATA`	Data access during lockdown mode
`UNLOCK_MEM`	-	-	Disable DRAM lockdown
`DESTROY`†	-	-	Stop SE-PIM execution, clear local memory and unlock bank

* *Parameters and results are defined by SE-PIM kernel;* † *does not require authentication.*

Also, such a requirement creates an unnecessary constraint in the PIM core design, which is commonly limited in processing power [41]. On the other hand, implementations of AES-enabled DMA engines already exists [61], [62] and can be easily incorporated into existing hardware due to its predictable power and area requirements. The introduction of such an acceleration engine can be an indispensable component in achieving confidential data-intensive computing.

We design the AES-enabled DMA engine as a hardware module integrated with the PIM core through MMIO. The PIM core configures the DMA engine with the keys (`SESSION_KEY` and `DATA_KEY`) obtained during secure channel establishment with the host enclave. After, all data transfers between the bank and PIM local memory by the DMA engine are simultaneously encrypted/decrypted using the shared keys. Data would arrive decrypted at the PIM local memory from the memory bank and vice versa. We further explain the details of the component and our simulation methodology in Section 6.2.

4.4 Preventing Side-Channels and Tampering During Computation

In this section, we elaborate on the components and their functionality that help us overcome the challenges outlined in Section 3.4 (C2, C3 and C4). We introduce two techniques to `Baseline-PIM`, *DRAM lockdown* and *controlled memory access channel*. DRAM lockdown allows the host enclave to disable host-side accesses into the memory region containing data, thus hiding the memory content changes leaked by the SE-PIM kernel's computation. We also eliminate the splicing and replaying attacks by preventing unauthorized memory writes. Our controlled memory access channel provides an interface for the host enclave to perform memory accesses without leaking the address side-channels. It accepts encrypted memory requests through the parameter channel and let SE-PIM performs the actual memory operation on behalf of the host enclave. The channel hides access pattern leakages on the memory bus when DRAM lockdown is active. We now go over the detailed description of each technique.

4.4.1 DRAM Lockdown

Our DRAM lockdown mechanism efficiently addresses both C3 and C4, by prohibiting access to data processed by SE-PIM during PIM computation. We observe that most data movements occur at the start of the PIM workload, where the host loads data into the memory bank. After the initial data transfer, the data rarely leaves the memory banks and is commonly reused in-place by the PIM core. Moreover, the same in-memory data can be processed by different PIM kernels. Hence, the direct access to memory from the host side can be safely disabled during PIM computation.

We now compare our solution with alternative designs. Integrating ORAM algorithms (described in Section 2.2) into SE-PIM's processing core [47] could hide its memory access pattern (C3), but would incur significant performance degradation and complexity to our design [13], [47]. Ultimately, using ORAM would render SE-PIM unusable as an accelerator for large data computation. To provide splicing and replaying protection (C4), a *memory verification engine* that maintains a counter for each encrypted memory block to guarantee its freshness could be introduced [63], [64]. However, introducing such a mechanism does not address C3, requires extra counter storage, and would inadvertently introduce extra overheads for memory accesses [63], [64].

Based on the observations above, we conclude that denying host access to the memory regions containing data for SE-PIM computation is essential to eliminate the memory content change side-channel (C3) while also providing protection against splicing and replaying (C4). We observe that the placement of data into the memory banks only leaves a *sequential* pattern with very little information to be leaked. This is because the memory side-channels happen *during* calculation due to a distinct pattern of behavior in the code [9], [11], [14]. Hence, we allow the usage of direct memory mappings to load data into the memory banks efficiently. After data have been located inside memory, the host enclave can request a *lockdown* of memory by sending a `LOCK_MEM` command to the SE-PIM unit. SE-PIM, in turn, configures the lockdown range registers in DRAM lockdown unit only to allow access to the DMA-based secure communication channel, which is protected by authenticated encryption. We describe the implementation of this mechanism in Section 6. To remain functional as memory storage during bank lockdown, we provide a data channel that allows the host to send encrypted memory requests to SE-PIM, which we describe in Section 4.4.2.

Bank Memory Integrity Measurement. Even with the lockdown mechanism, the attacker could still perform tampering attacks (e.g., splicing and replaying) in the time window between the data load and DRAM lockdown. We implement a memory integrity measurement to thwart such attacks. Upon receiving the `LOCK_MEM` command, SE-PIM takes the measurement of the data received and sends it to the host enclave. The host enclave compares the measurement to the precomputed hash value of the data to confirm that it has not been tampered with. Measurement must be performed every time the memory bank is unlocked for data transfer, then locked to guarantee memory integrity.

4.4.2 *Controlled Memory Access Channel*

We introduce an interface for the host enclave to access the memory bank *during* DRAM lockdown, called the *controlled memory access channel*. The channel provides exclusive access to only the authenticated entities (i.e., the host enclave) while the DRAM lockdown is in effect. We adopt a similar approach to the related works that build a secure communication channel between the secure enclave and memory [15], [16], [17]. We introduce the `ACCESS_DATA` and `MEMCPY` commands for the host enclave to securely perform data access and movement (Table 2). The controlled access channel has the *access addresses*, the *size*, and the *operation* of the memory request encrypted as parameters of the new data access commands. Upon receiving the commands, SE-PIM PIM core performs the actual data access based on the command parameters on behalf of the enclave. Since the DRAM lockdown mechanism already protects the access pattern of the PIM core, memory accesses from the host enclave performed through this interface are also resistant against the *address side-channel* on the memory bus (C2).

Controlled Data Access. `ACCESS_DATA` allows the host enclave to read or write a block of data during bank lockdown while hiding the access pattern from observers. Parameters for `ACCESS_DATA` are the encrypted data block, the requested address, the size of access, and the type of operation (read or write). Instead of introducing separated commands for data read and write, we define the type of access in the parameters to mimic the security guarantees of oblivious RAM algorithms [45]. Fig. 6 shows the data access from the host enclave using `ACCESS_DATA`. First, the encrypted parameters are written into the DMA-based secure channel, and the command is sent to the SE-PIM unit. The SE-PIM unit will decrypt and fetch the memory request into its local memory on receiving the commands. The data block is placed in the requested location for a write request. Otherwise, on reading requests, the SE-PIM unit fetches the block, encrypts, and writes it to the secure channel. Collaboration from the host enclave is required to achieve full ORAM-like security properties. Namely, to hide the type of memory access, the memory requests must contain dummy data on a read, and a dummy read request must follow every write request.

Controlled Data Movement. The `MEMCPY` command instructs SE-PIM to copy data from one address to another, which is useful when data is already located inside the memory. On receiving the command, SE-PIM will fetch the requested data block to local memory and write it to the destination address. The operation performs data movement using the fast internal bus on PIM, which reduces expensive data transfers on the memory bus.

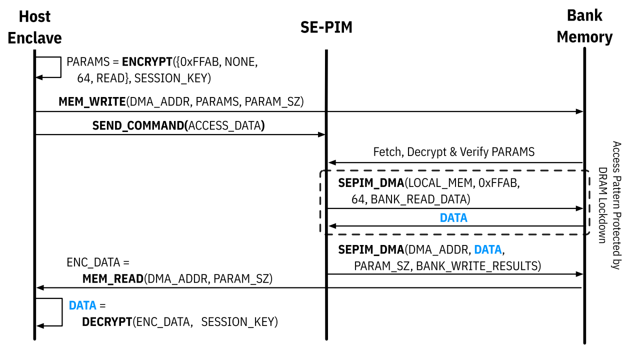

Host Enclave
SE-PIM
Bank Memory
PARAMS = ENCRYPT({0xFFAB, NONE, 64, READ}, SESSION_KEY)
MEM_WRITE(DMA_ADDR, PARAMS, PARAM_SZ)
SEND_COMMAND(ACCESS_DATA)
Fetch, Decrypt & Verify PARAMS
SEPIM_DMA(LOCAL_MEM, 0xFFAB, 64, BANK_READ_DATA)
DATA
SEPIM_DMA(DMA_ADDR, DATA, PARAM_SZ, BANK_WRITE_RESULTS)
ENC_DATA = MEM_READ(DMA_ADDR, PARAM_SZ)
DATA = DECRYPT(ENC_DATA, SESSION_KEY)
Access Pattern Protected by DRAM Lockdown

Fig. 6. An example of data read using controlled memory access channel. The data access pattern of host is protected by encryption, while the data access pattern of PIM kernel is protected by DRAM lockdown.

5 SE-PIM PROGRAMMING MODEL

In this section, we explain the programming model of SE-PIM. Our programming model supports the two design objectives of SE-PIM that we outline in Section 3.1: a secure memory extension and a secure large data accelerator for the secure enclave. We elaborate the software components of SE-PIM that help an enclave interact with SE-PIM in Section 5.1. Then, we go through the design of our programming model that enable our design objectives in Section 5.2.

5.1 Software Components

As the programming model of SGX enclaves does not support a trusted I/O path, the communication channel between the enclave and the devices has to be set up by the untrusted kernel and host application. Hence, there are untrusted and trusted components in our software infrastructure. Our trusted software includes the enclave host program, the trusted SE-PIM user library, and the SE-PIM kernel. Our untrusted components are the kernel driver and the untrusted user library. The untrusted components only set up the secure communication channel; after a secure session is established, all communication between the enclave and SE-PIM is encrypted and authenticated.

SE-PIM *Driver and User Library.* The user library and the driver initialize the communication interfaces between the host and the SE-PIM unit. The driver maps the physical addresses corresponding to the memory banks in the SE-PIM-enabled memory modules into the host kernel address space. It also maps the registers of each SE-PIM unit to expose the device control channel to the host. In order to provide access to SE-PIM, the driver creates a virtual file interface to the host library in the user space (e.g., `/dev/pim0`). A user library at the host side is required to allow an in-host program to communicate and send workloads to a SE-PIM unit. The SE-PIM library has two components, a trusted part that resides in the enclave code and an untrusted part in the host program. The untrusted part of the library initializes the communication interfaces by obtaining the device driver's memory banks and control registers. The trusted library contains functions that facilitate the communication between the enclave and

SE-PIM units (e.g., transfer data to the memory bank and sending commands to SE-PIM).

Enclave Host Program and SE-PIM *Kernel.* The enclave host program contains the confidential computation logic that uses SE-PIM as its accelerator. SE-PIM kernel includes protected functions to be executed inside SE-PIM. The developer of the host enclave must select the computation (e.g., functions) to be offloaded, then implement and cross-compile them for SE-PIM. Similar to the interfaces of `Baseline-PIM`, host enclave communicates with the SE-PIM kernel running on the SE-PIM unit core through two physical interfaces (described in Section 3.2): *memory-mapped registers* and *direct memory access (DMA)*. On the two interfaces, we establish two communication *channels*. The *command channel* (CMD CH) use the memory-mapped registers to issue *commands* (listed in Table 2) to the PIM core to control its execution.

The *parameters channel* (PARAM CH in Fig. 2) uses DMA to send parameters of the commands. It functions as a secure communication channel between the host enclave and a SE-PIM unit. Messages through the channel are encrypted using a shared session key (`SESSION_KEY`) yielded during the initial attestation and key exchange (Section 4.2). On `EXECUTE` commands, the parameters become the arguments for execution that are defined by the SE-PIM kernel.

5.2 SE-PIM Usage Model

Listing 1 demonstrates the code example of a host-side enclave that uses a SE-PIM unit as an accelerator. We will use the example in the remaining of this section to demonstrate SE-PIM's usage model.

Listing 1. Host-Side Code Snippet Illustrating Computation Offloading to `SE-PIM`

```
1  // Initialize and and attest a SE-PIM unit
2  SE_PIM *se_pim = SE_PIM_init(MEM_MODULE_ID,
3                               BANK_ID,
4                               "kernel.elf.enc");
5  // Transfer the data to SE-PIM memory bank
6  void *ptr = se_pim.transfer(obj, DATA_SZ);
7
8  se_pim.lock_mem();
9
10 // Send command, execute and get result
11 SE_PIM_params_t params = {.fn = "increment",
12                           .data_ptr = ptr,
13                           .size = DATA_SZ};
14 for (int i = 0; i < ITERATIONS; i++)
15    SE_PIM_result_t *result = se_pim.execute(params);
16 // Read in-memory data
17 se_pim.read(result, ptr, DATA_SZ);
```

Initialization. The enclave first obtains an object representing a SE-PIM unit with the library function call `SE_PIM_init` (line 4). The call performs remote attestation and key exchange as described in Section 4.2, then offloads the encrypted SE-PIM kernel to the SE-PIM unit. Note that if more than a single SE-PIM unit is employed, the initialization function needs to be called for each SE-PIM unit identified by the memory bank it belongs to (`BANK_ID`).

Extending Enclave Memory With SE-PIM . The user library provides functions to offload data into the protection of SE-PIM and access data without leaking data access patterns on the memory bus. Its data loading function (line 6 of Listing 1) uses direct memory mappings to efficiently load data into the SE-PIM memory banks without compromising security. As we discussed in Section 4.4.1, the act of sequential data loading does not generate distinguishable access patterns. Furthermore, we observe in Section 7.2.2 that using the controlled memory access channel that is side-channel resistant, would inevitably create overheads for large data transfers. After the initial transfer, the enclave commands the memory bank to enter *lockdown* to protect the data inside memory (line 8).

After DRAM lockdown, data within memory is accessed through the user library functions `read()`, `write()` and `memcpy()`. The functions internally create encrypted memory requests to the controlled memory access channel (demonstrated in Section 4.4.2), and let SE-PIM perform the actual data access. Hence, the data access pattern is protected from any observing attacker. In the example in Listing 1, the host enclave use the function `read()` to read the incremented data block from memory without leaking the requested address and the type of operation (line 17).

Listing 2. `SE-PIM` Kernel Code Snippet Fetches Data into Local Memory, Increments, Writes Updated Data to Bank Memory, and Finally Returns the Results to the Host Enclave

```
void increment(data_ptr, size){
   char local_mem[BATCH_SZ];
   SE_PIM_result_t result;
   result.count = 0;
   // Request data from bank memory
   for (int i = 0; i < size / BATCH_SZ; i++){
      sepim_dma(data_ptr + BATCH_SZ * i, local_mem,
                BATCH_SZ, BANK_READ_DATA);
      // Process data in local memory
      for(int j = 0; j < BATCH_SZ; j++){
         local_mem[i] += 1;
         result.count++;}
      sepim_dma(local_mem, data_ptr + BATCH_SZ * i,
               BATCH_SZ, BANK_WRITE_DATA);
   }
   return_result(&result);
}
```

Offloading Confidential Computation to SE-PIM . Listing 2 demonstrates the kernel logic used in Listing 1. After remote attestation and key exchange, the encrypted SE-PIM kernel binary is offloaded through the secure channel as the argument to the command `LOAD_KERNEL`. The kernel is loaded into SE-PIM local memory and used by the PIM core for execution. Then, the PIM core takes the measurement (i.e., the hash) of PIM local memory and sends the measurement to the host enclave. The host enclave can use this measurement to verify the initial state of a SE-PIM unit. After loading the PIM kernel, the SE-PIM unit goes into an idle state and wait for future commands (e.g., `EXECUTE` or `LOCK_MEM`).

SE-PIM makes use of the *zero-copy* model of PIM-based computation, as described in Section 3.2. Data is not moved for computation, but only the *pointers* to the data are embedded in function arguments (`SE_PIM_params_t`). SE-PIM uses the pointers to fetch the actual data from the bank memory into

TABLE 3
Configuration of the Simulated System

Simulation Configuration	
Host CPU	8 in-order ARMv7 cores (2 Thr/core, 4.0 GHz)
CPU Cache	L1: Per-core, 32KB I-cache, 64KB D-cache, LLC: Shared across cores, 256 KB
OS	Linux kernel 3.16.0-rc6
Memory	8 PIM-enabled memory banks, 64 MB/bank, Memory type: `gem5 HMCVault`
PIM Cores	1 × in-order ARMv7 core/bank (1 thread/core, clock rate 1 GHz), 1 MB local memory/core

its local memory and perform the requested operation on the data. This eliminates the data transfers for each computation and makes use of the fast internal bus of PIM systems [38], [39]. Moreover, the computation does not leave any pattern on the memory bus. This is only possible on PIM-based computation since CPUs, or any accelerators on the peripheral bus, inevitably access memory with patterns unless using ORAM primitives. To command a SE-PIM unit to execute the kernel, the host sends the encrypted parameters through the encrypted parameter channel; then sends an EXECUTE command to the corresponding SE-PIM unit. After execution, SE-PIM kernel encrypts and returns the execution results to the enclave through the same encrypted channel (line 16).

Supporting Even Larger Data Computation. The above SE-PIM usage model can be repeated to compute data larger than the maximum capacity of SE-PIM-enabled memory module. Similar to the usage of other types of accelerators (e.g., GPUs), the workload can be split into batches to compute data larger than finite memory capacity. Once a batch of workload has finished, the host enclave may save the encrypted batch output to a larger normal DRAM space, unlock the SE-PIM bank and proceed with the next batch.

6 Implementation

We implemented a prototype of the SE-PIM design using a cycle-accurate full system simulation [65]. Regarding the low-level simulation configurations, we referenced well-documented industry-grade products [26] and simulation configurations from other works on PIM [16], [27], [29], [30]. Our simulation is well modularized and generalizable; the hardware components that we implement can be easily modified and provides well-defined APIs for writing host-side and PIM-side applications.[1]

6.1 PIM Core and ROM

The PIM cores are general-purpose execution cores that use the local memory as their RAM and have low-latency access to the memory banks. The specifications for the cores are described in Table 3. Fig. 7 demonstrates the address space of the PIM core and the memory mapping of the components into the address space. Local memory is mapped into the address space of SE-PIM as a contiguous memory region, which it uses as the memory space for the SE-PIM kernel, the stack space, and DMA transfers. The registers of the AES-enabled DMA engine and the DRAM lockdown unit are memory-mapped to fixed addresses so that the PIM core can configure and send commands to the components.

ROM contains the code for cryptographic primitives (e.g., key generation) and functions for handling the commands. The list of supported commands is demonstrated in Table 2. Our prototype implements ROM code directly inside the PIM kernel code and loads it to the local memory. This would simplify the implementation while achieving the same functionality and performance characteristics. ROM must be implemented in a separated read-only memory module to protect the security-sensitive code against tampering in an actual implementation. Moreover, PIM core should be implemented so that only the attestation code can access the key storage containing the root endorsement key to prevent key leakage. As noted by [66], the requirement can be easily enforced with an introduction of a hardware monitor on a PIM core's program counter that only allows access to the K_E when the PC is within the attestation code.

6.2 AES-Capable DMA Engine

We implemented and incorporated the AES-GCM acceleration capability in the DMA engine of the PIM core package. Our strategy in incorporating a hardware design into the simulation was two-track: we first modified the DMA engines in the memory controller of the PIM core package to include AES-GCM acceleration hardware to test the correctness aspect. Then, we calculated the increased number of cycles consumed for each DMA transfer by inspecting the specifications of AES hardware specifications.

Functional Correctness. We implemented AES functionality in the memory controller and its DMA engine of each PIM core. The implementation is based on open hardware IPs [61], [67] and uses functions from the `OpenSSL` library. We expose the configuration registers for various configuration parameters and commands by mapping them into the PIM core address space (Fig. 7). Through the commands, the PIM core can specify whether the direction of data indicates whether decryption or encryption should be performed; decryption is used to transfer from the memory

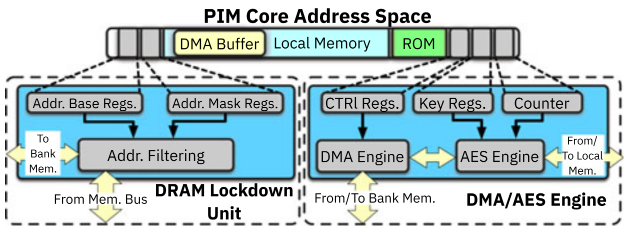


Fig. 7. The address space of a PIM core. The control registers of the DRAM lockdown unit and the AES-enabled DMA engine is mapped into the address space of the PIM core for MMIO communication.

1. We plan to make our simulation public after the publication of this work

bank into PIM local memory bank and vice versa. The DMA engine also uses different encryption keys (`SESSION_KEY or DATA_KEY`), depending on the command. Upon receiving a command from the PIM core, the DMA engine initiates the transfer, then encrypts or decrypts the contents appropriately. Listing 3 demonstrates the function `sepim_dma()` that use the AES-capable DMA engine for encrypted data movement.

Listing 3. The Definition of `sepim_dma` Inside a SE-PIM Kernel. The Function Uses the Control Registers of the DMA/AES Engine Mapped into PIM's Address Space

```
void sepim_dma(src, dst, sz, cmd){
   write_reg(DMA_SRC_ADDR_REG, src);
   write_reg(DMA_DST_ADDR_REG, dst);
   write_reg(DMA_SZ_REG, sz);
   write_reg(DMA_CMD_REG, cmd);
   // Wait for transfer to finish
   wait(DMA_STATUS_REG);
}
```

Simulating Latency. The performance overhead of incorporating AES-GCM acceleration into DMA engines is well-studied and can be accurately estimated. We adapt the latency described in the specification of the IP [67]. Also, previous works have estimated the latency of almost identical hardware [15], [16]. We estimated that the AES-GCM accelerator can operate at a 300MHz clock rate and could encrypt 128-bit blocks each cycle. Based on the estimation, we add a delay of 1 cycle to each 16 bytes data chunk processed by the DMA engine in our simulation.

6.3 DRAM Lockdown Unit

We discuss the implementation of the DRAM lockdown unit in different PIM architectures. For 3D-stacked memory, the implementations often use packetized interfaces that replace the traditional low-level DDR commands [15], [38], [68]. Hence, inserting the logic for denying access to a region inside memory is straightforward. Many works have taken advantage of the flexibility of 3D-stacked memory to incorporate authenticated encryption into the packet handling logic [15], [16]. Therefore, we expect our proposed logic change that implements DRAM lockdown can be easily incorporated into 3D-stacked PIM hardware. On the other hand, it is more challenging to implement a filtering mechanism on DIMM-based memory since the DDR interface is very sensitive to timing differences [69]. Hence, we focus on the 3D-stacked memory-based implementation in this work.

Simulating DRAM Lockdown. To simulate DRAM lockdown, we introduce an *address filtering mechanism* into the packet handling logic of the memory. We mapped two registers, *address start* and *address mask* to the address space of a PIM core, which it can use to configure the protected address range (Fig. 7). The address start register specifies the start of the protected address range. The address mask register is combined with the address start register to specify the upper bound of protected addresses. In our simulation, we made modifications to the AbstractMemory object in gem5, which is an abstraction of the memory-side memory controller. As AbstractMemory uses packets to interact with other components, its functionality is similar to how the logic layer of 3D-stacked memories handles the requests.

7 Evaluation

We perform evaluations to show vital aspects of our design: first, offloading computations to SE-PIM leaks no address-based side-channels on the bus. Second, our introduced modifications do not inhibit PIM's ability to perform data-intensive computation efficiently. Finally, we demonstrate SE-PIM's benefits as enclave extended memory. We first show the lack of bus traffic side-channel in Section 7.1. We perform a set of microbenchmarks to evaluate the performance of encrypted data movement and the secure memory access interface in Section 7.2. Lastly, we evaluate the data-intensive computation performance of SE-PIM-assisted confidential k-mean clustering application in Section 7.3.

Evaluation Setting. We evaluated SE-PIM through a cycle-accurate simulation. The simulation configuration we used for simulating our system can be found in Table 3. On the host system, we allocate 8 cores with a clock rate of 4 GHz throughout our experiments, for this configuration is sufficient to fully saturate the host memory bandwidth. On the PIM side, we configure each memory bank to have 64 MB of memory, and the PIM cores to operate at the clock rate of 1 GHz to reflect the power and area constraint in the DRAM or 3D-stacked memory packages. The timing parameters used for the simulation were determined through existing works that conducted simulations on similar hardware [27], [29], [70].

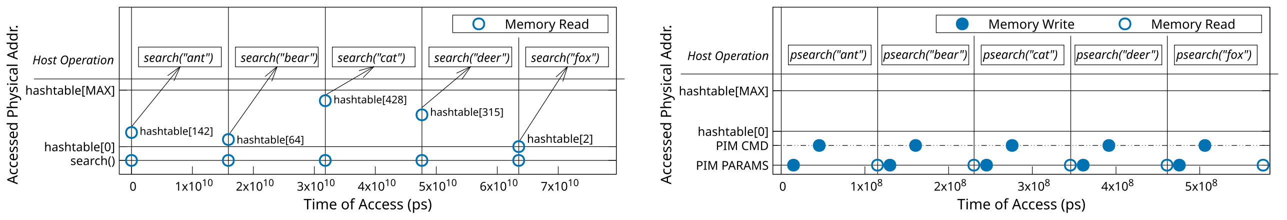
Listing 4. Dictionary Word Lookup Function Used for Memory Access Pattern Analysis

```
struct data_item *search(const char* key) {
   int index = murmur3_hash(key) % TABLE_SIZE;
   while (hashtable[index].value != -1) {
      if(key_match(hashtable[index], key))
         return &hash_array[index];
      ++index;
      index %= TABLE_SIZE;
   }
   /* Key not found */
   return NULL;
}
```

Note that there might be discrepancies between our simulation setup and a real hardware implementation. We discuss in details some of such possible differences in Section 8.2.

7.1 Memory Access Pattern Analysis

We set up an experiment that imitates the off-chip bus snooping attack illustrated in Membuster [11], which demonstrates that bus snooping can undermine the confidentiality of processor enclaves through memory access pattern analysis on the bus traffic. We use a minimal hash table lookup that mimics the attack examples on the `hunspell` dictionary program shown in Membuster. Each entry in the hash table stores a hash key and an integer value in the place of the word's definition for simplicity. The attack illustrates how the address side-channel observable on the bus can reveal the behavior of the seemingly protected computation.



(a) Memory access pattern observed on memory bus during host enclave dictionary lookup. (b) Memory access pattern observed on the memory bus during SE-PIM-assisted dictionary lookup.

Fig. 8. Memory access pattern observed on bus during confidential `hunspell` dictionary program. In host CPU enclave-only model, accessed `hashtable` location reveals looked up word. In SE-PIM-assisted model, observed memory pattern is consistent for all looked up words.

We implemented a `search()` function that takes a string as an argument. The argument is hashed using the *Murmur3* hash algorithm, then used as an index to the hash table. The exact operation of `search()` is defined in Listing 4.

The original attack from [11] requires that the host enclave caches are constantly flushed to yield reliable and distinct access patterns on the bus. We adapted the same premise for our experiment. However, since the gem5 x86 CPU model does not correctly simulate the cache flush instructions (e.g., `clflush`), we implemented a function that forces cache flush by repetitive memory accesses and used it after each hash table access to preventing caching. In the host-only setup, which represents a case where the host processor enclave is used to protect the hash table, we place the hash table inside a single memory page so that it is contiguous in physical memory. Then, we obtain the virtual to physical mapping of the program from `/proc/process pidpagemap`. For the PIM-assisted version of the program, the hash table is stored in the PIM memory bank, and a PIM kernel that performs hash table lookups is loaded into the PIM processor local memory. We leverage our simulation framework to place a memory tracing unit between the L2 cache and DRAM (i.e., the memory bus) to collect the accessed physical addresses. Fig. 8 illustrates the memory access pattern observed on the bus – by the attacker who is snooping on the bus traffic – in host processor execution and SE-PIM-assisted execution. Fig. 8a shows the memory access pattern recorded during the execution of the host-only version of the test program, and Fig. 8b shows that of the SE-PIM-assisted version (`psearch()`). The memory writes are denoted by • and reads by ∘.

Results. In the CPU-only version, with pre-obtained knowledge on the addresses of the functions and data of the program, the adversary learns that `search()` has been executed and in-memory dictionary locations, such as `hashtable[7]`, `hashtable[237]` are accessed. Eventually, they can infer the queried words by looking at the bus traffic. On the other hand, the PIM-assisted version shows only the communication with PIM. The PIM command channel and parameter channel are fixed single-channel memory-mapped I/O channels. The host program first writes the parameter (e.g., `"apple"` to the parameter channel, then writes the command number that corresponds to the PIM function `search()` in the PIM kernel to the command channel to invoke the offloaded task. These patterns are identical to all PIM kernel invocation, regardless of the invoked function and parameters. This example clearly shows the security advantages of SE-PIM against side-channel attacks that target the bus traffic. We further discuss the security of SE-PIM in a more comprehensive manner in Section 8.2.

7.2 Microbenchmarks

We measure the performance of encrypted DMA transfer and the secure memory access interface with our microbenchmarks. The results suggest that encrypted DMA transfer would incur only minor overhead in each transfer, and the throughput of the controlled memory access channel would scale with the transfer size.

7.2.1 Encrypted DMA Transfers

We evaluate the overhead of encrypted data movement between PIM local memory and the memory bank, when accelerated by the hardware AES engine. The average throughput of the operation is also evaluated. For both PIM-baseline and SE-PIM, we measured the access time for the following cases: 1. in both directions (local-to-bank and bank-to-local), 2. sequential and random accesses and 3. for varying block sizes. Note that since we compare the latency of AES-accelerated DMA transfer against DMA transfer *without* encryption, the difference in performance between the two would also precisely demonstrate the performance cost of using AES-GCM in SE-PIM.

Results. Fig. 9 illustrates the average access time measured for 1000 DMA accesses in each configuration. The access time for random and sequential access is roughly similar because there is no cache on PIM that incurs variations in access time. Our simulation results report an average of 22.35% increase in access time due to the AES-capable DMA engine. The measured overhead is in line with the previous works that simulated the performance overhead of hardware AES support [15], [16]. We further investigated the throughput for data transfer between the memory bank and local memory. The DMA maximum throughput of moving data with the AES-capable DMA engine is compared to that of without encryption in PIM-Baseline. Overall, the AES-enabled DMA engine produces an average throughput of 2.9 GB/s, compared to 3.53 GB/s for unencrypted DMA accesses.

Our experiments show that SE-PIM's AES acceleration renders encrypted data transfers via DMA feasible for large data computation. When compared to the overhead incurred from software encryption that we demonstrated in Section 3.4, encrypted data transfer between the memory bank and SE-PIM local memory is no longer the bottleneck. This is further confirmed in our application performance evaluation

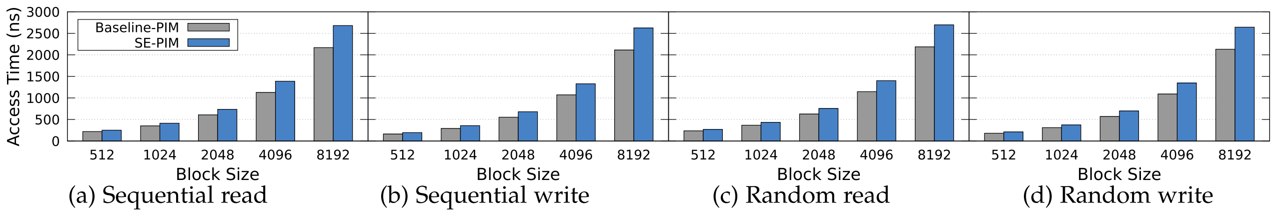


Fig. 9. Access latency of SE-PIM's AES-encrypted DMA versus Baseline-PIM without encryption.

in Section 7.3, where SE-PIM is only marginally slower than `Baseline-PIM`.

7.2.2 Controlled Data Access Throughput

We measure the throughput of data access on different block sizes, performed through the controlled data channel (Section 4.4.2) and compare it to the throughput of normal DRAM access to the memory. Additionally, we also measure the throughput of Path ORAM [45], one of the fastest and most common ORAM algorithms for comparision.

Results. Fig. 10 illustrates the results of the evaluation. Only read accesses are shown since the performance of read and write accesses is almost identical. The throughput of normal memory access is roughly the same among different block sizes, about 1300 MB/s. The results show that when small block sizes are used, a large proportion of the overhead comes from the communication channels (e.g., the MMIO accesses for writing commands), which significantly degrade the throughput of the controlled memory access channel. For instance, the throughput of 1 KB block accesses is 127× lower than normal memory accesses.

Interestingly, the throughput of SE-PIM's controlled memory access increases when increasing block sizes are used, 362.7 MB/s for 64 KB blocks and 504.2 MB/s for 128 KB blocks, while the throughput of Path ORAM decreases. This suggests that using larger block sizes for controlled memory accesses leads to better performance. Despite having a lower throughput than direct memory access, the interface is faster than most ORAM implementations (Path ORAM achieved less than 1 MB/s throughput in our evaluation). Since most of the data is already placed inside memory for PIM computation, we expect the controlled memory accesses not to be a bottleneck in PIM workloads.

7.3 Protected Data-Intensive Computation Performance

To evaluate the performance of SE-PIM on real-world workloads, we implement a confidential k-mean clustering application powered with SE-PIM and compare its performance with an implementation on the baseline PIM. k-mean clustering is a standard algorithm in data analytics that aggregates data points into clusters. The application requires a large amount of data movement, making it a prime candidate for PIM-based accelerator [71], [72]. In our confidential k-mean clustering, the adversary observing the computation offloaded to SE-PIM learns no information. Moreover, our system allows the host enclaves to process data larger than the memory limitation of SGX-based enclaves (96 MB). When 8 SE-PIM-enabled memory banks are used, up to 512 MB of data can be processed concurrently. As more SE-PIM units are employed, more data can be processed.

Results. Fig. 11 demonstrates the performance of executing a round of k-mean on SE-PIM in comparison to executing the same algorithm on the `Baseline-PIM`. We also measure the algorithm's performance on the CPU (without PIM assistance) on similar data sizes for comparison. The specification of the CPU is described in Table 3. The total runtime of SE-PIM's confidential k-mean on encrypted data has negligible overhead to the baseline model without encryption. The results also show that as the data set sizes increase, the overhead of SE-PIM over `Baseline-PIM` also decreases. On average, the execution time of the confidential k-mean algorithm incurs 0.075%, 0.079%, 0.09%, 0.11% overhead compared to the insecure version when 1, 2, 4, and 8 PIM cores are used, respectively. Moreover, our system attains speed up in execution time compared to performing k-mean only with the CPU. The CPU-only computation on 8 CPU cores does not scale well to the increasing data set sizes due to the saturated memory bandwidth. In contrast, the SE-PIM cores benefit from a much higher bandwidth within a memory module. SE-PIM outperforms the CPU-only k-mean when more than 4 PIM cores are used. When processing 512 MB of data with 8 PIM cores, SE-PIM achieves 3.61× speed up to just using the CPU, with the execution time of 17.6 seconds (0.05% more than the baseline-PIM version). Note that the application's execution time on the CPU is optimistic in our evaluation. We do not consider the overhead of EPC swapping due to extensive memory usage, which can incur up to 1000× performance degradation in some applications [13], [73].

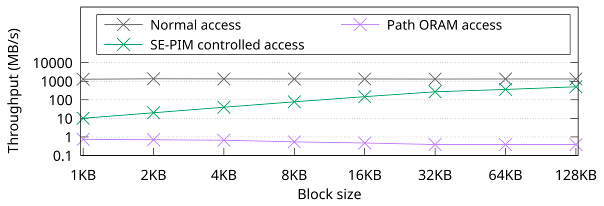


Fig. 10. Throughput of the controlled memory access channel on different block sizes in log-scale, compared against normal DRAM accesses and Path ORAM accesses.

8 Security Analysis and Discussion

In this section, we provide an in-depth security analysis of SE-PIM in a more formal manner.

8.1 Security Analysis

We revisit the security guarantees of SE-PIM through a point-by-point assessment of the SE-PIM's security objectives. The analysis is structured as the following: For each *Objective N*, we discuss the supporting *Properties N.M* from our design that directly contributes to achieving the objective.

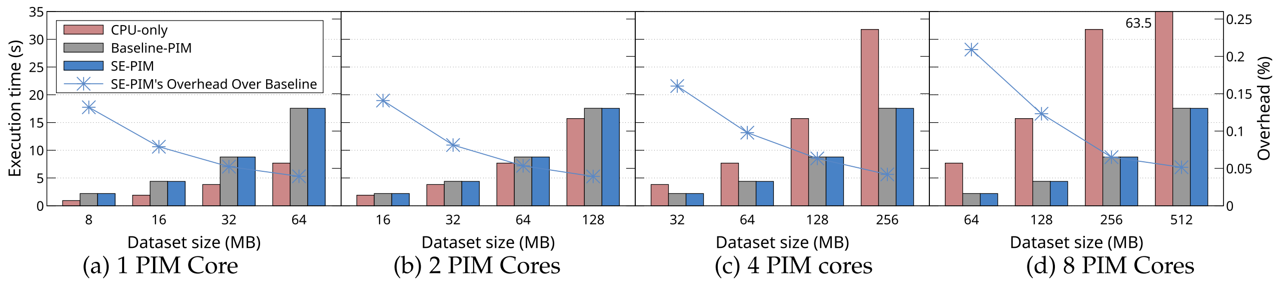


Fig. 11. The execution time of confidential k-mean using SE-PIM against the baseline PIM and CPU-only versions (8 CPU cores, without data encryption). SE-PIM overhead over the baseline PIM and the performance numbers of using different numbers of PIM cores are measured.

Objective 1. *The communication channel establishment between SE-PIM unit and host CPU enclave is secure.*

Property 1.1. *The endorsement key K_E is only known to the SE-PIM core.*

The security of remote attestation and key exchange K_E depends on the fact that the root endorsement key (K_E) is never leaked outside the memory bank package. A new attestation key pair is derived using the K_E at each invocation of `GET_CERT`. Moreover, only a very limited number of instructions in the attestation and key exchange procedures interact with K_E as described in Section 6.1.

Property 1.2. *Attestation and key exchange procedures are immutable.*

ROM in each SE-PIM unit ensures the immutability of the procedures for attestation and key exchange. During secure communication channel establishment, the procedures are copied into PIM local memory for execution. The attestation procedure itself measures (i.e., through hashing) the PIM local memory contents and presents it to the host enclave. This way, the host enclave can verify that the initial state of the PIM-side execution is trustworthy by comparing the provided hash against a known-good hash value.

Objective 2. *Communication between SE-PIM unit and host CPU enclave leaks no information about the exchanged messages.*

Property 2.1. *All commands and parameters are delivered to secure channel with fixed MMIO addresses.*

SE-PIM ensures that all communication messages have a fixed size, are encrypted and delivered to fixed MMIO addresses. This means that the communication between a SE-PIM unit and host CPU enclave always has the identical destination address throughout the session. Our experiment results presented in Section 7.1 precisely confirms this security property.

Property 2.2. *Commands and parameters are encrypted in indistinguishable way using AES-GCM encryption.*

The *data* part of the bus packet P that contains commands and parameters is encrypted with AES-GCM (counter mode). Therefore, each encrypted messages, even when the contents are identical, are indistinguishable to the observers.

Objective 3. *Confidentiality and integrity of data* in stationary state *inside SE-PIM memory bank is guaranteed*

Property 3.1. *Protected data is always encrypted inside SE-PIM memory bank.*

Our design choice was to keep the data residing in the memory bank encrypted, even with the new capability to drop access to the memory bank. An adversary with physical access to the memory module may launch a cold boot attack to dump the memory contents in the memory bank, but would only obtain encrypted values.

Property 3.2. *The integrity of initial data is verifiable by the host enclave through bank memory measurement.*

After the host enclave transfers the initial data into SE-PIM's memory bank, SE-PIM's memory lockdown prevents further modification to in-memory data. A bank memory integrity measurement (the hash checksum) (Section 4.4.1) is performed and notified to the enclave. Then, the host enclave compares the memory measurement with the known checksum to ensure that data has not been tampered with.

Objective 4. *SE-PIM's controlled memory access leaks no information about the access.*

Property 4.1. *The whole controlled access request (Access A = $(addr, op, size, data)$) is encrypted.*

The controlled access channel uses the secure communication channel whose security we demonstrated in Objective 2, so it inherits all of its security properties. By encrypting every information related to the access, the accesses are indistinguishable from any adversaries that observe the bus packets.

Property 4.2. *SE-PIM's lockdown prevents all host software or hardware from directly accessing memory bank.*

The lockdown mechanism prevents the attackers from accessing the memory bank, preventing any information leakages from attacker looking for memory content changes.

Property 4.3. *Data transfer only occurs within the memory bank (i.e., no main bus transactions).*

The controlled access offloads the actual memory access to the PIM core to use the internal connection between PIM and memory that is assumed to be secure from probing. Therefore, no bus transaction is generated from controlled accesses.

Objective 5. *Confidentiality and integrity of data inside SE-PIM memory bank* during computation *is guaranteed.*

Property 5.1. *SE-PIM's lockdown prevents all host software or hardware from directly accessing memory bank.*

The execution of a SE-PIM kernel might perform memory updates that reveal its access pattern to attackers that scan the memory. DRAM lockdown prevents attackers from observing such information.

Property 5.2. *Data transfer only occurs within the memory bank.*

During computation, there is no bus packet between the host CPU and SE-PIM memory bank induced from the data computation, thanks to the inherent efficiency of PIM-based computation. The adversary observing the main bus can not obtain any information regarding the computation.

Property 5.3. *Computed data is only decrypted inside PIM local memory.*

SE-PIM's AES-enabled DMA encrypts data flow from PIM local memory to the bank and decrypts in the reverse direction. Hence, the plaintext form of the computed data can only reside in the PIM's local memory. As we stated in Section 3.3, we assume that the SE-PIM unit and memory bank hardware are secure; the adversary cannot physically probe the PIM's local memory tightly integrated with the silicon.

Summary. Our security analysis outlined the security objectives achieved through the design properties of SE-PIM. We revisit the challenges for confidential PIM-based computation (C1-C4) to conclude our security analysis. First, C1 (Performance overhead of encrypted data transfer in PIM) is addressed by SE-PIM's AES-capable DMA Engine and extensively evaluated through benchmarks (Section 7). C2 (Address side-channel on the bus) is addressed by the combination of Objective 2, and Objective 4. The two objectives eliminate the leakage of information in the PIM-CPU communication that is encrypted by Objective 1. Both objectives are achieved through using a secure fixed channel over the bus, whose effect is well illustrated in Fig. 8. Objective 3 and Objective 4 directly address C3 (Memory content change side-channel); DRAM lockdown prevents attackers from taking snapshots of the memory to infer the content changes, while Objective 4 provides exclusive memory access to the CPU enclave. Lastly, C4 (Splicing and replaying attacks) are addressed by Objective 3 and Objective 5 such that the data inside PIM memory banks are protected both when it is stationary (Objective 3) data and in-computation (Objective 5). In all, SE-PIM's design addresses all the challenges that we outlined in Section 3.4 and provides performant and *confidential* PIM-assisted computation when satisfied.

8.2 Discussion

Supporting Cross-Bank Communication. In our current design, memory banks inside SE-PIM-enabled memory modules do not communicate or share resources. A PIM core can only access its local memory and bank memory (i.e., it cannot access other bank memory). Hence, no hardware connections can be exploited to create side channels between PIM cores. Such strict separation prevents security risks from malicious cotenants who have access to one or more SE-PIM banks. Being the first to explore confidential computing inside PIMs, our design focuses on maintaining the confidentiality of the computation. We expect that secure communication between PIM banks can enable more flexible forms of computation inside PIM [27].

Rowhammer Attacks and Physical Side-Channels. Mitigation of Rowhammer and physical (e.g., electromagnatic) side-channel mitigation is largely a still open field, and many works are exposing new attacks and proposing mitigations [57], [74]. Rowhammer attack [56] exploits inadvertent *flipping* of bits in DRAM during frequent and repeated memory accesses. DRAM bank in SE-PIM hosts confidential data. However, SE-PIM maintains the in-DRAM data is always encrypted with AES-GCM, which includes data integrity. Therefore, we expect that the impact of Rowhammer would be limited on SE-PIM. Moreover, the recently manufactured DRAM modules are devoid of the Rowhammer issues. For this reason, future DRAM-based PIM hardware with SE-PIM is unlikely to be affected by Rowhammer. Electromagnetic [58] and power side-channels [59] stem from the distinguishable physical effects of a processor when processing different data types or performing different operations. Existing works have explored circuit-level defenses that normalize or randomize the physical effects of the computation [74]. We consider the integration of such defenses into SE-PIM as future work.

Limitations of Cycle-Accurate Simulation. Our simulation setup might differ from an actual hardware implementation in a few aspects. First, we assume the presence of processor enclaves in the host, as our simulator does not support simulation of enclaves such as SGX [7] or TrustZone [75]. For this reason, we did not consider the overhead from the use of host processor enclaves, although the exclusion of the overhead works against our favor. Second, we only simulate the functional correctness of the DRAM lockdown unit; it is rather challenging to accurately simulate the latency of such a low-level mechanism in a simulator or even FPGAs. We leave actual hardware implementation and verification of the DRAM lockdown unit as future work.

9 Conclusion

We presented SE-PIM, an architecture that enables confidential computing inside memory by retrofitting the processor-in-memory architecture. Through a careful security analysis of the new architecture, we proposed a set of non-intrusive yet imperative changes that guarantees the data's confidentiality and integrity computed inside PIM. We evaluated our design and a proof-of-concept application for our design using a cycle-accurate full-system simulation. Our evaluation shows that the encrypted data transfer in our design only incurs a 17.85% overhead in the maximum throughput inside memory compared to the unmodified PIM architecture that does not support encryption. We also evaluated a confidential k-mean program that runs on our architecture to accelerate data-intensive operations. We observed only 0.05% increase in total execution time on the most significant data size compared to offloading to the unprotected PIM architecture.

References

[1] Google Cloud, "Confidential computing," 2021. [Online]. Available: https://cloud.google.com/confidential-computing

[2] Microsoft Azure, "Azure confidential computing," 2021. [Online]. Available: https://azure.microsoft.com/en-us/solutions/confidential-compute/

[3] Amazon Web Services, "AWS nitro enclaves," 2021. [Online]. Available: https://aws.amazon.com/ec2/nitro/nitro-enclaves/

[4] F. Schuster et al., "VC3: Trustworthy data analytics in the cloud using SGX," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 38–54.

[5] T. Lee et al., "Occlumency: Privacy-preserving remote deep-learning inference using SGX," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, 2019, Art. no. 46.

[6] F. Shaon, M. Kantarcioglu, Z. Lin, and L. Khan, "SGX-BigMatrix: A practical encrypted data analytic framework with trusted processors," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1211–1228.

[7] F. McKeen et al., "Innovative instructions and software model for isolated execution," in *Proc. 2nd Int. Workshop Hardware Architect. Support Secur. Privacy*, 2013, Art. no. 10.

[8] V. Costan and S. Devadas, "Intel SGX explained," Cryptol. ePrint Arch., Report 2016/086, 2016. [Online]. Available: https://eprint.iacr.org/2016/086

[9] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on Intel SGX," in *Proc. 10th Eur. Workshop Syst. Secur.*, 2017, Art. no. 2.

[10] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *Proc. 11th USENIX Workshop Offensive Technol.*, 2017, Art. no. 11.

[11] D. Lee, D. Jung, I. T. Fang, C.-C. Tsai, and R. A. Popa, "An off-chip attack on hardware enclaves via the memory bus," in *Proc. 29th USENIX Conf. Secur. Symp.*, 2020, Art. no. 28.

[12] A. Ahmad, B. Joe, Y. Xiao, Y. Zhang, I. Shin, and B. Lee, "OBFUSCURO: A commodity obfuscation engine on Intel SGX," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019, pp. 1–14.

[13] S. Sasy, S. Gorbunov, and C. W. Fletcher, "ZeroTrace: Oblivious memory primitives from Intel SGX," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–14.

[14] A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing digital side-channels through obfuscated execution," in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 431–446.

[15] S. Aga and S. Narayanasamy, "InvisiMem: Smart memory defenses for memory bus side channel," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, 2017, pp. 94–106.

[16] A. Awad, Y. Wang, D. Shands, and Y. Solihin, "ObfusMem: A low-overhead access obfuscation for trusted memories," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, 2017, Art. no. 107119.

[17] H. Oh, A. Ahmad, S. Park, B. Lee, and Y. Paek, "TRUSTORE: Side-channel resistant storage for SGX using Intel hybrid CPU-FPGA," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 1903–1918.

[18] M. Gao, G. Ayers, and C. Kozyrakis, "Practical near-data processing for in-memory analytics frameworks," in *Proc. Int. Conf. Parallel Archit. Compilation*, 2015, pp. 113–124.

[19] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu, "Processing-in-memory: A workload-driven perspective," *IBM J. Res. Develop.*, vol. 63, no. 6, pp. 3:1–3:19, 2019.

[20] X. Tang, M. T. Kandemir, H. Zhao, M. Jung, and M. Karakoy, "Computing with near data," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 3, Dec. 2018, Art. no. 42.

[21] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Enabling practical processing in and near memory for data-intensive computing," in *Proc. 56th Annu. Des. Autom. Conf.*, 2019, pp. 1–4.

[22] A. Boroumand et al., "LazyPIM: An efficient cache coherence mechanism for processing-in-memory," *IEEE Comput. Archit. Lett.*, vol. 16, no. 1, pp. 46–50, Jan.–Jun. 2017.

[23] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 1–14, 2018.

[24] S. Newsroom, "Samsung develops industry's first high bandwidth memory with AI processing power," 2021. Accessed: Apr. 25, 2021. [Online]. Available: https://news.samsung.com/global/samsung-develops-industrys-first-high-bandwidth-memory-with-ai-processing-power

[25] Advanced Micro Devices, Inc, "AMD high bandwidth memory," 2021. Accessed: Apr. 25, 2021. [Online]. Available: https://www.amd.com/en/technologies/hbm

[26] F. Devaux, "The true processing in memory accelerator," in *Proc. IEEE Hot Chips 31 Symp.*, 2019, pp. 1–24.

[27] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," *SIGARCH Comput. Archit. News*, vol. 43, no. 3S, Jun. 2015, Art. no. 105117.

[28] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit.*, 2015, pp. 336–348.

[29] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "Design and evaluation of a processing-in-memory architecture for the smart memory cube," in *Proc. 29th Int. Conf. Archit. Comput. Syst.*, 2016, pp. 19–31.

[30] K. Hsieh et al., "Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation," in *Proc. IEEE 34th Int. Conf. Comput. Des.*, 2016, pp. 25–32.

[31] A. F. Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, "NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules," in *Proc. 21st IEEE Int. Symp. High Perform. Comput. Archit.*, 2015, pp. 283–295.

[32] P. Chi et al., "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit.*, 2016, pp. 27–39.

[33] H. A. Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, "Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2016, pp. 1–13.

[34] S. H. Pugsley et al., "Comparing implementations of near-data computing with in-memory MapReduce workloads," *IEEE Micro*, vol. 34, no. 4, pp. 44–52, Jul./Aug. 2014.

[35] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "GraphPIM: Enabling instruction-level PIM offloading in graph computing frameworks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2017, pp. 457–468.

[36] A. Shafiee, R. Balasubramonian, M. Tiwari, and F. Li, "Secure DIMM: Moving ORAM primitives closer to memory," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2018, pp. 428–440.

[37] A. Gundu, A. S. Ardestani, M. Shevgoor, and R. Balasubramonian, "A case for near data security," in *Proc. Workshop Near-Data Process.*, 2014, pp. 1–3.

[38] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Proc. IEEE Hot Chips 23 Symp.*, 2011, pp. 1–24.

[39] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "A modern primer on processing in memory," 2020, *arXiv: 2012.03112*.

[40] S. Lee et al., "Hardware architecture and software stack for PIM based on commercial dram technology: Industrial product," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit.*, 2021, pp. 43–56.

[41] J. Nider et al., "A case study of processing-in-memory in off-the-shelf systems," in *Proc. USENIX Annu. Tech. Conf.*, 2021, pp. 117–130.

[42] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on GPUs," in *Proc. 13th USENIX Symp. Operating Syst. Des. Implementation*, 2018, pp. 681–696.

[43] K. Kim et al., "Vessels: Efficient and scalable deep learning prediction on trusted processors," in *Proc. 11th ACM Symp. Cloud Comput.*, 2020, pp. 462–476.

[44] W. Wang et al., "Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, Art. no. 24212434.

[45] E. Stefanov et al., "Path ORAM: An extremely simple oblivious RAM protocol," *J. ACM*, vol. 65, no. 4, 2018, Art. no. 18.

[46] L. Ren et al., "Constants count: Practical improvements to oblivious RAM," in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 415–430.

[47] C. W. Fletcher, L. Ren, A. Kwon, M. van Dijk, and S. Devadas, "Freecursive ORAM: [Nearly] free recursion and integrity verification for position-based oblivious RAM," in *Proc. 20th Int. Conf. Architect. Support Program. Lang. Operating Syst.*, 2015, Art. no. 103116.

[48] T. Hunt et al., "Telekine: Secure computing with cloud GPUs," in *Proc. 17th USENIX Symp. Netw. Syst. Des. Implementation*, 2020, pp. 817–833.

[49] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, 2015, pp. 105–117.

[50] J. V. Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 1041–1056.

[51] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, "Processing-in-memory for energy-efficient neural network training: A heterogeneous approach," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit.*, 2018, pp. 655–668.

[52] B. Y. Cho, J. Jung, and M. Erez, "Accelerating bandwidth-bound deep learning inference with main-memory accelerators," 2020, *arXiv:2012.00158*.

[53] A. Boroumand et al., "CoNDA: Efficient cache coherence support for near-data accelerators," in *Proc. 46th Int. Symp. Comput. Archit.*, 2019, pp. 629–642.

[54] Intel Corporation, "Intel software guard extensions (Intel SGX)," 2015. [Online]. Available: https://www.intel.com/content/dam/develop/external/us/en/documents/332680–002-600685.pdf

[55] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "CacheZoom: How SGX amplifies the power of cache attacks," 2017, *arXiv: 1703.06986*.

[56] Y. Kim et al., "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit.*, 2014, pp. 361–372.

[57] O. Mutlu and J. S. Kim, "RowHammer: A retrospective," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 8, pp. 1555–1571, Aug. 2020.

[58] H. Yu, H. Ma, K. Yang, Y. Zhao, and Y. Jin, "DeepEM: Deep neural networks model recovery through EM side-channel information leakage," in *Proc. IEEE Int. Symp. Hardware Oriented Secur. Trust*, 2020, pp. 209–218.

[59] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I know what you see: Power side-channel attack on convolutional neural network accelerators," in *Proc. 34th Annu. Comput. Secur. Appl. Conf.*, 2018, pp. 393–406.

[60] T. M. John, S. K. Haider, H. Omar, and M. van Dijk, "Connecting the dots: Privacy leakage via write-access patterns to the main memory," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 2, pp. 436–442, Mar./Apr. 2020.

[61] NXP Semiconductor, "LPC540xx/LPC54S0xx user manual," 2019. [Online]. Available: https://www.zlgmcu.com/data/upload/file/Utilitymcu/LPC540xx-yhsc.pdf

[62] G. Ma and H. He, "Design and implementation of an advanced DMA controller on AMBA-based SoC," in *Proc. IEEE 8th Int. Conf. ASIC*, 2009, pp. 419–422.

[63] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "Using address independent seed encryption and Bonsai Merkle trees to make secure processors OS- and performance-friendly," in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2007, pp. 183–196.

[64] S. Na, S. Lee, Y. Kim, J. Park, and J. Huh, "Common counters: Compressed encryption counters for secure GPU memory," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2021, pp. 1–13.

[65] N. Binkert et al., "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[66] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik, "A minimalist approach to remote attestation," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2014, pp. 1–6.

[67] OpenCores.org, "AES," 2019. [Online]. Available: www.opencores.org/projects/tiny_aes

[68] Hybrid Memory Cube Consortium, "Hybrid memory cube specification 2.0," 2014. [Online]. Available: https://www.nuvation.com/sites/default/files/Nuvation-Engineering-Images/Articles/FPGAs-and-HMC/HMC-30G-VSR_HMCC_Specification.pdf

[69] Micron Technology, "DDR4 SDRAM," 2014. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr4/16gb_ddr4_sdram.pdf

[70] Z. Liu, I. Calciu, M. Herlihy, and O. Mutlu, "Concurrent data structures for near-memory computing," in *Proc. 29th ACM Symp. Parallelism Algorithms Archit.*, 2017, pp. 235–245.

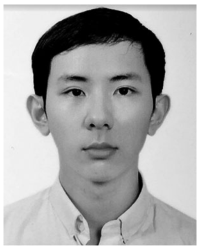
[71] M. Imani, S. Pampana, S. Gupta, M. Zhou, Y. Kim, and T. Rosing, "DUAL: Acceleration of clustering algorithms using digital-based processing in-memory," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2020, pp. 356–371.

[72] R. Kaplan, L. Yavits, and R. Ginosar, "PRINS: Processing-in-storage acceleration of machine learning," *IEEE Trans. Nanotechnol.*, vol. 17, no. 5, pp. 889–896, Sep. 2018.

[73] S. Arnautov et al., "SCONE: Secure Linux containers with Intel SGX," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 689–703.

[74] W. Yu, O. A. Uzun, and S. Köse, "Leveraging on-chip voltage regulators as a countermeasure against side-channel attacks," in *Proc. 52nd ACM/EDAC/IEEE Des. Autom. Conf.*, 2015, pp. 1–6.

[75] T. Alves, "TrustZone : Integrated hardware and software security," 2004.



Kha Dinh Duy received the BS degree in computer science from the Hochiminh University of Technology, in 2018. He is currently working toward the PhD degree with the Department of Computer Science and Engineering, Sungkyunkwan University, South Korea. His main research interests include trusted execution environments, hardware-assisted security, and program transformations for security.

Hojoon Lee received the BS degree from the University of Texas at Austin, and the PhD degree from KAIST, in 2018, advised by Prof. Brent Byunghoon Kang. He is currently an assistant professor with the Department of Computer Science and Engineering, Sungkyunkwan University since September 2019. Prior to his current position, he spent one year as a postdoctoral researcher with CISPA under the supervision of Prof. Michael Backes. His main research interests lie in retrofitting security in computing systems against today's advanced threats. His research interests include operating system security, trusted execution environments, program analysis, software security, and secure computation in the cloud.