FatPTE - Expanding Page Table Entries for Security

Lukas Lamster [1](✉), Martin Unterguggenberger [1], Moritz Waser [1], David Schrammel [2*], and Stefan Mangard [1]

[1] Graz University of Technology firstname.lastname@tugraz.at
[2] Rivos Inc. davidschrammel@rivosinc.com

Abstract. Modern computing systems have a strong need for security and require protection against attacks such as control-flow hijacking, information leakage, or data manipulation. As a solution, academia and industry propose essential security technologies operating on page granularity. Through metadata bits located in page table entries (PTEs), these technologies provide security with high efficiency. While PTE bits allow for highly efficient implementations, the reliance on spare bits is not future-proof. Due to the steady increase in memory capacity and the introduction of new security features, the spare bits are exhausted. Thus, the implementation of new features is impossible while the security of existing features is severely limited.

In this work we introduce FatPTE, a novel approach that enhances page table entries with dedicated metadata regions for security features. Our design provides up to 192 metadata bits, thus far exceeding the 7 reserved bits of x86-64 and RISC-V. We perform a case study on academic and commercial PTE-based control-flow integrity, memory protection, and confidential computing features. Our findings show that FatPTE easily accommodates all bits needed by the considered features, thus highlighting the practical relevance of our design. We implement an x86-64 prototype using the gem5 system simulator as well as a RISC-V FPGA prototype using the CORE-V CVA6 processor. We evaluate FatPTE in four configurations derived from the requirements identified in our case study. Our evaluation using SPEC CPU 2017 workloads yields a geomean performance overhead of 0.21% to 1.34% for the gem5 simulator and 0.51% to 1.99% for the FPGA prototype.

1 Introduction

Software vulnerabilities allow attackers to hijack a program's control flow, disclose secret information, corrupt victim data, or even take over the complete system. Thus, strong security measures protecting against a range of attack vectors are necessary. CPU vendors and academic researchers proposed a variety of security features mitigating the increasing number of vulnerabilities [33,11,17,19,21].

* The work was done while the author was at Graz University of Technology.

A core principle of such security features is memory protection. Memory Protection Keys (MPK) efficiently isolate in-process memory through page-granular access restriction [2,15,21]. Confidential computing technologies protect memory through cryptographic isolation and facilitate secure virtual machines, page-granular encryption, and the secure execution of mutually untrusted software [13,14,2]. Shadow stacks provide control-flow integrity by confining return addresses to access-protected memory regions [15,37]. Due to the strong need for *efficient page-granular protection*, most security features implemented in commercial processors store metadata in spare bits of the system's page table entries (PTEs) [33,11,17,19,21]. With the constant increase in system memory capacity the number of spare bits is steadily decreasing. While repurposing PTE bits has worked in the past, this approach has ultimately reached its limits in current CPU generations. Not only does the scarcity of PTE bits prevent the implementation of additional PTE-based security features, *it also severely limits the security of existing features.* The memory protection keys of AMD and Intel, for example, are constrained to four bits per page, thus only allowing for 16 protection domains per page. Memory encryption schemes require up to 15 key identifier bits, yet only 7 bits are available in x86-64 and RISC-V systems. Thus, they repurpose physical address bits in the PTE and reduce the amount of addressable system memory to a degree that is unsustainable for current-generation server systems. Upcoming academic and commercial security features are unable to use PTE bits and must rely on more inefficient methods of storing the required metadata. Hence, *the adaptation of novel security features using PTE bits is hindered and existing features are strongly limited.*

In this work, we present FatPTE, a design that extends page table entries such that all required security-critical metadata is co-located with the paging information *without repurposing any PTE bits.* By increasing the size of PTEs to *explicitly account for security-critical metadata*, we eliminate the limitations imposed by the increase of system memory capacity and the growing number of competing schemes. We propose two variants of FatPTE allowing for 64 or 192 metadata bits, which is a vast improvement over the 7 remaining bits in existing x86-64 and RISC-V processors. We conduct a case study on commercial and academic memory protection and confidential computing security features [33,11,17,19,21]. Not only does FatPTE allow the implementation of all considered page-granular security schemes *without repurposing PTE bits*, it also increases the security of certain schemes, e.g., by increasing the number of available protection domains or the amount of encryption keys. Based on the case study, we identify parameterizations that allow the efficient implementation of existing security features using FatPTE. We use the gem5 computer architecture simulator system [4] to implement an out-of-order prototype of FatPTE for x86-64. Moreover, we implement an in-order FPGA prototype by integrating FatPTE into the CORE-V CVA6 application-class RISC-V processor. Our prototypes allows us to perform a performance evaluation for the previously identified configurations using SPEC CPU 2017.

Our evaluation yields a maximum geomean performance overhead of 1.34% for x86-64 and 1.99% for RISC-V, showcasing the feasibility of our approach. We find that FatPTE allows to *implement all the security schemes discussed in our case study with minimal performance and memory overhead.*

Contributions. We make the following key contributions:

- We present FatPTE, a lightweight hardware extension that enhances PTEs with metadata regions, providing up to 192 bits for security features.
- We perform an extensive case study demonstrating that FatPTE enables the integration of all discussed countermeasures while increasing the security of most of them.
- We provide an x86-64 prototype based on the gem5 simulator and a RISC-V FPGA implementation of FatPTE using the CORE-V CVA6 processor.
- We evaluate FatPTE, showcasing a maximum geomean overhead of 1.34% and 1.99% for x86-64 and RISC-V, respectively.

Outline. The paper is structured as follows. Section 2 provides the background on virtual memory in modern operating systems and hardware support for paging. Section 3 discusses the current issues faced when implementing schemes that rely on spare PTE bits and motivates our approach. Section 4 presents the design space and possible configurations of FatPTE. Section 5 conducts a case study showing the wide applicability and feasible parameterizations of FatPTE. Section 6 discusses the prototype implementations of FatPTE and provides the respective evaluation. Section 7 compares FatPTE to related work, while Section 8 concludes this work.

2 Background

In this section, we discuss virtual memory and paging in modern operating systems. Furthermore, we elaborate on hardware support for paging in modern CPUs.

2.1 Virtual Memory and Paging

Virtual memory is a fundamental feature of modern computing systems. With virtual memory, processes do not directly access to physical memory. Instead, the OS provisions virtual addresses, which are translated to physical addresses upon use. From an application's perspective, virtual memory is a continuous range, while the OS can allocate and map physical memory on demand in a fragmented fashion. This provides an elegant solution to several problems that arise in modern multithreaded and multiprocessed environments. It enables efficient management of memory, allows for access permission control, and provides strong isolation between processes that operate in disjunct virtual memory spaces.

Pages and Page Sizes. Modern virtual memory systems divide their physical memory into *pages*, which serve as memory granules that are distributed to processes by the operating system. This smallest provisioned memory granule is

sometimes called the *translation granule* [3]. The actual granule and the number of available page sizes depend on the instruction set architecture (ISA). In x86-64 and RISC-V systems, the translation granule is 4 KiB and regular pages hold 4 KiB of data. On AArch64, the translation granule can be 4 KiB, 16 KiB, or 64 KiB. Each virtual memory page has a virtual page number and maps to one physical page in memory. While each virtual page points to *exactly one* physical page, the opposite is not necessarily true. Through *alias mappings*, multiple virtual addresses can refer to the same physical page, which enables features such as shared memory or shared libraries.

Page Tables and Translation Tables. The mappings from virtual to physical pages are stored in hierarchical data structures called *page tables* (PTs) or *translation tables*. A PT holds *page table entries* (PTEs), which contain a *physical page number* (PPN) and its associated access permissions. Typically, one bit of a PTE determines if the referred physical page is a leaf node, *i.e.*, it holds data, or if it contains another level of the paging hierarchy. The translation consists of a multi-stage table lookup where parts of the virtual address are used as indices for PTs. Depending on the number of layers n in the hierarchy, we talk about *n-level paging*. In current x86-64 systems, a page table may consist of up to five layers, allowing the operating system to provision virtual addresses with at most 57 address bits. Since the translation requires a starting point, every process is assigned a root page table as the base of virtual address translation. Providing each process with a separate paging hierarchy isolates their views of virtual memory against each other. Through these disjoint views, each process can use the full virtual address space without interfering with other processes.

In modern systems, the PTEs in each layer are 64 bits wide. However, the physical address indexing the underlying memory does not use the full number of available bits. Current x86-64 CPUs allow up to 52-bit physical addresses, thus theoretically supporting petabytes of main memory. However, in practice the amount of supported DRAM is often limited to a few terabytes. The lowermost 12 bits of the physical address in a leaf PTE are not explicitly stored as they determine the offset within the page and are instead taken from the virtual address during memory accesses. The remaining PTE bits are typically used to encode status and permission bits, while some bits remain unused.

Figure 1 illustrates 5-level paging with a 4 KiB page size, as used by x86-64 or RISC-V processors. Each level in the paging hierarchy except for the last layer (*i.e.*, PTL4 to PTL0) represents a page table indexed by 9 bits of the virtual address. With 9 index bits for each of the five layers and a 12-bit index for the leaf page, the 7 topmost bits of each 64-bit virtual address remain unused. Thus, current paging implementations of the Linux kernel for these ISAs support up to 57-bit virtual addresses. Translating a virtual to a physical address requires a sequential table lookup called *page table walk*. The paging structure's root is formed by the physical page number (PPN) of the root page table. This PPN is stored in a control register (CR3 on x86-64, SATP on RISC-V, TTBR0_ELx on ARM). The root page table holds the topmost entries of the paging hierarchy. The least significant bits of the virtual address act as a byte-granular offset for

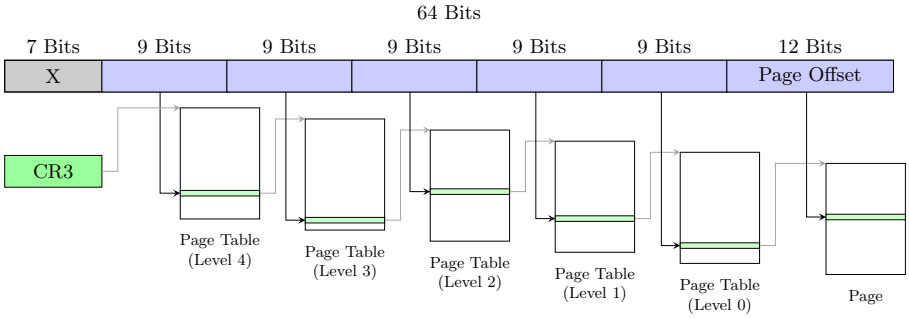


Fig. 1: 5-level paging in x86-64. On the lowermost level, 12 bits are used to address individual bytes within a page while the 9-bit values in the upper levels act as indices that select 64-bit entries within page tables.

the leaf page. Note that the number of offset bits equals the base-2 logarithm of the number of bytes per leaf page.

Huge Pages. Virtual memory systems often support *huge pages*, which cover a larger continuous memory region than the basic translation granule [34]. A huge page can be mapped by removing a translation layer and extending the offset bits with the former page table index bits. For instance, in a 4 KiB virtual memory system, a 2 MiB page can be mapped by removing the final translation layer and extending the 12-bit offset to a 21-bit offset using the 9 bits that were used as an index for the final page table. Depending on the translation granule and the presence of hardware support, varying sizes of huge pages can be mapped. Mapping huge pages reduces the metadata required for continuous memory regions and can improve performance for large allocations.

2.2 Hardware Support for Paging

In the case of 5-level paging, a single memory access may entail six loads from memory as each layer must be fetched sequentially. The first five loads fetch the PTEs, while the sixth memory load fetches the accessed memory location. Loading data from main memory frequently has a detrimental performance impact as DRAM is significantly slower than the on-chip caches. Memory-related operations such as loading data, checking access permissions, and performing address translations are handled by the *memory management unit* (MMU). Due to the high frequency of memory operations and their performance-critical nature, this component is deeply integrated into the CPU. When performing an address translation, the *page table walker* (PTW), which is a part of the MMU, iteratively fetches the PTEs according to the page tables of the current process. To further reduce the performance impact of address translations, recently used mappings are cached in the *translation lookaside buffer* (TLB). Modern CPUs use multiple TLBs in different layers of the caching hierarchy. A TLB entry maps

a combination of an address space identifier (ASID), which is required to distinguish processes, and a virtual page number, to a physical page and its associated permissions. When accessing a virtual address the CPU checks if the required translation can be directly served from the TLB. If a valid entry is present, a *TLB hit* occurs and the costly page table traversal is completely avoided.

3 The Need for Larger PTEs

Historically, PTEs consisted of a physical page number, permission bits, and a set of software-defined bits. This only left a small number of bits available for future use. 64-bit systems increase the PTE size from 32 bits to 64 bits, increase the size of the PPN field, and add more page attributes and permission bits such as the no-execute bit. Modern Intel x86-64 CPUs with 5-level paging allow for up to 53 physical address bits [15]. Combined with the other PTE bits in use, PTEs that map 4 KiB pages only provide seven bits for future use. Similarly, RISC-V's Sv57 format provides seven reserved bits for future use [23].

The introduction of security features like control-flow integrity, memory protection keys, and confidential computing [17,19,15] has decreased the number of available PTE bits drastically. Memory Protection Keys (MPK) use four bits to tag a page with a protection key and allow quick page permission modifications from userspace without a context switch [15,2]. Intel's Total Memory Encryption requires up to 15 bits to be stored in the PTE and currently repurposes physical address bits to store their key identifiers [15]. Intel Control-Flow Enforcement Technologies (CET) [33] introduces a *shadow stack* whose pages are identified by an unused combination of permission bits in the PTEs. Confidential computing architectures like Intel TDX [12,14] and AMD SEV [1] both require one PTE bit to indicate whether a memory page is encrypted or not [17]. This bit is also taken from the physical address, thus further reducing the physical address space.

Besides commercially available security features, a range of academic designs proposes repurposing the available PTE bits for added security. For example, PT-Guard [24] co-locates a MAC to detect Rowhammer attacks, and Multi-Tag stores a memory tag for memory safety [36].

All features requiring PTE bits compete for the few available spare bits. This hinders the adoption of new features, limits the achievable security, and ultimately slows the progress of innovation. Intel's shadow stack already resorts to using a special encoding of the regular permission bits to avoid using an additional PTE bit [15]. Features like TME-MK must repurpose physical address bits, thus limiting the amount of usable DRAM [11]. Finally, Intel's Sub-Page Write-Permission (SPP) feature requires 64 bits to be stored per page [15]. This per-page metadata must be fetched on-demand and accessed through a number of indirections, similar to existing page tables.

Clearly, per-page metadata is essential for many security features. The existing PTE structure is the ideal location for metadata, as it contains all other paging information. Furthermore, for most security features the metadata must

be fetched and checked synchronous to the associated PTEs. Hence, there is a clear need for larger PTEs that can co-locate security-critical metadata.

4 Design

In this section, we discuss the design principles of FatPTE. When discussing page tables and their modifications, we must distinguish between the different levels in the paging hierarchy. We refer to the lowest hierarchy level as the *last* layer and to the highest level as the *first* layer, according to the order in which they are fetched during a page table walk. A layer is considered *above* another layer if it is fetched first when traversing the hierarchy. We use *PTE* to refer to all entries of the paging structure, regardless of the layer. For clarity, we refer to the PTEs modified by our approach as fat page table entries (FPTEs). An FPTE consists of a legacy PTE and the associated metadata. A *leaf PTE* or *leaf node* refers to an entry that does not point to another translation layer but to a page containing arbitrary data. Upon reaching a leaf node, the translation is finished. In our design exploration, we identify four key parameters $\mathcal{P}_1$ to $\mathcal{P}_4$.

4.1 Metadata Region Size $\mathcal{P}_1$

The main goal of FatPTE is to extend the available number of bits for security features. Our approach enhances PTEs with a dedicated *page table entry metadata region* (PTE-MR), thus transforming them to FPTEs. Assuming a constant page size, using FPTEs instead of legacy PTEs reduces the overall number of entries that fit on a single page. Thus, the amount of mappable memory covered by the same number of hierarchy levels is also reduced. As each PTE points to a single page, using FPTEs which are twice as large halves the size of the memory region that can be mapped by an entry in the layer directly above (cf. Figure 2). Note that the overall amount by which the mappable memory decreases is *independent of the layer* at which FPTEs are used. It only depends on the *number* of layers with FPTEs and the size difference between FPTEs and PTEs. Increasing the size of FPTEs too much increases memory overheads as more entries are required to map the same amount of memory.

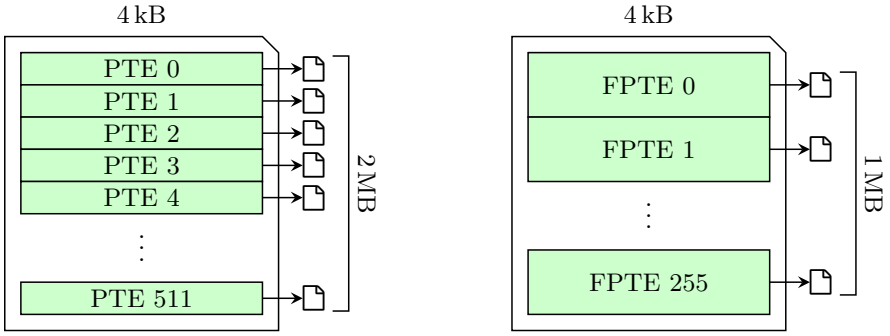
Each doubling in FPTE size halves the number of mappable memory and decreases the number of available virtual address bits by one. Thus, we are bound to power-of-two sizes. Using any other factor results in a mapping scheme requiring complex encodings of virtual addresses. Ultimately, this would lead to issues in pointer arithmetic and greatly increase the complexity of the PTW.

Given the constraints of a power-of-two size increase and the implications on the amount of mappable memory, we propose two parameterizations.

64-bit PTE-MR. First, we consider the case in which the additional metadata region holds 64 bits. With this configuration, the overall size of each FPTE equals 128 bits. We can illustrate the consequences of this size increase through a concrete example. Consider replacing the last-layer PTEs of a system with

4 KiB pages by FPTEs. The page containing the last-layer PTEs originally holds 512 64-bit entries. This number decreases to 256 FPTEs as the page size stays constant. A single 4 KiB page of PTEs originally maps a memory range of 2 MiB. When using FPTEs, the mappable memory decreases to 1 MiB (cf. Figure 2).

In commodity CPUs, the memory system operates on a 512-bit (cache line) granularity. Thus, a single 4 KiB page encompasses 64 cache lines. With the above configuration, we provide one metadata bit per cache line of the page, which can already be suitable for simple schemes at cache line granularity.



(a) A 4 KiB page contains 512 64-bit PTEs and can map 2 MiB.

(b) Only half as many FPTEs fit on the same page which can map 1 MiB.

Fig. 2: Using FPTEs reduces the amount of mappable memory. For an x86-64 system with 4 KiB pages, a single page of last-level PTEs can map 2 MiB. Doubling the size of each entry decreases this amount to 1 MiB.

192-bit PTE-MR. In this configuration, we further increase the size of the metadata region to hold up to 192 bits. This results in 256-bit FPTEs. As we further increase the size of the FPTEs, the number of FPTEs per page continues to decrease. Thus, with this configuration, a single page of FPTEs can only map a quarter of the memory that was originally mappable by the same number of entries. While this parameterization reduces the amount of mappable memory more significantly than the 64-bit variant, it simultaneously allows for the implementation of more complex schemes. The 192-bit variant provides three bits per cache line on 4 KiB pages, thus allowing for more complex schemes.

4.2 Metadata Region Layout $\mathcal{P}_2$

When enhancing PTEs with additional metadata, we cannot use a dedicated metadata storage region akin to tagged memory architectures. As we associate metadata with PTEs, the required metadata memory changes during runtime. Hence, we cannot statically reserve a memory region for metadata during system

initialization. Storing the metadata in a disjunct location causes further issues, as the PTW must fetch the metadata during page table walks. This can only be achieved using a lookup table or a shadowed paging structure dedicated to locating the associated metadata. Both cases require additional memory transactions with potentially poor spatial locality. To avoid such issues, *FatPTE always co-locates PTEs and metadata within the same memory page.*

When co-locating mapping information with metadata, the layout of the PTEs and the associated metadata regions within a page can be chosen arbitrarily. However, a complex mapping function for the metadata region may have a detrimental impact on the system performance and would require additional logic in the PTW. It is, thus, favorable to keep the layout as simple as possible. When discussing the layout of PTEs and metadata regions within the same page, we use the term *index* to refer to their 64-bit aligned locations in the page. Thus, index 0 denotes the first byte on the page, index 1 the 8th byte, and so on. We consider two variants of how the FPTEs are stored within a page.

Split-Page Layout. In the first variant the upper part of the page exclusively accommodates legacy PTEs. Thus, this portion of the page is equivalent to the case of using unmodified PTEs. The metadata for each entry is located on the remaining portion of the page. We denote this type of layout as the *split-page layout* due to the content of the page being clearly divided. When accessing a FPTE, the PTE and the corresponding metadata are fetched from their respective parts of the page. While this layout has the advantage that the PTW can directly use the index extracted from the virtual address to access the FPTE, it also entails disadvantages. As the index of the metadata region is determined by the formula given above, the PTW must implement additional logic to determine the location from which the metadata must be fetched. Furthermore, separating the PTE and the metadata may cause additional latency due to low spatial locality as the metadata may reside in a different cache line.

Cache-Line Localized Layout. In modern systems, the memory subsystem operates on cache line granularity, which spans 512 bits. Thus, a single cache line can hold eight 64-bit legacy PTEs. For 64-bit metadata regions, a cache line can accommodate four FPTEs. For the 192-bit metadata regions, the number of FPTEs per cache line decreases to two. Organizing the PTEs and their associated metadata such that they are co-located in the same cache line has the advantage that all data can be *fetched in one single memory transaction.* Furthermore, the fact that the metadata is directly co-located to the PTE simplifies the function computing the indices in the PTW. By co-locating the PTE and the metadata in the same cache line, the additional pressure imposed on the memory subsystem is kept to a minimum, as no additional memory requests are required.

4.3 Affected Levels $\mathcal{P}_3$

Only using FPTEs on the last paging level limits the usage of security-relevant metadata to the case where default-sized pages are used. As modern systems

allow to directly map a leaf page instead of the next-lower table (*i.e.*, huge pages), one must decide on the layer up to which FPTEs are supported. We denote a configuration in which only the last-level page table supports FPTEs as a *PTL0* configuration. Consequently, a configuration supporting FPTEs in the last level and the level directly above is called a *PTL1* configuration (cf. Figure 1). This naming scheme continues for all layers of the hierarchy and we assume the configurations includes all lower levels.

PTL0 Configuration. The PTL0 configuration is the most straightforward configuration, as only one layer in the paging hierarchy is modified. As only the last-level PTEs are replaced by FPTEs, the PTW only requires additional fetches in the last translation stage in the case of the split-page layout. Also, the virtual address bit decrease is minimized as even a 192-bit PTE-MR only causes an overall decrease of two bits. However, a PTL0 configuration limits security schemes to requiring metadata on the smallest paging granule. Larger mappings consequently cannot utilize any of the additional bits, which may be a limiting factor for certain features.

PTL1 Configuration. With a PTL1 configuration, FPTEs are available on the second-to-last layer and on the last layer of the paging hierarchy. Thus, this configuration allows for the implementation of security schemes even in the case of huge pages. A single entry in the second-to-last layer maps a larger continuous virtual memory range than an entry in the last layer. Hence, the granularity on which this metadata is assigned is coarser than for PTL0 configurations. Security schemes that require fine-grained metadata may consequently not profit from PTL1 configurations. Furthermore, the memory overhead of such a configuration is also increased compared to PTL0 as the number of FPTEs increases. As the PTEs of the last two layers are modified in a PTL1 configuration, the amount of available virtual address bits decreases by at least two bits. In the case of 192-bit PTE-MR, the overall decrease sums up to four bits.

Depending on the implemented scheme, it may be possible that even more levels of the paging hierarchy use FPTEs. In such cases, the memory overhead increases with each additional level and the number of available virtual address bits decreases further. We do not consider implementing FPTEs on a higher level without including all levels below as this would introduce the same address space reductions while yielding a coarser granularity of metadata bits.

4.4 TLB Entry Size $\mathcal{P}_4$

As each FPTE consists of a legacy PTE and associated metadata, we must consider whether to increase the size of the TLB entries as well. Each TLB entry holds a mapping from a virtual page to the associated physical page and the accompanying permission bits. Furthermore, each entry holds a tag and potentially additional data.

The exact number of required additional metadata bits in each TLB entry is dictated by the implemented security scheme. If the metadata is only relevant

during the initial translation process, *i.e.*, when performing a page table walk, no additional TLB bits are required. If, however, the additional bits are checked on every access, the relevant bits must be stored in the TLB entries, thus increasing their size. This size increase necessitates a tradeoff between the overall TLB area and the number of TLB entries. As the TLB is implemented in hardware, its size is static and the TLB entry size cannot be configured during runtime. Consider the case in which the size of each TLB entry is doubled and assume that each entry only consists of storage and no additional logic. Keeping the number of TLB entries constant roughly doubles the overall storage space requirements of the TLB. Contrarily, doubling the entry size while halving the number of entries keeps the overall TLB size constant. However, reducing the number of TLB entries has a detrimental impact on the system performance as the number of page table walks due to TLB misses increases. At the same time, increasing the TLB size is increasingly costly due to its close proximity to the core and L1 caches. In our design, we consider both options to underline the tradeoff between performance and area overhead resulting from modifying the TLB.

5 Case Study

The main objective of FatPTE is to support as many security schemes depending on additional metadata located in the page table entries as possible. We perform a case study to showcase the applicability of FatPTE for commodity and academic security features. In our case study, we provide a detailed comparison of security features and demonstrate that their required metadata can be effectively stored using FPTEs. Furthermore, we analyze whether each scheme would benefit from an even larger number of metadata bits. For each security scheme, we investigate whether additional TLB bits are required and, if necessary, the number of additional bits. Additionally, we explore how the findings influence our choice of the design parameters $\mathcal{P}_1$ to $\mathcal{P}_4$.

Considered Applications. Various academic designs and commercial security features rely on metadata located in page table entries. These designs utilize PTE bits to associate metadata for *essential security technologies*, such as control-flow integrity (CFI), memory protection, and confidential computing. Table 1 provides a concise overview of the analyzed security technologies, which are discussed below. In our study, we discuss commercial security features and academic designs separately. A scheme is designated as a commercial product if it is present in commodity off-the-shelf CPUs or will be available in upcoming CPU generations. We do not consider schemes using tagged memory architectures [16], *i.e.*, store metadata in a dedicated tag storage, such as ARM MTE [31] as physical memory tagging is an orthogonal technology to implement security features.

5.1 Commercial Security Features

Intel Control-Flow Enforcement Technologies (CET) [33] introduce the indirect branch tracking (IBT) and shadow stack (SHSTK) features to protect forward-

Table 1: A comparison of commercial and academic designs that use PTE metadata to enable key security technologies.

	Mechanism	Application	ISA		# PTE bits
Commercial	Intel CET SHSTK [33]	Access Permission	x86-64		1-bit†
	ARM BTI	Legacy Compatibility	ARMv8.5		1-bit
	Intel MPK [21]	Protection Key	x86-64	⊕	4-bit
	ARM Domains [39]	Protection Key	ARMv7	⊕	4-bit
	Intel TME-MK [13]	Key Identifier (KeyID)	x86-64	⊕	15-bit‡
Academic	IMIX [9]	Access Permission	x86-64		1-bit
	Donky [29]	Protection Key	RISC-V	⊕	10-bit
	Multi-Tag [36]	Page-granular Tag	x86-64	⊕	16-bit
	PT-Guard [24]	Cryptographic MAC	x86-64	⊕	12-bit
	SecWalk [25]	Redundancy Code	RISC-V	⊕	25-bit

⊕ Profits from additional bits † Implicitly encoded ‡ Up to 15-bit

edge and backward-edge control-flow transfers. The shadow stack protects return addresses and, thus, mitigates attacks like return-oriented programming [6,32]. As the shadow stack holds security-critical data, it must be protected from unauthorized access. This is achieved by uniquely marking shadow stack pages using a previously unused combination of the read/write bit and the dirty bit in the PTE. Shadow stack pages can only be accessed using dedicated instructions, such as the `wrss` instruction. Other architectures, such as ARM and RISC-V, provide comparable features to enforce control-flow integrity. The RISC-V SHSTK also relies on memory protection using a new, previously unused, page type. This type is encoded by setting the page permissions in the PTE to write-only. The ARM guarded control stack (GCS) implements a shadow stack feature for the ARM architecture by leveraging a memory attribute for page protection. ARM branch target instructions (BTI) also provide landing pads for forward-edge CFI. However, ARM BTI leverages the GP-bit in the translation table to enable compatibility with legacy binaries. For all of the above schemes, it is essential to include the access permissions for the page in the TLB entry.

Apart from control-flow integrity, fine-grained and efficient control over page-granular access permissions can protect against memory safety issues. Intel memory protection keys (MPK) [21] leverage a 4-bit protection key encoded in the PTE to enforce read and write access policies during runtime. Memory protection is achieved by comparing the protection key with the user space protection keys register (PKRU), reflecting the currently active access permissions. PKRU implements read and write permission bits for each distinct protection key and is software-controlled from user space. MPK reduces performance overheads when partitioning the virtual address space through logical integrity checks, as no context switch is required to change permissions. ARM memory domains [39] introduce a 4-bit domain identifier for page-granular memory protection. As the

memory protection keys and the domain identifiers are checked on every access, they both must be included in the corresponding TLB entries.

Transparent memory encryption technologies like Intel Total Memory Encryption (TME) [11] and AMD Secure Memory Encryption (SME) [17] have been available in commodity CPUs for several generations. However, TME and SME only allow for one encryption key when encrypting DRAM data. Recent CPUs introduce new features, allowing for multiple encryption keys, thus paving the way for cryptographically isolated domains. Intel total memory encryption multi-key (TME-MK) [13] provides up to 15 key identifiers (keyID) bits located in the PTE for DRAM encryption. Intel TME-MK is used to encrypt guest memory for Intel's confidential computing technology [7]. To encode the keyIDs, TME-MK changes the specification of the physical address, *i.e.*, physical address bits are repurposed to encode the keyID into the PTE, thus reducing the overall addressable physical memory. The secure memory encryption (SME) feature of AMD uses a single bit in the physical address to indicate whether a page is encrypted or not. This *C-bit* thus reduces the available physical address size by one bit. Building on SME, AMD secure encrypted virtualization (SEV) [1] leverages the same C-bit to identify encrypted pages but provides per-VM encryption keys to cryptographically isolate virtual machines against each other and the host. As the keyID and the C-bit are part of the physical address, they are implicitly stored in the TLB entries.

We find that FatPTE increases the achievable security of certain commercial security features. Intel MPK is often criticized for allowing only 16 unique keys [30,29]. Schemes that use MPK for in-process isolation are thus constrained to 16 domains, which can be a limiting factor for their applicability. FatPTE facilitates the usage of more PTE bits for security and, thus, allows for an increased number of protection domains. This enables a more fine-grained compartmentalization of the application, resulting in increased security properties. Similarly, the security of ARM memory domains would improve by offering more domain identifiers. Consider, for example, a configuration with 64-bit PTE-MR. By providing 64-bit identifiers, we support a number of isolation domains that far exceeds the currently possible maximum of 16. In its current form, Intel TME-MK, which is essential for the cryptographic isolation of VMs, introduces a trade-off between the number keyIDs and addressable physical memory. While this approach works in current generations, future CPUs may require a larger number of keyIDs. As each additional keyID bit comes at the cost of halving the available physical address space, the maximum number of keyIDs is inherently limited. Through FatPTE, it is possible to increase the number of key bits without forfeiting any of the physical address bits.

5.2 Academic Designs

Security researchers proposed various security features that repurpose PTE bits [5,9,29,36,24,25]. Like the commercial features discussed above, academic schemes suffer from the same limitations due to the scarcity of spare PTE bits.

Donky [29] proposes the extension of MPK to use 10-bit protection keys on the RISC-V platform. Here, MPK greatly benefits from an increased number of protection keys, e.g., 10-bit protection keys enable the use of more than 1000 memory domains. Additionally, multiple designs leverage Intel TME-MK memory encryption for security beyond VM memory encryption [26,20,35,27]. However, the number of available keys is platform-specific. Schemes based on Intel TME-MK significantly benefit from an increased number of available keyIDs. Due to the steady increase in physical memory capacity, it is vital to allow for a large number of keyIDs without reducing the addressable physical memory.

Multi-Tag [36] proposes a multi-granular tagging strategy combining tagged memory with PTE-based tags to perform logical integrity checks for memory safety. As the integrity checks are performed at every access, Multi-Tag must use TLB entries for the page-granular tag bits. PT-Guard [24] repurposes 12 PTE bits for a cryptographic MAC that protects the integrity of the PTE itself. In the case of PT-Guard, the MAC is only required when performing the page table walk to detect tampering of the paging structures. Once the PTE is stored in the TLB, it is considered valid and does not require additional authentication. SecWalk [25] requires 25 bits in the PTE (thus significantly reducing the addressable physical memory) to provide fault protection with redundancy codes. Furthermore, SassCache [10] explores the use of available PTE bits for their secure randomized cache architecture.

Other academic designs rely on a single-bit in the PTE to enforce dedicated security policies and protect memory resources. IMIX [9] leverages one bit in the PTE for in-process isolation to mark security-critical memory. Similarly, CETIS [37] repurposes Intel CET SHSTK protection for in-process isolation. Cornucopia Reloaded [8] proposes the extension of one PTE bit to block capability loads, which is beneficial for implementing a fast memory sweep.

These academic designs highlight the strong need for more feature-specific PTE bits. Similar to commercial security features, FatPTE improves the security for several of the presented designs. While Donky proposed extending the protection key size from 4 to 10 bits, FatPTE allows for an even larger amount of protection domains. FatPTE also allows Multi-Tag to increase the number of tag bits beyond 16 bits without reducing the addressable physical memory. PT-Guard greatly benefits from FatPTE in terms of security and performance as it can compute a single MAC for each PTE, thus eliminating the need for additional fetches. When using 192-bit PTE-MR, the probability of a MAC collision becomes vanishingly small, thus providing exceedingly strong security against attacks. Similarly, SecWalk's security is significantly increased by allowing larger redundancy codes, resulting in stronger fault detection capabilities.

5.3 Insights

Our case study underlines the need for dedicated PTE bits for essential security features. In current CPU generations the spare PTE bits of major CPU manufacturers, such as Intel, are exhausted. Due to PTE size constraints, promising academic designs requiring additional PTE bits are hard to adopt for industry

use. Not only can FatPTE provide enough bits for all of the considered security features, *it even allows the implementing multiple features without compromising on their security.* Thus, FatPTE is an effective approach that facilitates easy adaptation and combination of commercial and academic security technologies. Furthermore, we find that 7 out of the 10 investigated security features benefit from additional PTE bits, thus underlining the relevance of adding security-specific bits to page table entries.

While the parameterization of FatPTE depends on the implemented security feature, we find that many of these features share common characteristics and requirements. We find that most schemes profit strongly from a 64-bit PTE-MR. The schemes with further security gains from using 192-bit PTE-MR are PT-Guard and SecWalk. Thus, we conclude that a 64-bit metadata region is feasible for all of the analyzed security features. All of the investigated security schemes can, in theory, operate on the granularity of huge pages. However, the effectiveness of schemes like Multi-Tag suffers when using larger mappings due to the coarser granularity of the metadata. We thus consider both a PTL0 configuration and a PTL4 configuration as feasible. However, note that none of the schemes requires the metadata to be present in all layers *at the same time*. The metadata is only required if a FPTE is a leaf entry, *i.e.*, if it points to a page and not another level in the hierarchy. Thus, even in the case of a PTL4 configuration, only one additional memory transaction for the metadata is necessary.

6 Implementation and Evaluation

To evaluate the performance impact of FatPTE, we create exemplary prototype implementations for x86-64 and RISC-V. Both variants implement 4 KiB pages and a 64-bit PTE-MR. It is not possible to include FatPTE's hardware changes on commodity x86-64 hardware. Thus, we use the gem5 system simulator [4] to implement FatPTE for the x86-64 ISA. We modify the PTW so that the number of index bits fits the number of entries in each level. In the case of the PTL0 configuration we change the indexing used for last-level pages such that the lowest level uses 8 instead of 9 index bits. We implement both the split-page and the cache-line localized layout. When using the split-page layout, our modified PTW fetches the metadata associated with a PTE by accessing the 64 bits located 256 indices lower on the same page. For the cache-line localized layout, the fetches are cache-line aligned and the fetch width increases to 512 bits. We implement the PTL4 configuration by modifying the PTW such that the index modification applies to all levels of the hierarchy.

Besides the gem5 prototype we also implement FatPTE for RISC-V using the CVA6 core [38], which uses 39-bit virtual addresses. We extend the core's PTW to support the additional fetches required by FatPTE and synthesize it for a Digilent Genesys 2 FPGA board. Here, we assume the cache-line localized layout with 64 bits of metadata, analogous to the x86-64 prototype. Our implementation supports configurable metadata fetching at all layers which we can control using a dedicated control and status register (CSR). The metadata of intermediate

layers can be checked directly after it was fetched and we store the metadata of the last translation layer in the TLB. We synthesize our prototype using a Digilent Genesys 2 FPGA board.

Depending on the feature implemented with FatPTE, it is necessary to extend the TLB entries by a certain number of bits. Thus, TLB entries increase in size as the amount of metadata bits increases. We implement two TLB configurations for the simulator prototype and the hardware prototype. The first configuration extends the size of each TLB entry by 64 bits while keeping the number of TLB entries constant, thus increasing the overall TLB area. The second configuration increases the TLB entry size while reducing the number of TLB entries to model the case in which the TLB area is limited.

6.1 Performance Evaluation

We evaluate the performance of FatPTE using the SPEC CPU 2017 benchmark suite. Our evaluation consist of multiple configurations to exemplify the design choices discussed above.

Simulation Setup. For our simulation, we model an x86-64 system with 4 KiB pages. We implement FPTEs on the lowest level of the hierarchy (PTL0) and on all levels (PTL4). Each FPTE holds 64 bits of metadata. We implement the cache-line localized and the split-page layout and investigate two TLB configurations. In the full-TLB configuration (TLB-F), the amount of TLB entries stays constant, which represents the case in which the TLB area increases. For the half-TLB configuration (TLB-H), we reduce the number of TLB entries by halving it, thus representing the case in which the area is limited. We base our gem5 configuration on the CPU model used by LeMay et al. [18] which represents current Intel Ice Lake CPUs. We use the out-of-order (O3) CPU model as this is the most accurate gem5 CPU model. We evaluate four variants of FatPTE (PTL0 TLB-F, PTL0 TLB-H, PTL4 TLB-F, PTL4 TLB-H). Our baseline system uses legacy PTEs, 128 iTLB entries, and 64 dTLB entries.

As the simulation duration using the O3 CPU can become infeasibly high for complex workloads, we use simpoints [22]. The simpoints tool identifies representative regions within the workload for its different execution phases. Simulating these regions and computing a weighted sum of the partial results provides a performance estimation for the complete workload. We use *cycles per instruction* (CPI) as the performance metric. Each simulation interval consists of a constant number of instructions. Thus, CPI values of different runs are normalized to the same instruction count.

Hardware Evaluation. We use our hardware prototype in two different configurations to evaluate the performance implications of FatPTE. Both models implement a PTL0 configuration and differ in the number of TLB entries. Our baseline features 64 iTLB and 32 dTLB entries and does not fetch metadata. While the TLB-F configuration leaves the TLB size unchanged, TLB-H reduces the number of iTLB entries to 32 and the number of dTLB entries to 16.

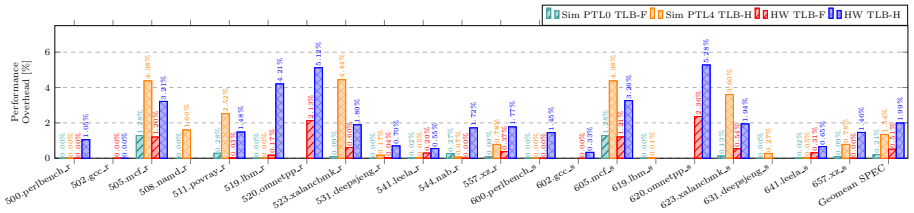


Fig. 3: The simulated and measured overheads using the SPEC CPU 2017 benchmark suite. The geomean overheads range from 0.21% to 1.99%.

For both evaluation platforms, we exclude SPEC CPU 2017 benchmarks that fail to compile, crash due to runtime issues, or do not finish within a reasonable amount of time (e.g., less than 100 hours).

6.2 Performance Evaluation Results

Figure 3 illustrates the relative performance overheads for our simulation and our FPGA prototype. For the simulation, we consider PTL0 TLB-F and PTL4 TLB-H using a cache-line localized layout. For our hardware prototype, we use a PTL-0 configuration with TLB-F and TLB-H. We select these configurations as they are the lowest and highest overheads, respectively.

We find that using FPTEs has an overall low impact on the system performance with geomean overheads ranging from 0.21% to 1.34% and most benchmarks experience overheads below one percent. For the benchmarks that show larger overheads, the performance impact is still within a reasonable range. We find that the number of TLB entries ($\mathcal{P}_4$) has a far stronger impact on the measured performance overhead than the paging levels on which the PTEs are extended ($\mathcal{P}_3$). This is mainly due to the fact that performing cache-line aligned requests synergizes with the granularity of the memory subsystem and the caching layers. Furthermore, reducing the number of TLB entries increases the amount of TLB misses and, thus, the need for costly page table walks. For the split-page layout, we observe larger overheads ranging from 1.04% for the (PTL0, TLB-F) configuration to 2.66% when using a (PTL4, TLB-H) configuration.

Similar to the simulation, the performance impact on the FPGA is low, with a geomean of 0.51% for TLB-F and 1.99% for TLB-H. Note that our hardware implementation requires an additional fetch during the page table walk to fetch the required metadata. While we use a cache-line localized layout, the hardware operates on word-sized granularity. As we run our benchmarks using a PTL0 configuration, the additional fetch is performed at the end of the walk. The given results suggest that FatPTE imposes feasible overheads, even in the case of resource-constrained systems with lower TLB capacity and smaller caches.

Table 2: FPGA Utilization of our modified CVA6 core. The TLB-Full configuration increases the number of lookup tables (LUTs) and flipflops (FFs). For TLB-Half, the total area decreases slightly compared to an unmodified system.

	iTLB	dTLB	LUTs	FFs
Base	64	32	Total 45626	29843
			TLB 7072	8990
TLB-F	64	32	Total 51700 (+13.3%)	36152 (+21.1%)
			TLB 11040 (+56.1%)	15134 (+68.3%)
TLB-H	32	16	Total 45544 (-0.2%)	28519 (-4.4%)
			TLB 3801 (-46.3%)	7566 (-15.8%)

6.3 Memory and Area Overhead

Increasing the size of PTEs also increases the memory footprint of the paging structures which scale with the amount of mapped virtual memory. We compute the relative overhead introduced by the different parameterizations assuming 64-bit PTE-MR metadata, 5-level paging, 4 KiB pages and that memory pages are mapped in virtually contiguous regions. Mapping a single page requires a PTE in each of the five layers and causes an overhead of 0.98%. As the amount of mapped memory increases, the overhead converges to a lower bound of 0.196%. When using FPTEs on the last layer (PTL0 configuration), the overhead converges to 0.391%. In a PTL4 configuration, this limit further increases to 0.392%. When using 192-bit PTE-MR, the overheads increase to 0.783% and 0.787% for PTL0 and PTL4, respectively.

For most of the security features analyzed in Section 5, the metadata bits are checked on every access and, thus, cached in the TLB. A TLB entry typically consists of the the PTE bits and additional information for TLB lookups and security checks. Keeping the number of entries constant while increasing the entry size increases the total area of the TLB. Alternatively, an area-constrained TLB can provide larger entries at the cost of a reduced overall number of entries. The CVA6 CPU uses 135-bit TLB entries. Adding a 64-bit PTE-MR increases the required bits by 47% but does not necessarily translate to an area increase of that same amount.

Table 2 shows synthesis results with different numbers of TLB entries. The values are relative to the unmodified CVA6 (*Base*). Most of the area overheads stem from caching the extra metadata in the TLB. The TLB itself grows by 56% and 68% in lookup tables (LUTs) and FlipFlops (FFs), respectively. When halving the TLB (TLB-H), the area stays roughly the same compared to the baseline. Hence, when the TLB is constrained by area, the number of entries would have to be halved to cache the full 64-bit PTE-MR.

7 Related Work

SPEAR-V [28] implements enclaves for RISC-V using a page-granular tagged memory architecture. In contrast to FatPTE, SPEAR-V associates metadata with physical memory pages by reserving an access-restricted fixed memory region. FatPTE associates metadata with virtual memory, thus scaling better in terms of memory overhead. Furthermore, FatPTE profits from increased spatial locality as metadata is co-located with the paging information.

Intel Sub-Page Write-Permission (SPP) allows granting write-access for 128-byte memory regions within a read-only page [15]. The used metadata is fetched on-demand since current PTEs cannot accomodate it. SPP metadata is accessed through a number of indirection layers which are traversed on every write access. Thus, SPP may cause non-negligible overheads in the case of frequent write accesses to protected pages. Note that FatPTE can directly encode such permission bits in the PTE metadata, thus eliminating the need for additional lookups.

ARMv9 introduced a new translation table format that increases each descriptor's size to 128 bits. Similarly to FatPTE, this allows for a larger addressable physical memory size and new attribute fields. However, the translation descriptors of ARM can only provide a subset of the available bits for security-critical features. In its current form, it is not possible provide one metadata bit for each cache line of a 4 KiB page. Thus, schemes that scale with the number of bits (e.g., PTGuard) may not be usable due to a loss of security. In contrast, FatPTE provides at least 64 metadata bits per PTE, thus allowing for at least cache-line granular metadata and the efficient implementation of schemes requiring a certain lower bound of bits for them to be effective. Furthermore, the PTE layout of FatPTE does not depend on the layer at which the entry resides.

8 Conclusion

In this work we introduced FatPTE, a design that facilitates the implementation of security features relying on metadata bits in page table entries.

As the number of spare PTE bits is exhausted the adoption of new features is hindered and the security of existing features is limited. FatPTE transforms PTEs to FPTEs by enhancing them with dedicated metadata for security-critical bits (PTE-MR). We identify the metadata region size, the metadata layout, the number of affected levels, and the modification of TLB entries as possible parameters for our design. By performing a case study on commercial and academic security features we uncover suitable parameterizations of FatPTE.

We implement an x86-64 software prototype using gem5 and a RISC-V hardware prototype to evaluate our design. Using the insights gained in our case study, we configure and benchmark multiple parameterizations of FatPTE using workloads of the SPEC CPU 2017 benchmark suite.

We find that the imposed performance overheads are low, ranging from 0.21% to 1.34% for our simulated prototype and from 0.51% to 1.99% on our hardware implementation. Furthermore, we show that our design imposes negligible page table storage overheads.

Acknowledgements. We thank the anonymous reviewers for their valuable feedback. This project has received funding from the Austrian Research Promotion Agency (FFG) via the SEIZE project (FFG grant number 888087) and the AWARE project (FFG grant number 891092).

References

1. Advanced Micro Devices: Strengthening VM isolation with integrity protection and more. White Paper, January (2020)
2. Advanced Micro Devices: AMD64 Architecture Programmer's Manual Volume 2: System Programming (March 2024), https://www.amd.com/content/dam/amd/en/documents/processor-tech-docs/programmer-references/24593.pdf, revision 3.42, Accessed: 2024-08-01
3. ARM: Translation granule, https://developer.arm.com/documentation/101811/0103/Translation-granule, version 1.3, Accessed: 2024-08-01
4. Binkert, N.L., Beckmann, B.M., Black, G., Reinhardt, S.K., Saidi, A.G., Basu, A., Hestness, J., Hower, D., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Altaf, M.S.B., Vaish, N., Hill, M.D., Wood, D.A.: The gem5 simulator. SIGARCH Comput. Archit. News **39**, 1–7 (2011)
5. Bratus, S., Locasto, M.E., Ramaswamy, A., Smith, S.W.: Traps, events, emulation, and enforcement: managing the yin and yang of virtualization-based security. In: Proceedings of the 1st ACM Workshop on Virtual Machine Security, VMSec 2008, Alexandria, VA, USA, October 27, 2008. pp. 49–58 (2008)
6. Buchanan, E., Roemer, R., Shacham, H., Savage, S.: When good instructions go bad: generalizing return-oriented programming to RISC. In: Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008. pp. 27–38 (2008)
7. Cheng, P., Ozga, W., Valdez, E., Ahmed, S., Gu, Z., Jamjoom, H., Franke, H., Bottomley, J.: Intel TDX Demystified: A Top-Down Approach. ACM Comput. Surv. pp. 238:1–238:33 (2024)
8. Filardo, N.W., Gutstein, B.F., Woodruff, J., Clarke, J., Rugg, P., Davis, B., Johnston, M., Norton, R.M., Chisnall, D., Moore, S.W., Neumann, P.G., Watson, R.N.M.: Cornucopia Reloaded: Load Barriers for CHERI Heap Temporal Safety. In: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024. pp. 251–268 (2024)
9. Frassetto, T., Jauernig, P., Liebchen, C., Sadeghi, A.: IMIX: In-Process Memory Isolation EXtension. In: 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018. pp. 83–97 (2018)
10. Giner, L., Steinegger, S., Purnal, A., Eichlseder, M., Unterluggauer, T., Mangard, S., Gruss, D.: Scatter and Split Securely: Defeating Cache Contention and Occupancy Attacks. In: 44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023. pp. 2273–2287 (2023)
11. Intel: Intel Architecture Memory Encryption Technologies. https://www.intel.com/content/www/us/en/content-details/679154/intel-architecture-memory-encryption-technologies-specification.html (08 2022), revision 1.4, Accessed: 2023-01-31
12. Intel: Intel Trust Domain Extensions. https://cdrdv2-public.intel.com/690419/TDX-Whitepaper-February2022.pdf (2022), accessed: 2024-05-27

13. Intel: Runtime Encryption of Memory with Intel Total Memory Encryption-Multi-Key (Intel TME-MK). https://www.intel.com/content/www/us/en/developer/articles/news/runtime-encryption-of-memory-with-intel-tme-mk.html (2022), accessed: 2024-05-27
14. Intel: Architecture Specification: Intel Trust Domain Extensions (Intel TDX) Module. https://cdrdv2-public.intel.com/733568/tdx-module-1.0-public-spec-344425005.pdf (2023), accessed: 2024-05-27
15. Intel: Intel® 64 and ia-32 architectures software developer manual (June 2024), https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html
16. Jero, S., Burow, N., Ward, B.C., Skowyra, R., Khazan, R., Shrobe, H.E., Okhravi, H.: TAG: Tagged Architecture Guide. ACM Comput. Surv. **55**, 124:1–124:34 (2023)
17. Kaplan, D., Powell, J., Woller, T.: AMD memory encryption. White paper **13** (2016)
18. LeMay, M., Rakshit, J., Deutsch, S., Durham, D.M., Ghosh, S., Nori, A., Gaur, J., Weiler, A., Sultana, S., Grewal, K., Subramoney, S.: Cryptographic Capability Computing. In: MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021. pp. 253–267 (2021)
19. Li, X., Li, X., Dall, C., Gu, R., Nieh, J., Sait, Y., Stockwell, G.: Design and Verification of the Arm Confidential Compute Architecture. In: 16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022. pp. 465–484 (2022)
20. Nasahl, P., Sultana, S., Liljestrand, H., Grewal, K., LeMay, M., Durham, D.M., Schrammel, D., Mangard, S.: EC-CFI: Control-Flow Integrity via Code Encryption Counteracting Fault Attacks. In: IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2023, San Jose, CA, USA, May 1-4, 2023. pp. 24–35 (2023)
21. Park, S., Lee, S., Kim, T.: Memory Protection Keys: Facts, Key Extension Perspectives, and Discussions. IEEE Secur. Priv. **21**, 8–15 (2023)
22. Perelman, E., Hamerly, G., Van Biesbrouck, M., Sherwood, T., Calder, B.: Using simpoint for accurate and efficient simulation. ACM SIGMETRICS Performance Evaluation Review **31**(1), 318–319 (2003)
23. RISC-V Foundation: The RISC-V Instruction Set Manual, Volume II: Privileged Architecture (2024), https://github.com/riscv/riscv-isa-manual
24. Saxena, A., Saileshwar, G., Juffinger, J., Kogler, A., Gruss, D., Qureshi, M.K.: PT-Guard: Integrity-Protected Page Tables to Defend Against Breakthrough Rowhammer Attacks. In: 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Network, DSN 2023, Porto, Portugal, June 27-30, 2023. pp. 95–108 (2023)
25. Schilling, R., Nasahl, P., Weiglhofer, S., Mangard, S.: SecWalk: Protecting Page Table Walks Against Fault Attacks. In: IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2021, Tysons Corner, VA, USA, December 12-15, 2021. pp. 56–67 (2021)
26. Schrammel, D., Sultana, S., Grewal, K., LeMay, M., Durham, D.M., Unterguggenberger, M., Nasahl, P., Mangard, S.: MEMES: Memory Encryption-Based Memory Safety on Commodity Hardware. In: Proceedings of the 20th International Conference on Security and Cryptography, SECRYPT 2023, Rome, Italy, July 10-12, 2023. pp. 25–36 (2023)

27. Schrammel, D., Unterguggenberger, M., Lamster, L., Sultana, S., Grewal, K., LeMay, M., Durham, D., Mangard, S.: Memory tagging using cryptographic memory integrity on commodity x86 cpus. In: 9th IEEE European Symposium on Security and Privacy, EuroS&P 2024, Vienna, Austria, July 8-12, 2024. IEEE (2024)
28. Schrammel, D., Waser, M., Lamster, L., Unterguggenberger, M., Mangard, S.: SPEAR-V: Secure and Practical Enclave Architecture for RISC-V. In: Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security, ASIA CCS 2023, Melbourne, VIC, Australia, July 10-14, 2023. pp. 457–468 (2023)
29. Schrammel, D., Weiser, S., Steinegger, S., Schwarzl, M., Schwarz, M., Mangard, S., Gruss, D.: Donky: Domain Keys - Efficient In-Process Isolation for RISC-V and x86. In: 29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020. pp. 1677–1694 (2020)
30. Schwarzl, M., Borrello, P., Kogler, A., Varda, K., Schuster, T., Schwarz, M., Gruss, D.: Robust and Scalable Process Isolation Against Spectre in the Cloud. In: Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part II. pp. 167–186 (2022)
31. Serebryany, K.: ARM Memory Tagging Extension and How It Improves C/C++ Memory Safety. login Usenix Mag. **44** (2019)
32. Shacham, H.: The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In: Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007. pp. 552–561 (2007)
33. Shanbhogue, V., Gupta, D., Sahita, R.: Security Analysis of Processor Instruction Set Architecture for Enforcing Control-Flow Integrity. In: Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy, HASP@ISCA 2019, June 23, 2019. pp. 8:1–8:11 (2019)
34. The kernel development community: HugeTLB Pages (2024), `https://docs.kernel.org/admin-guide/mm/hugetlbpage.html`
35. Unterguggenberger, M., Lamster, L., Schrammel, D., Schwarzl, M., Mangard, S.: TME-Box: Scalable In-Process Isolation through Intel TME-MK Memory Encryption. In: 32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025 (2025)
36. Unterguggenberger, M., Schrammel, D., Nasahl, P., Schilling, R., Lamster, L., Mangard, S.: Multi-Tag: A Hardware-Software Co-Design for Memory Safety based on Multi-Granular Memory Tagging. In: Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security, ASIA CCS 2023, Melbourne, VIC, Australia, July 10-14, 2023. pp. 177–189 (2023)
37. Xie, M., Wu, C., Zhang, Y., Xu, J., Lai, Y., Kang, Y., Wang, W., Wang, Z.: CETIS: Retrofitting Intel CET for Generic and Efficient Intra-process Memory Isolation. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022. pp. 2989–3002 (2022)
38. Zaruba, F., Benini, L.: The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology. IEEE Trans. Very Large Scale Integr. Syst. **27**, 2629–2640 (2019)
39. Zhou, Y., Wang, X., Chen, Y., Wang, Z.: ARMlock: Hardware-based Fault Isolation for ARM. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014. pp. 558–569 (2014)