IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 2 July 2024, accepted 15 July 2024, date of publication 19 July 2024, date of current version 15 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3431231

RESEARCH ARTICLE

TEEm: Supporting Large Memory for Trusted Applications in ARM TrustZone

JUN LI[1,2], XINMAN LUO[3], HONG LEI[1,4], AND JIEREN CHENG[5], (Member, IEEE)

[1]School of Cyberspace Security (School of Cryptology), Hainan University, Haikou 570228, China
[2]Oxford–Hainan Blockchain Research Institute, Chengmai 571924, China
[3]School of Information Science and Technology, Qiongtai Normal University, Haikou 571127, China
[4]SSC Holding Company Ltd., Chengmai 571924, China
[5]School of Computer Science and Technology, Hainan University, Haikou 570228, China

Corresponding author: Hong Lei (leihong@hainanu.edu.cn)

This work was supported in part by the China Computer Federation (CCF) - Huawei Populus Grove Fund, and in part by the Research Startup Fund of Hainan University under Grant KYQD(ZR)-21071.

ABSTRACT Trusted Execution Environments (TEEs), like ARM TrustZone, are increasingly crucial in fields like machine learning, blockchain, WebAssembly, and databases due to their robust security features. Despite their growing importance, TrustZone-based compact TEE operating systems such as OP-TEE are not equipped to support large memory for trusted applications. This is because TrustZone was primarily used in embedded and mobile devices, which typically do not require large memory capacities. However, this restriction is particularly critical as it limits TEEs' effectiveness in processing large-scale data and conducting memory-intensive computations. In this paper, we propose TEEm, a novel solution that enables large secure memory support in TEEs without compromising security. To the best of our knowledge, this is the first public method that supports large memory for Trusted Applications (TAs) to run directly within TrustZone. TEEm designs the single-to-multiple memory mapping policy to expand virtual address space for TA, and a parameter-based memory allocation mechanism that allows TAs to request more trusted memory from TEE. To validate the feasibility and performance of TEEm, we build a prototype based on OP-TEE and evaluate it using multiple memory micro-benchmarks. Security and performance evaluations demonstrate that TEEm not only achieves a performance of 3.48 times faster than Linux in memory allocation but also maintains a high level of security, providing substantial memory support for memory-intensive applications.

INDEX TERMS Large memory, trusted applications, TEE, ARM TrustZone.

I. INTRODUCTION

Trusted Execution Environments (TEEs) are a crucial secure technology designed to protect sensitive application data, attracting significant interest due to their strong data protection capabilities. Currently, Intel Software Guard Extensions (Intel SGX) [1], AMD Secure Encrypted Virtualization (SEV) [2], ARM TrustZone [3], and Keystone [4] are considered primary TEE technologies. Intel SGX2, with its capability to offer up to 1TB of secure memory, has been widely adopted into cloud platforms such as Amazon Web Services, Google Cloud, Microsoft Azure, and IBM Cloud [5]. Meanwhile, ARM TrustZone plays a crucial role in mobile and edge scenarios, which typically have limited memory resources. ARM TrustZone creates secure and isolated environments within the secure world, which is distinct from the normal world where regular applications run, specifically for the execution of various Trusted Applications (TAs) [6], [7], [8], [9], [10]. Although TrustZone is currently primarily deployed on mobile platforms, the increasing demand for cloud computing and new requirements for data center efficiency and computational performance reveal significant potential for ARM architecture in the server market [11]. For instance, the launch of Qualcomm's 48-core Centriq 2400 server chip signifies a significant advancement of ARM processors in competition with traditional x86 servers [12].

The associate editor coordinating the review of this manuscript and approving it for publication was Sedat Akleylek.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

The need to support larger memory in TEEs has become critical for applications requiring extensive data processing capabilities. Unfortunately, existing works [13], [14], [15], [16], [17] neglect the memory limitation of ARM TrustZone, lacking designs and implementations to expand large secure memory for TA.

In this paper, we focus on ARM platforms, as they increasingly play significant roles in the realms of mobile, edge, and cloud computing [18] and when we refer to the TEE, we are specifically referring to implementations based on ARM TrustZone technology. Open Portable Trusted Execution Environment (OP-TEE) [19] is the most commonly used Trusted Operating System (TOS) based on ARM TrustZone. Due to its open source, compliance with the GlobalPlatform (GP) TEE standard, and compatibility with a variety of ARM processors, researchers can easily access the source code and modify or extend it according to their research needs. Crucially, the architecture of ARM TrustZone uses a three-level page table structure for address translation, which includes Level 1 (L1), Level 2 (L2), and Level 3 (L3) [20]. The L1 table is the top-level table, dividing the virtual address space into large regions, each described by an L1 table entry. The L2 table is the middle-level table, subdividing the large regions from L1 into smaller regions, each described by an L2 table entry. The L3 table is the bottom-level table, subdividing the small regions from L2 into pages, typically 4KB, 16KB, or 64KB in size, and L3 table entries directly point to physical memory pages, completing the final address translation. This structured approach allows TrustZone to manage memory securely and efficiently, addressing both large-scale and minute security-sensitive operations.

Although TEEs offer numerous benefits and are increasingly utilized in various emerging scenarios, devices equipped with TEEs often struggle with supporting large memory within TAs. This limitation is largely a consequence of constrained resources and configuration restrictions. As a result, TEEs are not well-suited for computationally intensive tasks that require extensive memory resources. To address these limitations, in this paper, we propose TEEm, a design that enables secure and efficient large trusted memory support for TAs in ARM TrustZone. The primary idea of this work is to design the Single-to-Multiple (S2M) memory mapping policy to adjust the number of L2 page tables for each TA, thereby expanding the virtual memory space. Additionally, a Parameter-based Memory Allocation (PbMA) mechanism is introduced, which allocates memory according to the specific resource requirements of each TA. By implementing these strategies within the OP-TEE framework, TEEm addresses the limitations of existing TEE in supporting large memory applications.

We implemented the prototype of TEEm on a TrustZone-enabled device and conducted a security analysis and performance evaluation. The evaluation results show that TEEm's memory allocation performance is, on average, 3.48 times faster than that of Linux, while maintaining a high level of security.

In summary, the main contributions of this paper include:

- A comprehensive analysis of the TEE memory management mechanism and uncover the key factors that limit the available memory capacity for TAs.
- A design of TEEm that can securely support large memory TAs in ARM TrustZone. Firstly, we introduce the S2M memory mapping policy that allows TAs to expand and use larger virtual memory space. Secondly, we design a PbMA mechanism that adjusts the memory allocation for TAs based on their specific resource requirements.
- An implementation of TEEm on real devices and evaluation with memory micro-benchmarks. The evaluation results indicate that the performance overhead incurred by memory expansion is negligible.

The remainder of this paper is organized as follows: Section II presents related work of TEEm. Section III introduces the required background information. Section IV presents the design and implementation of TEEm. Section V details the performance of TEEm. Then, Section VI provides a discussion of the results. Section VII details the security analysis. Finally, Section VIII concludes this paper.

II. RELATED WORK

In this section, we delve into research related to ARM TrustZone, focusing on the design of TEEs based on TrustZone and the implementation of secure solutions using TrustZone in various domains.

A. DESIGN AND OPTIMIZATION FOR ARM TRUSTZONE

We survey related systems on secure design and optimization in TEEs, specifically focusing on ARM TrustZone. SecTEE [13] is a software-based secure isolation architecture that does not rely on specific hardware security features in the ARM CPU. It offers robust security attributes such as integrity measurement, remote attestation, and data sealing, providing a high level of security comparable to hardware-based secure isolation architectures. However, its hardware independence may lead to lower performance compared to hardware-enhanced solutions. TEEp [14] introduces a novel multi-threading mechanism within TEE, aiming to enable multithreaded applications to run directly within the TEE. Despite its advantages, the introduction of multi-threading support increases the complexity of the TEE system, potentially introducing new security vulnerabilities. LEAP [15] offers a lightweight TEE solution allowing parallel code execution, convenient access to peripheral devices, flexible resource management, and automated DevOps tools for code preparation. While it provides flexible resource management, its efficiency may be limited in high-load or complex applications. CRONUS [16] divides heterogeneous computing into independent TEE enclaves, each encapsulating specific computation (such as GPU), enabling spatial sharing of multiple enclaves on a single accelerator. However, the complexity and hardware dependence of CRONUS may

TABLE 1. Comparison of TEEm with existing works.

Publication	Characteristics	Security Features	Hardware Dependency	Applicable Scenarios	Large Memory Support
Smaug [6]	Secure SQLite database	ARM TrustZone, TPM	Yes	Databases	No
WaTZ [7]	Secure Wasm runtime	ARM TrustZone	No	WebAssembly	No
TSC-VEE [8], MECAT [21]	Smart contract execution environment	ARM TrustZone	No	Smart contracts	No
DarkneTZ [9], Trusted-DNN [23], T-Slices [24]	DNN model privacy protection	ARM TrustZone	No	Deep learning	No
OMG [10]	Secure offline ML	ARM TrustZone	No	Mobile devices	No
SecTEE [13]	Software-based secure isolation	ARM TrustZone	No	General	No
TEEp [14]	Multi-threading	ARM TrustZone	No	Multi-core environments	No
LEAP [15]	Lightweight TEE	ARM TrustZone	No	Intelligent mobile applications	No
CRONUS [16]	Heterogeneous computing TEE encapsulation	ARM TrustZone, GPU	Yes	Heterogeneous computing	No
RusTEE [17]	Rust-enhanced TA security	ARM TrustZone	No	General	No
Xie et al. [26]	Memory management	ARM TrustZone	No	Deep learning	No
SOTPM [28]	Lightweight shared memory protection	ARM TrustZone	No	Secure data exchange	No
GateKeeper [29]	Operator-centric TA management	ARM TrustZone	No	IoT	No
TEEm (Our work)	**Memory expansion**	**ARM TrustZone**	**No**	**General**	**Yes**

limit its applicability in other environments, and its spatial sharing mechanism requires strict isolation policies to prevent resource contention and security issues. RusTEE [17] leverages Rust, a memory-safe language, to enhance the security of TAs. This approach effectively prevents memory corruption vulnerabilities but relies on the limited support of Rust for low-level operations, potentially introducing security risks through the use of unsafe code. Additionally, migrating existing C to Rust involves significant engineering effort.

Although the majority of research on TEE design and optimization has focused on enhancing security and performance, a critical but largely unexplored problem is how to support large memory capacity within TAs. Existing designs do not adequately meet the needs of memory-intensive applications.

B. SECURE TRUSTED APPLICATIONS

Many research projects have proposed various secure applications based on ARM TrustZone. Smaug [6] proposes a general security solution based on TEE and Trusted Platform Module (TPM) to ensure the confidentiality and integrity of databases like SQLite. However, integrating TPM and TEE security mechanisms may introduce performance overhead, especially in high-frequency database operations. WATZ [7] designs an efficient secure runtime for WebAssembly based on ARM TrustZone and offers a lightweight remote attestation mechanism optimized for Wasm applications. However, its reliance on remote attestation, while enhancing security, may impact performance. TSC-VEE [8] designs a virtual execution environment to support the execution of smart contracts programmed by Solidity language. Implementing the Ethereum Virtual Machine (EVM) and optimization techniques on TrustZone significantly enhances execution performance. However, this approach increases system complexity and depends on the EVM implementation, which may introduce security risks. In contrast, MECAT [21] uses Rust to write smart contracts and execute them in the secure world of TrustZone. This approach avoids memory safety issues while maintaining system simplicity. OMG [10], based on SANCTUARY's [22] work, implements hardware-enforced isolation using TEE on ARM platforms, providing protection for entire machine learning models and guaranteeing the privacy of customer data privacy, the confidentiality of models, and the integrity of algorithms. However, the implementation of secure offline machine learning on mobile devices may introduce performance overhead, especially in high-frequency operations. Previous research on using TEEs to protect Deep Neural Network (DNN) models has primarily focused on the design of different methods to optimize resource requirements. For instance, DarkneTZ [9], Trusted-DNN [23], and T-Slices [24] have developed distinct partitioning techniques for the DNN models, as well as other strategies to protect sensitive model components from being leaked, such as SRFL [25]. While these approaches offer strong privacy protection, model partitioning may introduce performance overhead and compatibility issues. Xie et al. [26] have improved DNN inference performance on resource-constrained devices through dynamic adjustment of memory priority and optimization of small libraries, but they do not support large memory applications. Additionally, the Penetralium framework [27] focuses on privacy-preserving and memory-efficient neural network inference at the edge, integrating privacy-preserving techniques with memory optimization. SOTPM [28] presents a lightweight and secure scheme for shared memory in TEEs. While it effectively

addresses the vulnerabilities associated with shared memory, its approach of making memory read-only after writing may limit flexibility in some applications. GateKeeper [29] addresses the obstacle of deploying in-house security systems in ARM TrustZone.

Although these studies provide valuable insights and techniques for enhancing the security and functionality of TEEs, they primarily concentrate on designing various TAs within the memory-constrained environment of ARM TrustZone. Our work focuses on overcoming the memory limitations traditionally associated with TEE, thereby expanding their applicability. Table 1 provides a comparison of these related works based on their key contributions, security mechanisms, hardware dependencies, applicable scenarios, and support for large memory.

III. BACKGROUND

A. TRUSTED EXECUTION ENVIRONMENT

TEE has emerged as a promising solution to protect the confidentiality and integrity of code and data from various attacks [11]. ARM TrustZone, first introduced in 2004, is a comprehensive trusted computing solution integrated into ARM processor architectures. It establishes hardware-based isolation by dividing physical system resources into secure and normal worlds [21], [30]. The secure world runs a trusted OS like OP-TEE and provides a trusted environment for TAs. It has isolated memory, peripherals, and CPU modes from the Rich Operating System (ROS) in the normal world [31]. The hardware-based partitioning in ARM TrustZone establishes a foundation of trust, rooted in the CPU's built-in TrustZone features. TrustZone has become widely deployed across billions of devices like mobile phones, embedded systems, Internet of Things (IoT), and increasingly servers through ARM server chips. By providing a hardware-backed trusted foundation, TrustZone enables use cases such as secure payments, intellectual property protection, authentication, and more across the spectrum of smart connected devices [32].

Our work targets OP-TEE with native support for ARM TrustZone. OP-TEE is an open-source trusted OS that complies with the GP standard. It is the only TEE kernel integrated with Linux and ARM Trusted Firmware (ATF) [33]. OP-TEE follows the GP API [34] specifications to enable communication between the Rich Execution Environment (REE) and TEE through client applications, Linux drivers, and the OP-TEE OS.

B. MEMORY MANAGEMENT

Memory management in OP-TEE revolves around the fundamental principles of security, isolation, and efficient resource utilization. Within the secure world, each TA operates within its isolated memory space, protected by hardware features like the Memory Management Unit (MMU) and TrustZone technology. Two translation table base registers, TTBR0_EL1 and TTBR1_EL1, are used to store the base addresses of the page tables for user space and kernel space, respectively. TTBR0_EL1 is responsible for translating the virtual address space in user mode, while TTBR1_EL1 is responsible for translating the virtual address space in kernel mode. The processor selects the appropriate page table based on its current privilege level.

In OP-TEE, the trusted kernel allocates a contiguous physical memory region during startup, known as the TEE memory pool. For TAs, OP-TEE uses a simple linear memory allocation mechanism, where each TA partitions its own virtual memory space from the physical TEE memory pool. When a TA is invoked, OP-TEE first allocates memory and loads the TA's binary file (code and data) into secure memory. Then, OP-TEE allocates memory for the TA's execution context. This includes memory for the stack, heap, and any other data structures required for the execution of the TA. The TA heap is explicitly defined as a static buffer residing in the .bss section. The configuration of this buffer is determined by the TA_DATA_SIZE parameter during the TA completion process. OP-TEE also sets up page tables for newly allocated memory regions. These page tables are used to map the virtual addresses used by the TA to physical addresses in secure memory. Finally, during the initialization process, the TA calls the init_instance and malloc_add_pool interface, thereby adding ta_heap to the memory pool managed by bget. When running a TA, the allocation and release of memory buffers within the TA are handled by the OP-TEE kernel using malloc-like APIs. Specifically, when the TA calls the malloc interface to perform memory allocation, it sequentially calls a series of functions such as raw_malloc, raw_memalign, and bget, ultimately obtaining memory space from the TEE memory pool.

C. THE NEED FOR LARGE MEMORY SUPPORT IN TEE

In the rapidly evolving landscape of IoT, Artificial Intelligence (AI), and big data, TEE, exemplified by ARM TrustZone, faces new challenges. Initially designed for mobile and embedded devices, TrustZone is also increasingly recognized for its strong security features in cloud computing. However, the current support for large memory TAs within TEE OS is limited, making it difficult to meet the real-world needs of modern applications for large-scale data processing. This gap significantly hinders the expansion and potential of TEE technology in emerging fields. Expanding memory capabilities within TEEs is vital for several reasons. Firstly, it enables the efficient handling of large-scale, memory-sensitive computational tasks. Secondly, enhanced memory support can drastically reduce the overhead associated with data transition between secure and normal worlds. Moreover, large memory support can foster innovation in TEE applications by allowing developers to explore more complex and memory-intensive applications without being constrained by memory limitations.

D. GOALS

The design of TEEm has the following three goals:

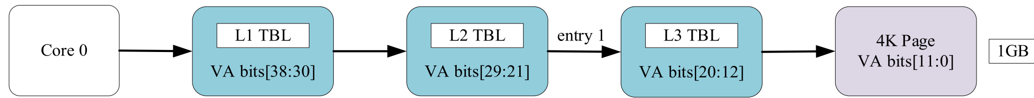

FIGURE 1. Constraints of the TEE OS memory page table mapping.

1) PERFORMANCE

Support for an expanded TA address space and contiguous buffer allocation enhances allocation efficiency and meets the demands for higher memory bandwidth.

2) SMALL TRUSTED COMPUTING BASE (TCB) SIZE

The system reutilizes existing memory management mechanisms, with minimal expansion to kernel memory-related data structures, such as page tables and memory descriptors.

3) COMPATIBILITY

Maximizing compatibility with existing TAs, the system does not require modifications to the current TEE programming model. While introducing new features, it preserves backward compatibility with legacy interfaces.

IV. DESIGN AND IMPLEMENTATION

In this section, we introduce TEEm. First, we explain the design of TEEm. Then, in contrast to previous efforts that primarily focused on optimizing individual modules within TAs to accommodate limited memory, TEEm designs a novel S2M memory mapping policy to extend the scope of a single TA's page table mapping. Furthermore, PbMA is provided to address the need for large memory configurations.

A. DESIGN OVERVIEW

In OP-TEE, TA runs in a limited memory environment, typically with a few megabytes of memory. This limitation often hinders the execution of complex tasks requiring substantial memory and requires the user to effectively manage memory resources to ensure that TAs do not run into memory shortages during execution. Our goal is to increase memory allocation and usage within TA while ensuring that the security and isolation principles of TEE remain unchanged. In OP-TEE, the memory expansion for TA depends on two critical factors: the available virtual address space and physical memory. Firstly, TA operates within their own isolated virtual address spaces, managed through a multi-level page table structure. This structure includes L1, L2, and L3 page tables. The L1 page table, as the top level, divides the TA's entire virtual address space into large blocks, with each entry corresponding to an L2 page table. These L2 tables further subdivide these blocks into smaller sections, each pointing to an L3 page table. The MMU translates virtual addresses accessed by a TA into physical addresses using this hierarchy of page tables, ensuring memory isolation and security. Secondly, the TA needs to request the TEE OS to allocate memory resources. When compiling a TA, the TA_DATA_SIZE defines the size of a global array called ta_heap, which is located in the TA's .bss segment. Upon loading a TA, the TEE OS allocates and maps memory for this segment, guided by the TA_DATA_SIZE. Besides, there are several core parameters used to configure different types of memory areas in OP-TEE. CFG_TZDRAM_START and CFG_TZDRAM_SIZE define the starting address and total size of the physical memory pool for the secure world, respectively. This memory area is managed and allocated by OP-TEE OS to various secure world components. CFG_TA_RAM_START and CFG_TA_RAM_SIZE define the runtime memory area specifically for TA, which is partitioned from the pre-provisioned secure memory pool. OP-TEE relies on these key parameters to configure the secure physical memory pool from the system memory; then, it allocates different types of logical memory areas from the secure pool, such as runtime space for TA, file system cache, etc. The configuration of these memory parameters directly affects the memory capacity that OP-TEE can manage and allocate.

Figure 1 illustrates the configuration of page tables for TA. Each entry in the L1 page table points to an L2 page table, each entry in the L2 page table points to an L3 page table, and each entry in the L3 page table directly maps a 4KB memory block. This mechanism of allocating only one L2 page table to TA limits its ability to map more than 1GB of virtual address space. Therefore, expanding the memory of TA fundamentally involves enlarging its virtual address space and requesting more memory from the TEE OS. We address these challenges by introducing a novel S2M memory mapping policy and PbMA mechanism. Figure 2 shows the architecture and interaction steps of TEEm. The creation of the large memory TA is initiated by the Client Application (CA) by calling TEEC_OpenSession in libteec. S2M mapping policy expands the virtual address space available to a single TA by allowing it to allocate and use multiple L2 page tables. PbMA enables dynamic and flexible adjustment of TA memory allocation by allowing users to modify heap size parameters according to specific application requirements.

B. SINGLE-TO-MULTIPLE MEMORY MAPPING POLICY

In OP-TEE, each TA thread is bound to a single L2 page table, and the TA's context is scheduled to a particular thread before entering user space, thereby associating with the L2 page table bound to that thread. However, each thread is only bound to a single L2 page table, which limits the virtual address space of each TA to 1GB. Therefore, we designed the S2M memory mapping policy, which allows each TA thread

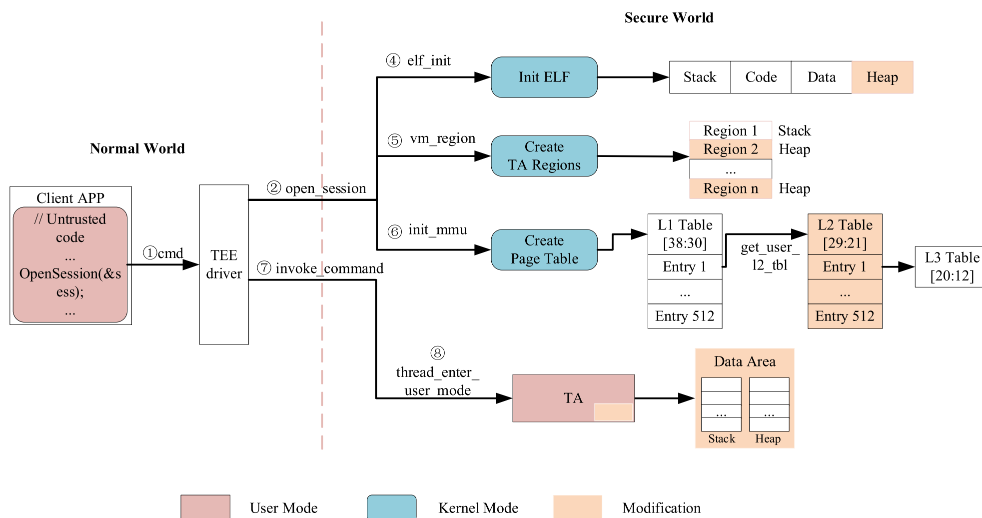

FIGURE 2. **Overall TEEm architecture.**

to manage multiple L2 page tables, thereby expanding the available virtual address space for TAs. Figure 3 illustrates the S2M design scheme.

Firstly, we use the user_va_idx to mark the starting position of the free L1 page table entries allocated for the TA, simplifying the process of locating L1 entries to map user virtual addresses. Then, we define the CFG_NUM_TA_L2_TBLS parameter. This parameter has a dual function: it determines the maximum number of L2 page tables a single TA can manage, and it also acts as a protective measure to prevent a TA from allocating too many L2 page tables. Then, we modified how a TA stores L2 page tables, associating the thread of TA with CFG_NUM_TA_L2_TBLS, thereby allowing a single TA thread to manage more L2 page tables. We add the core_mmu_get_user_l2_tbl_va to pinpoint the exact location of L2 page tables within the virtual address space. Finally, we restructure key data structures and functions related to the user MMU to support L2 page table arrays, allowing each user address space to be associated with multiple L2 page tables. Given that the L1 page table has 512 entries, theoretically, if each entry is fully utilized, S2M can expand the virtual address space of TA up to 512GB. S2M enables TA to manage and utilize larger memory resources more efficiently by increasing the virtual address space accessible to TA.

C. PARAMETER-BASED MEMORY ALLOCATION

S2M addresses the limitation that restricted a TA to mapping only a maximum of 1GB of virtual address space. However, the memory a TA can utilize still depends on the TA_DATA_SIZE parameter. In OP-TEE, memory allocation for a TA based on TA_DATA_SIZE involves three stages: compilation, loading, and initialization. During the compilation stage, the compiler creates a global memory buffer array named ta_heap, whose size is determined by TA_DATA_SIZE. The ta_heap is placed in the .bss segment

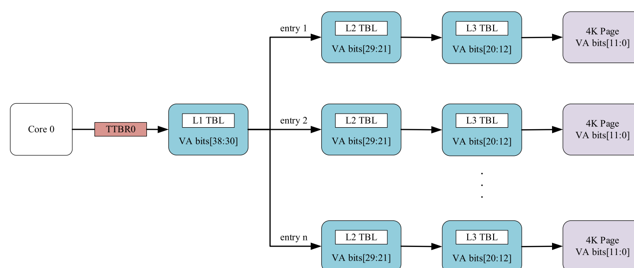

FIGURE 3. **The design of S2M memory mapping policy.**

of the TA's executable file. Subsequently, when the TEE OS loads the TA, the TEE kernel allocates and maps memory space for various ELF segments using an interface similar to mmap. This process assigns a virtual address to the ta_heap array. Finally, during the initialization process of the TA, the initialization code passes the start address and length information of the ta_heap array to the memory allocator function in the user-space libc library. Consequently, the memory allocator adds this memory space to the heap memory pool it manages, making it available for dynamic allocation during the TA's runtime. However, the compiler restricts variables within the TA executable file, like the ta_heap static array, from spanning a 4G address space. To address this, we considered two options: optimizing the compiler to remove this restriction or design an alternative method.

We design the PbMA mechanism, introducing a large memory configuration parameter, TA_DATA_GIBIBYTE. This parameter is not used to configure the static array ta_heap at the compile stage, but rather it is used to apply for memory via a syscall interface during the TA initialization. This method was chosen over modifying the compiler for several compelling reasons. It offers enhanced flexibility and scalability to meet the diverse memory requirements of different TAs. By not altering the compiler, we maintain

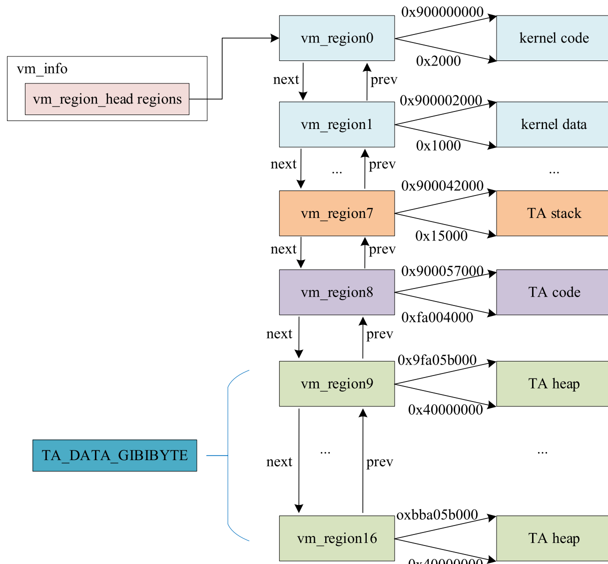

FIGURE 4. Overview of PbMA workflow.

compatibility and simplify maintenance, avoiding the complexities associated with such changes. Additionally, this approach ensures more extensive applicability across various platforms. In the PbMA, as illustrated in Figure 4, the TA_DATA_GIBIBYTE parameter plays a crucial role. It allows users to modify the memory size of TA according to their specific application requirements. The workflow begins with the vm_info structure, each vm_region represents a specific area of memory, with vm_region0 through vm_region8 pre-allocated for essential components like kernel code, kernel data, and the TA's stack and code. The PbMA process is highlighted by the presence of vm_region9 through vm_region16, demonstrating the scalability of the TA's heap. When PbMA is engaged during TA initialization, the TA_DATA_GIBIBYTE parameter is used to request memory blocks through the tee_map_zi interface. Each re-quested block is efficiently mapped into the TA's virtual address space, significantly expanding its capacity. The next and prev pointers in each vm_region ensure the proper linkage and order within this linked list structure, maintaining a coherent allocation sequence. In this case, TEEm can use this new parameter to dynamically expand TA's memory.

D. TIME COMPLEXITY

1) S2M

In OP-TEE, each TA is associated with only one L2 page table. This means that memory management for a TA is very straightforward, with a time complexity of $O(1)$ because memory mapping operations only need to traverse a fixed page table path. In the S2M, however, a TA can be associated with multiple L2 page tables. This increases the complexity of management, especially when determining which L2 page table to use for address mapping. This design significantly expands the virtual address space but also involves additional complexity in page table traversal and update operations. In theory, if a TA manages n L2 page tables, the time complexity for finding the correct L2 page table for mapping in the worst case could increase to $O(n)$.

2) PbMA

When a TA dynamically requests memory using the TA_DATA_GIBIBYTE parameter, it requires system calls to allocate a specified size of the memory region dynamically. The process determines the memory size to be allocated based on the parameter value and then performs mapping and permission configuration operations. Assuming that each additional gigabyte of memory requires extra initialization and configuration time, this time is essentially proportional to the memory size requested, resulting in linear growth. Therefore, we can assume that the time complexity of the PbMA mechanism is $O(n)$, where n represents the memory size requested. This linear complexity is consistent with standard memory allocation mechanisms in operating systems like Linux, where the allocation time is also proportional to the size of the memory requested. Thus, PbMA does not introduce additional overhead compared to existing systems.

Although the time complexity of both S2M and PbMA is $O(n)$, they provide efficient support and management of large memory requirements. Through experiment analysis and verification, we demonstrate that this complexity is reasonable and acceptable when dealing with large-scale memory allocation.

V. PERFORMANCE EVALUATION

We implemented a prototype of TEEm based on OP-TEE, which is an open-source secure TEE framework designed to run on ARM processors. It implements the GP standardized API between the REE and TEE. In this section, we describe the experiment setup (including the platform and evaluation methods) and evaluate the functionality and performance of the prototype.

A. EXPERIMENT SETUP

On the hardware side, we implement and deploy TEEm on a desktop computer equipped with 8 FTC663 cores and 64GB of physical memory. We configured 32GB of memory each for the REE and TEE. On the software side, the REE OS was Kylin V10 with kernel version 4.19.91. All code in REE and TEE was written in C and compiled with GCC 7.5.0. All evaluations were performed in single-thread mode. TEEm only modifies the memory management in the TEE kernel with about 200 lines of code, while still adopting the original communication model between the REE and TEE. In addition to the aforementioned setup, the configuration of the TEE OS memory played a crucial role in our experiments. Specifically, CFG_TZDRAM_START was set at 0xC0000000 with CFG_TZDRAM_SIZE at 0×40000000. For the TA memory, CFG_TA_RAM_START was configured at 0×3000000000 and CFG_TA_RAM_SIZE at 0×0800000000. Additionally, we set the maximum memory

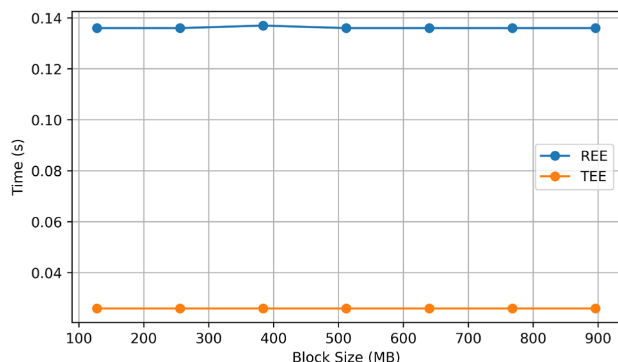

FIGURE 5. Memory sequential read/write test.

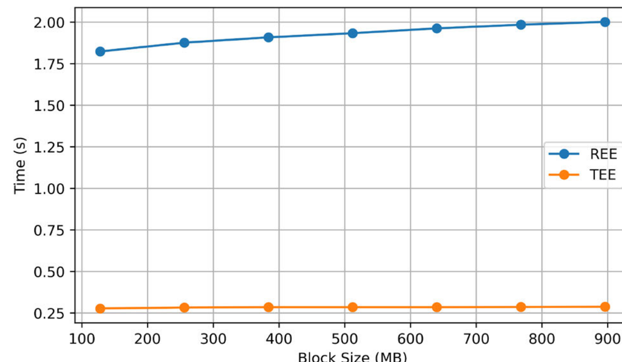

FIGURE 6. Memory random read/write test.

parameter that a TA can request, TA_DATA_GIBIBYTE, to 32. This configuration of memory parameters was essential to ensure that the expanded TA memory within TEE was capable of handling the required workload efficiently.

B. MEMORY ACCESS PATTERN

We first investigate the impact of memory access patterns on performance. We selected varying data block sizes within a 1GB memory and conducted experiments encompassing two primary access patterns: random access and sequential access. For each block size, 10 million memory access operations were executed, and the tests were repeated 10 times to ensure statistical rigor. Figure 5 revealed that the REE has consistent access times across all block sizes, maintaining around 0.136s. In contrast, the TEE demonstrates significantly faster access times at 0.026s, regardless of the block size. On the other hand, the random access results from Figure 6 reveal a different pattern. The REE shows a gradual increase in access times as the block size increases, starting from 1.8s for the 128MB block size and going up to 2.0s for the 896MB block size. This gradual increase suggests a degradation in performance as the demand on the memory system grows, which is typical due to the overhead associated with seeking random memory locations. The TEE, however, maintains remarkably lower access times than the REE, only slightly increasing from 0.28s to 0.29s as block sizes increase. The TEE's relatively flat and low response times indicate robustness against the block size increase, implying more efficient handling of random memory accesses that could be attributed to a more responsive memory allocation strategy.

C. MEMORY BANDWIDTH

We assess the memory bandwidth of TEEm using two recognized microbenchmark tools, mbw [35] and stream [36], each offering unique insights into the performance dynamics under various operational scenarios. The mbw tool, which stands for Memory Bandwidth, provides a comprehensive evaluation through diverse memory copy methods. These methods are designed to simulate different types of memory-intensive tasks that a TA might need to handle, thereby showcasing the effective bandwidth available within both the REE and the TEE. In Figure 7, mbw provided a multi-faceted memory bandwidth test through different copy methods, while stream conducted standardized memory operations (such as COPY, SCALE, ADD, and TRIAD), offering an industry-standard memory performance assessment. All tests were conducted in a single thread, with each test performed 10 times. We tested the performance of three methods (MEMCPY, DUMB, and MCBLOCK) within REE and TEE across different memory sizes (128MB, 256MB, 512MB, 1024MB, and 2048MB). In the MEMCPY method, the performance range in REE was from 3693.57MB/s to 3704.83MB/s, while in TEE, it ranged from 3667.62MB/s to 3710.15MB/s. This indicates that TEE's performance in handling continuous memory copy operations is comparable to REE. For the MCBLOCK method, both REE and TEE maintained high and close performances, demonstrating superior processing capabilities in both environments for this method. In the DUMB method, TEE's performance (reaching up to 3210.92MB/s) was significantly better than REE (with a maximum of 1165.38MB/s), especially at larger memory allocations. This could be due to the reduced contention for memory operations or threads within the secure isolated environment, leading to a more efficient memory access.

During the evaluation of the expanded TA memory in TEEm using the stream tool, we compared the performance of REE and TEE under different memory requirements (with individual array memories of 128MB, 256MB, 512MB, and 1024MB, and total test memories of 384MB, 768MB, 1536MB, and 3072MB, respectively). As can be seen from Figure 8, overall, REE slightly outperforms TEE in the tests, but TEE still maintains high performance close to REE after memory expansion, especially when handling larger memory requirements. For example, with a memory requirement of 1024MB, TEE reached a best rate of 7634.0MB/s in the Copy operation, only slightly less than REE's 7677.5MB/s. This indicates that TA can still effectively handle complex memory operations when executing programs with large memory. Additionally, as the memory requirement increases, the performance gap between REE and TEE gradually narrows. Although there is still room for performance improvement in TEE for complex memory operations (such as Add and

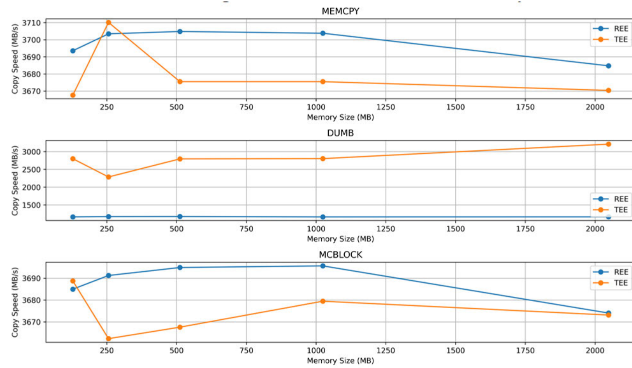

FIGURE 7. The results of mbw.

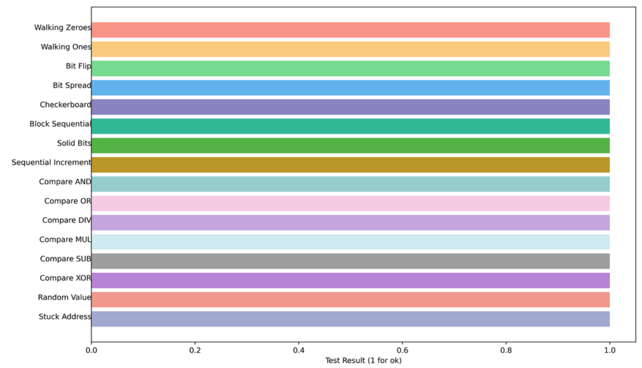

FIGURE 9. Memory stress test of TEEm.

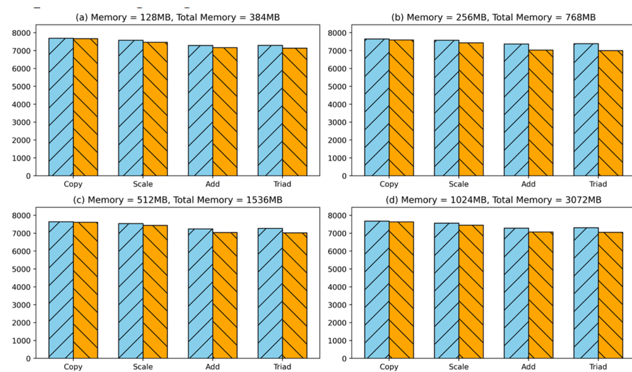

FIGURE 8. The results of stream. skyblue represents REE and orange represents TEE.

Triad), overall, the expansion of TA memory has significantly enhanced TEE's performance in processing large amounts of data.

D. MEMORY STRESS

We employ memtester [37], a widely recognized memory reliability testing tool, to validate the functional integrity of the newly allocated memory space. Memtester performs a variety of stress tests on memory, including pattern-based tests and algorithmic operations that simulate a wide range of memory access patterns. During memory testing, we noticed that the memtester tool employs the mlock and munlock system calls to lock the process's memory, preventing it from being swapped to disk. Although the current implementation of OP-TEE does not support these system calls within the TEE, TA operates in a strictly protected memory address space. In this secure setting, the sensitive data of TAs is effectively isolated, inaccessible to non-secure processes, and not subject to swapping to insecure storage mediums. Therefore, the absence of implemented memory locking does not impact the results of memory testing conducted by memtester. The memtester covered a variety of categories, each tailored to detect different potential errors. Stuck Address tests ensured that memory cells could store and switch between 0 and 1. Random Value tests checked the reliability of storing and retrieving random data. A series of binary operation tests, including XOR, SUB, MUL, DIV, OR, and AND, were conducted to verify data integrity. Sequential Increment tests were performed to identify errors in incrementally increasing memory values. Solid Bits, Block Sequential, and Checkerboard tests evaluated the memory's sensitivity to specific data patterns. Bit Spread and Bit Flip tests searched for interaction errors between adjacent memory cells. Lastly, Walking Ones and Zeroes tests were utilized to pinpoint failures in individual bits by moving a single '1' or '0' bit through a sea of the opposite value. The consistent positive outcome across all test categories from Figure 9 demonstrates that the expanded memory is stable and operates correctly under various conditions. The absence of errors in tests that stress different aspects of memory operations from storage of values to execution of complex binary operations suggests that the memory expansion not only increased the capacity but also preserved the expected functional reliability.

E. OVERHEAD OF MEMORY ALLOCATION

To evaluate the performance of memory allocation during TA runtime, we expanded TA's memory from 1GB to 30GB. As shown in Figure 10, in REE, the time required for memory allocation without initialization always remains zero, demonstrating a common deferred allocation strategy in the REE system. Actual physical memory allocation occurs only when the memory is first accessed, and merely requesting memory without initializing it does not immediately incur any time cost. In TEE, TA has already provided the allocated virtual addresses to the user-space malloc interface at runtime. This means that when TA requests memory, it is essentially accessing a pre-allocated and mapped virtual address space. Therefore, the time required for memory allocation without initialization is zero in TEE, as the physical memory has already been allocated and mapped at the time of TA loading. However, when memory is initialized in TEE, the time cost gradually increases from 65ms for 1GB to 1970ms for 30GB. Compared to REE, the rate of increase in time for memory initialization in TEE is significantly lower than that in REE. In REE, initialization time increases from 226ms for 1GB to 9711ms for 30GB, indicating that physical memory allocation and initialization operations bring greater time costs in

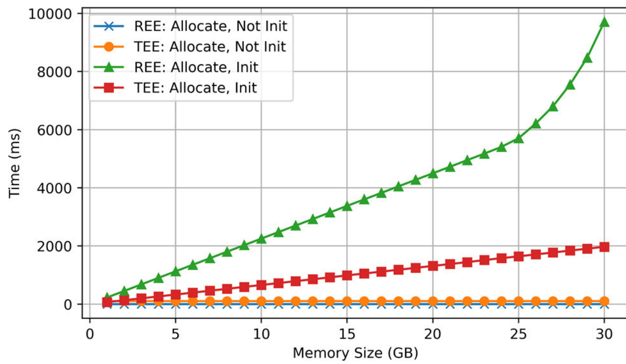

FIGURE 10. Comparison of allocation and initialization time of different memory sizes.

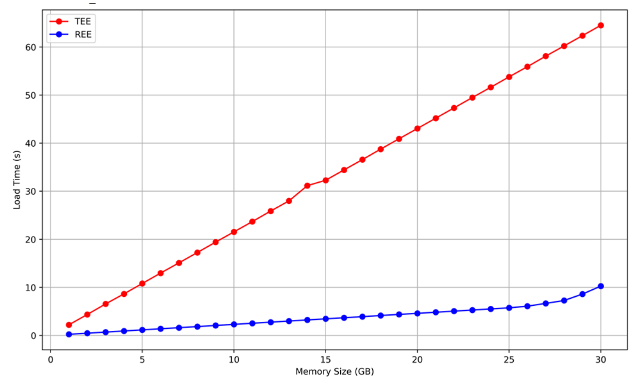

FIGURE 11. Time overhead of loading TA with different memory sizes.

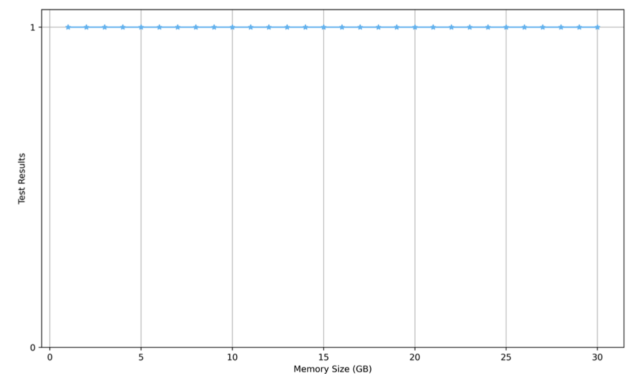

FIGURE 12. Memory allocation stability test of TEEm.

the REE environment. On average, TEE achieves a performance improvement of approximately 3.48 times over REE. Additionally, the experiment results demonstrate a consistent linear relationship between TA memory size and allocation time. This linear growth indicates that the memory allocation and initialization mechanism in TEE can effectively handle larger memory demands, with predictable costs, emphasizing the scalability and efficiency of expanded TA memory in TEEm.

F. TA LOADING TIME

We tested the loading times of TAs with different memory sizes, and as shown in Figure 11, the loading time of TAs increases linearly with the increase in memory size. Specifically, in the REE, the loading time for 1GB of memory is 0.232s, which increases to 9.05s for 30GB of memory; by contrast, in the TEE, loading times rise from 2.21s to 64.51s. This significant difference primarily stems from the additional security mechanisms in the TEE, as well as the complex interaction processes between the CA and TA. TEE includes various security measures, such as memory encryption and decryption, and complex authentication processes, which, while enhancing data protection, also introduce additional computational overhead. Additionally, the interaction between CA and TA involves multiple steps, such as context initialization, parameter passing, and secure channel establishment, which further contributes to the increased loading time. We will consider further analysis and optimization of the TA loading time as part of our future work.

G. MEMORY STABILITY

Memory stability is a crucial procedure to detect any potential errors, instability, or issues that may arise during memory allocation. In this experiment, to assess memory stability, we selected data blocks of different sizes, ranging from 1GB to 30GB, with a 1GB interval between each size. Figure 12 suggests that the stability test of memory allocation after memory expansion was successful across the entire range from 1GB to 30GB. This uniform success across a broad range of memory sizes indicates robustness in the memory allocation mechanism post-expansion. It demonstrates that the system's expanded memory is reliably accessible to applications.

H. TENSORFLOW LITE

In this experiment, we conducted comparative tests on confidence levels using TensorFlow Lite [38] models in both REE and TEE. We selected a range of deep learning models of varying sizes for evaluation, such as MobileNet V1, Inception V1, V2, V3, and V4, and Resnet V2, ranging from 4MB to 170MB. By comparing the model executions in REE and TEE, we assessed the performance of TEEm's large memory support in practical applications. The results show that even memory-intensive models, such as the 170MB Resnet V2, can be run directly in TEE without any modifications to the model. It is important to note that in the REE, we used the OpenCV library for data processing, whereas in the TEE, a functionally similar C++ library was employed. This difference in processing approaches directly affects the output confidence of the models. For instance, the confidence level of MobileNet V1 in REE is 0.98, while in TEE it is 0.87. We leave the work of optimizing model accuracy to developers. Figure 13 provides the confidence for these models. Through these experiments, we have verified the capability of the TEEm framework to support TensorFlow Lite models in large memory environments, demonstrating its advantages in terms of security and scalability.

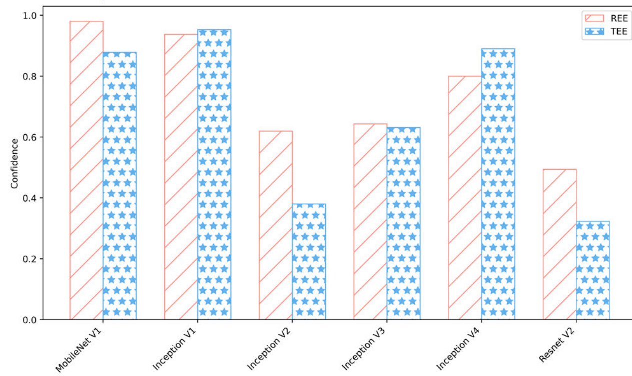

FIGURE 13. Performance of TensorFlow Lite on TEEm.

VI. DISCUSSION

The performance evaluation revealed notable differences between the REE (based on the Kylin system) and TEE (based on the OP-TEE). The reasons for these differences are primarily due to two factors: memory allocation strategies and memory allocators. The traditional demand allocation strategy used in REE, which records memory requests without immediate allocation, is efficient in resource utilization but may lead to delays in actual memory usage. In contrast, TEE employs a pre-allocated memory strategy, loading predetermined heap memory into the allocator before execution, which reduces latency and enhances performance. Additionally, the choice of memory allocator also plays a crucial role in performance differences. The REE uses the standard GNU glibc memory allocator [39], which is designed for general-purpose operating systems and emphasizes universality and flexibility. On the other hand, TEE uses the bget memory allocator [40], [41], which is specifically designed for secure environments and offers performance benefits tailored to the security requirements of TEEs. The bget allocator, therefore, aligns well with the needs of TEE applications, providing optimized performance.

To ensure the robustness and reliability of our findings, we have conducted statistical tests (t-tests) [42] to compare the performance differences between TEE and REE for both the mbw and stream across different memory sizes. The results show that for the mbw, the DUMB and MCBLOCK methods have statistically significant differences (p-values of 4.738e-05 and 0.045, respectively), while the MEMCPY method does not (p-value of 0.068). For the stream, the Scale, Add, and Triad methods show statistically significant differences (p-values of 4.459e-05, 0.0019, and 0.00057, respectively), while the Copy method does not (p-value of 0.088). These results indicate that TEEm and REE perform similarly in many cases, with significant differences in specific operations.

VII. SECURITY ANALYSIS

In this section, we delve into the security implications of memory extension made to OP-TEE, particularly focusing on the expanded TA memory, efficient memory allocation, and the balancing act between preserving a small TCB size. The expansion of the TA memory space has resulted in an enhancement of system performance. However, the increased address space might introduce new vulnerabilities, especially in scenarios where memory allocation and access control mechanisms are not meticulously managed. For this, TEEm depends on hardware-based isolation mechanisms in ARM TrustZone. TrustZone implements memory access control through the TrustZone Address Space Controller (TZASC), which ensures that memory access between the secure world and the normal world is strictly separated [3]. The TrustZone Protection Controller (TZPC) is used to set peripheral access permissions, ensuring that devices accessible by the secure world cannot be accessed by the normal world, thereby achieving peripheral isolation. In terms of memory, TZASC effectively provides each of the TEE and the REE with isolated physical address spaces by configuring memory region attributes. Memory in the secure world can only be accessed by the code residing on the TEE side, with this access security enforced through the NS bit check. The memory region configured by the CFG_TA_RAM_START and CFG_TA_RAM_SIZE parameters is shared for TAs, but different TAs have separate memory regions that do not affect each other. Each TA creates the TEEC_Context (ctx) when running, which manages the page tables. The ctx created by different TAs are independent. TEEm builds upon these mechanisms and has made essential software modifications to OP-TEE's memory management module to accommodate larger memory demands. Specifically, although the S2M strategy allows TAs to manage multiple L2 page tables, it does not introduce new security risks. In the S2M, the TA's page tables are still managed by ctx, and there is no cross-access between different TA page tables since each TA's ctx is different. Moreover, since OP-TEE currently only supports single-threaded, the thread is bound to the L2 page table during memory initialization, there is no situation where multiple threads access the same page table at the same time.

By reutilizing existing memory management mechanisms and making only necessary modifications, only about 200 lines of code were added to TEE OS. Specifically, the S2M mapping policy extended the L2 page tables to support larger memory allocations. Traditionally, OP-TEE utilizes fixed-size page tables that limit the maximum memory that can be securely managed. By increasing the number of entries in these tables and adjusting the page table handling logic, we accommodated larger memory blocks required by advanced applications without altering the fundamental page table architecture used in TrustZone. We introduced the PbMA to allocate memory based on the requirements of each TA dynamically. This mechanism uses a configurable parameter that determines the size of memory allocated to each TA at runtime, rather than being fixed at compile time. The TEE kernel will check the validity of the parameters when loading TA. If the parameter size exceeds the memory allocated to TA, an error will be reported. The smaller TCB reduces the attack surface, thereby decreasing the likelihood

of introducing new vulnerabilities. Additionally, these modifications were implemented based on a software approach while strictly adhering to the security mechanism provided by ARM TrustZone, ensuring that all memory operations remained within the secure execution environment and were fully isolated from the normal world.

To ensure the robustness and security of TEEm, we utilized Xtest [43], a comprehensive test suite designed by OP-TEE. The results demonstrated the success of all test cases, indicating that the modifications made to accommodate larger memory have not introduced any new errors or stability issues, thereby maintaining the system's high stability and reliability. Therefore, TEEm achieves a balance between performance and security. TEEm is applicable in both mobile and cloud computing environments. In mobile devices, TEEm provides the necessary support for large memory, enabling more complex and secure applications to run smoothly. For cloud platforms, the large memory capacity of TEEm is advantageous for handling memory-intensive tasks. Essentially, TEEm's design offers a simple and practical solution for extending the utility of TEE across various computing environments, ranging from mobile devices to extensive cloud infrastructures.

VIII. CONCLUSION

In this paper, we propose TEEm, the first publicly disclosed design for extending TA memory in ARM TrustZone using a software-based approach. TEEm leverages the isolation mechanism of TrustZone and the security management features of OP-TEE to protect software and physical attacks while offering memory expansion capabilities for TAs. We design the S2M mapping policy, which significantly extends the virtual address space available to TAs. Furthermore, the PbMA mechanism is introduced, empowering TAs to request more trusted memory from the TEE kernel. We conducted extensive experiments to evaluate the performance of TEEm in executing typical memory tasks. The results show that the memory allocation performance of TEEm is, on average, approximately 3.48 times faster than that of REE. In future research, we aim to further optimize the memory initialization time of TA, especially as memory allocation scales, making TEEm more practical for real-world applications. Furthermore, we plan to explore new application scenarios, such as large-scale data processing and machine learning model training, to further validate the applicability and advantages of TEEm.

REFERENCES

[1] *Software Guard Extensions Solution Brief*, 2022. Accessed: Feb. 13, 2024. [Online]. Available: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2022-09/sgx-protect-and-isolate-confidential-data-sb.pdf

[2] *AMD Memory Encryption*. Accessed: Feb. 13, 2024. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/memory-encryption-white-paper.pdf

[3] *ARM Confidential Compute Architecture*. Accessed: Feb. 13, 2024. [Online]. Available: https://developer.arm.com/documentation/den0125/0300/?lang=en

[4] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanovic, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proc. 15th Eur. Conf. Comput. Syst.*, Apr. 2020, pp. 1–16.

[5] *SGX-Hardware* Accessed: Feb. 13, 2024. [Online]. Available: https://github.com/ayeks/SGX-hardware

[6] D. Lu, M. Shi, X. Ma, X. Liu, R. Guo, T. Zheng, Y. Shen, X. Dong, and J. Ma, "Smaug: A TEE-assisted secured SQLite for embedded systems," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 5, pp. 3617–3635, Sep. 2023.

[7] J. Menetrey, M. Pasin, P. Felber, and V. Schiavoni, "WATZ: A trusted WebAssembly runtime environment with remote attestation for TrustZone," in *Proc. IEEE 42nd Int. Conf. Distrib. Comput. Syst.*, 2022, pp. 1177–1189.

[8] Z. Jian, Y. Lu, Y. Qiao, Y. Fang, X. Xie, D. Yang, Z. Zhou, and T. Li, "TSC-VEE: A TrustZone-based smart contract virtual execution environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 6, pp. 1773–1788, Jun. 2023.

[9] F. Mo, A. S. Shamsabadi, K. Katevas, S. Demetriou, I. Leontiadis, A. Cavallaro, and H. Haddadi, "DarkneTZ: Towards model privacy at the edge using trusted execution environments," in *Proc. 18th Int. Conf. Mobile Syst., Appl., Services*, Jun. 2020, pp. 161–174.

[10] S. P. Bayerl, T. Frassetto, P. Jauernig, K. Riedhammer, A.-R. Sadeghi, T. Schneider, E. Stapf, and C. Weinert, "Offline model guard: Secure and private ML on mobile devices," in *Proc. Design, Autom. Test Eur. Conf. Exhibition*, Mar. 2020, pp. 460–465.

[11] S. Pinto and N. Santos, "Demystifying arm TrustZone: A comprehensive survey," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–36, Jan. 2019.

[12] Z. Hua, J. Gu, Y. Xia, H. Chen, B. Zang, and H. Guan, "VTZ: Virtualizing ARM TrustZone," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 541–556.

[13] S. Zhao, Q. Zhang, Y. Qin, W. Feng, and D. Feng, "SecTEE: A software-based approach to secure enclave architecture using TEE," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1723–1740.

[14] Z. Li, W. Li, Y. Xia, and B. Zang, "TEEp: Supporting secure parallel processing in ARM TrustZone," in *Proc. IEEE 26th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2020, pp. 544–553.

[15] L. Sun, S. Wang, H. Wu, Y. Gong, F. Xu, Y. Liu, H. Han, and S. Zhong, "LEAP: TrustZone based developer-friendly TEE for intelligent mobile apps," *IEEE Trans. Mobile Comput.*, vol. 22, no. 12, pp. 7138–7155, Dec. 2023.

[16] J. Jiang, J. Qi, T. Shen, X. Chen, S. Zhao, S. Wang, L. Chen, G. Zhang, X. Luo, and H. Cui, "CRONUS: Fault-isolated, secure and high-performance heterogeneous computing for trusted execution environment," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Chicago, IL, USA, Oct. 2022, pp. 124–143.

[17] S. Wan, M. Sun, K. Sun, N. Zhang, and X. He, "RusTEE: Developing memory-safe ARM TrustZone applications," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2020, pp. 442–453.

[18] O. Demigha and R. Larguet, "Hardware-based solutions for trusted cloud computing," *Comput. Secur.*, vol. 103, Apr. 2021, Art. no. 102117.

[19] *OP-TEE*. Accessed: Feb. 13, 2024. [Online]. Available: https://github.com/OP-TEE/

[20] *Armv8-A Address Translation*. Accessed: May 13, 2024. [Online]. Available: https://developer.arm.com/documentation/100940/latest/

[21] S. Park, H. Kang, S. Han, J. M. Youn, and D. Kwon, "MECAT: Memory-safe smart contracts in ARM TrustZone," *IEEE Access*, vol. 12, pp. 56110–56119, 2024.

[22] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf, "SANCTUARY: ARMing TrustZone with user-space enclaves," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019, pp. 1–25.

[23] Z. Liu, Y. Lu, X. Xie, Y. Fang, Z. Jian, and T. Li, "Trusted-DNN: A TrustZone-based adaptive isolation strategy for deep neural networks," in *Proc. ACM Turing Award Celebration Conf.*, Jul. 2021, pp. 67–71.

[24] M. S. Islam, M. Zamani, C. H. Kim, L. Khan, and K. W. Hamlen, "Confidential execution of deep learning inference at the untrusted edge with ARM TrustZone," in *Proc. 13th ACM Conf. Data Appl. Secur. Privacy*, Apr. 2023, pp. 153–164.

[25] Y. Cao, J. Zhang, Y. Zhao, P. Su, and H. Huang, "SRFL: A secure & robust federated learning framework for IoT with trusted execution environments," *Expert Syst. Appl.*, vol. 239, Apr. 2024, Art. no. 122410.

[26] X. Xie, H. Wang, Z. Jian, T. Li, W. Wang, Z. Xu, and G. Wang, "Memory-efficient and secure DNN inference on TrustZone-enabled consumer IoT devices," 2024, *arXiv:2403.12568*.

[27] M. Yang, W. Yi, J. Wang, H. Hu, X. Xu, and Z. Li, "Penetralium: Privacy-preserving and memory-efficient neural network inference at the edge," *Future Gener. Comput. Syst.*, vol. 156, pp. 30–41, Jul. 2024.
[28] D. Shim and D. H. Lee, "SOTPM: Software one-time programmable memory to protect shared memory on ARM trustzone," *IEEE Access*, vol. 9, pp. 4490–4504, 2021.
[29] B. Gowrisankar, D. Mashima, W. Ong, Q. Ye, E. Esiner, B. Chen, and Z. Kalbarczyk, "GateKeeper: Operator-centric trusted app management framework on ARM TrustZone," in *Proc. IEEE Conf. Commun. Netw. Secur.*, Oct. 2022, pp. 100–108.
[30] *ARM. Building a Secure System Using TrustZone Technology*. Accessed: Feb. 13, 2024. [Online]. Available: https://documentation-service.arm.com/static/5f212796500e883ab8e74531?token=
[31] GlobalPlatform. *TEE System Architecture*. Accessed: Feb. 13, 2024. [Online]. Available: https://globalplatform.org/specs-library/tee-system-architecture/
[32] J.-E. Ekberg, K. Kostiainen, and N. Asokan, "The untapped potential of trusted execution environments on mobile devices," *IEEE Secur. Privacy*, vol. 12, no. 4, pp. 29–37, Jul. 2014.
[33] *ARM Trusted Firmware*. Accessed: Feb. 13, 2024. [Online]. Available: https://www.trustedfirmware.org/
[34] *OP-TEE Architecture*. Accessed: Feb. 13, 2024. [Online]. Available: https://optee.readthedocs.io/en/latest/architecture/index.html
[35] *MBW*. Accessed: Feb. 13, 2024. [Online]. Available: https://github.com/raas/mbw
[36] *Stream*. Accessed: Feb. 13, 2024. [Online]. Available: https://github.com/jeffhammond/STREAM
[37] *Memtester*. Accessed: Feb. 13, 2024. [Online]. Available: https://github.com/jnavila/memtester
[38] M. Abadi et al., "Tensorflow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Des. Implement.*, 2016, pp. 265–283.
[39] *The GNU Allocator*. Accessed: May 13, 2024. [Online]. Available: https://www.gnu.org/software/libc/manual/html_node/The-GNU-Allocator.html
[40] *The BGET Memory Allocator*. Accessed: May 13, 2024. [Online]. Available: https://www.fourmilab.ch/bget/
[41] *BGET*. Accessed: May 13, 2024. [Online]. Available: https://github.com/OP-TEE/optee_os/pull/120
[42] *T-Test*. Accessed: May 13, 2024. [Online]. Available: https://resources.nu.edu/statsresources/ttest
[43] *OP-TEE Xtest Framework*. Accessed: Feb. 13, 2024. [Online]. Available: https://github.com/OP-TEE/optee_test

JUN LI received the bachelor's degree from Tianjin University of Technology and Education, Tianjin, China, in 2017, and the master's degree from Hainan University, Hainan, China, in 2021, where he is currently pursuing the Ph.D. degree with the School of Cyberspace Security (School of Cryptology). His research interests include cybersecurity, trusted hardware, and blockchain.

XINMAN LUO received the bachelor's degree from Xijing University, Xi'an, China, in 2019, and the master's degree from Hainan University, Hainan, China, in 2022. She is currently a full-time Teacher with Qiongtai Normal University. Her research interests include network security and trusted computing.

HONG LEI received the bachelor's and master's degrees from Beijing University of Aeronautics and Astronautics, Beijing, China, in 2006 and 2009, respectively, and the Ph.D. degree from Michigan State University, East Lansing, MI, USA, in May 2015. He continued as a Research Associate with the Smart Microsystem Laboratory, Michigan State University. He joined the Department of Electrical and Computer Engineering, as a Tenure-Track Assistant Professor with Portland State University, in July 2018. He was the Associate Dean of Oxford–Hainan Blockchain Research Institute, SSC Holding Company Ltd. He is currently a Professor with Hainan University. His research interests include smart systems, trusted hardware, and blockchain technology.

JIEREN CHENG (Member, IEEE) received the Ph.D. degree in computer science and technology from the School of Computer, National University of Defense Technology, Changsha, China, in 2010.

He is currently a Vice Dean, a Professor, and a Ph.D. Supervisor with Hainan University, Haikou, China. He is the Director of Hainan Provincial Blockchain Technology Engineering Research Center. He hosted three National Natural Science Foundation of China, National Defense Key Research Projects, China–U.S. Computer Science Research Center Open Project, Ministry of Education Industry-University-Research Collaborative Education Project, Ministry of Education "Tiancheng Huizhi" Innovation and Education Fund, Hainan Province Key, Research and Development Innovation Team Projects, Hainan Provincial Key Research and Development Projects, Hainan Provincial Natural Science Foundation Projects, Hainan Provincial Science and Technology Enterprise Technology Innovation Fund Projects, and Hunan Provincial Twelfth Five-Year Plan Projects, with a total project funding of more than ten million. He has participated in ten national key projects as the main person in charge, including the National Natural Science Foundation of China, the National Defense Preliminary Research Key Project, the National Support Plan, and the Innovation Planning Project of the Ministry of Public Security. He has won 24 provincial-level projects and 12 school-level projects, such as the Provincial Natural Science Foundation, the Provincial Science and Technology Plan Fund, and the Provincial Department of Education Key Project. His research interests include cloud computing, artificial intelligence, network security, and intelligent transportation. He is a Senior Member of CCF and a member of ACM. He was awarded the "Famous South China Sea Scholar." He won the ICAIS 2021 Outstanding Organization Chairperson, the ICCCS 2018 Outstanding Contribution Award, and the first prize of ICCCS 2018 and ICCCS 2017 Excellent Papers. He has been invited to serve as a reviewer for several journals and international conferences, e.g., *Journal of Computer Research and Development*, *Computer Science*, and FAW, and a PC member for several international conferences.