PSSM: Achieving Secure Memory for GPUs with Partitioned and Sectored Security Metadata

Shougang Yuan
syuan3@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

Yan Solihin
Yan.Solihin@ucf.edu
University of Central Florida
Orlando, Florida, USA

Huiyang Zhou
hzhou@ncsu.edu
North Carolina State University
Raleigh, North Carolina, USA

ABSTRACT

In this paper, we investigate the secure memory architecture for GPUs and point out that conventional CPU secure memory architecture can not be directly adopted to the GPUs. The key reasons include: (1) accessing the security metadata, including encryption counters, message authentication codes (MACs) and integrity trees, requires significant memory bandwidth, which may lead to severe bandwidth competition with normal data accesses and degrade the GPU performance; (2) contemporary GPUs use partitioned memory organization, which results in storage and coherence problems for encryption counters and integrity trees since different partitions may need to update the same counter/integrity tree blocks; and (3) the existing split-counter block organization is not friendly to sectored caches, which are commonly used in GPU for bandwidth savings. Based on these observations, we propose partitioned and sectored security metadata (PSSM), which has two components: (a) using the offset addresses (referred to as local addresses) within each partition, instead of the virtual or physical addresses, to generate the metadata so as to solve the counter or integrity tree storage and coherence problem and (b) reorganizing the security metadata to make them friendly to the sectored cache structure so as to reduce the memory bandwidth consumption of metadata accesses. With these proposed schemes, the performance overhead of secure GPU memory is reduced from 59.22% to 16.84% on average. If only memory encryption is required, the performance overhead is reduced from 29.53% to 5.18%.

KEYWORDS

GPUs, secure memory, memory encryption, memory integrity, metadata cache

ACM Reference Format:
Shougang Yuan, Yan Solihin, and Huiyang Zhou. 2021. PSSM: Achieving Secure Memory for GPUs with Partitioned and Sectored Security Metadata. In *2021 International Conference on Supercomputing (ICS '21), June 14–17, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3447818.3460374

1 INTRODUCTION

Hardware supports for trusted execution environments (TEEs), such as Intel SGX [5, 10] and ARMTrustZone [24], have been integrated onto the CPUs to provide secure isolated computing environment for cloud users. TEEs can protect against both compromised system software, e.g., hypervisors and operating system (OS), and physical attacks such as memory tampering [10]. However, TEE support is missing on contemporary GPUs. As GPUs are widely used as accelerators in the cloud servers for many workloads such as machine learning, scientific computing, 3D rendering, etc., there is a growing need for strong security protection like TEE on GPUs.

Some recent works, including Graviton [33] and HIX [14], aim to provide TEE for workloads offloaded to GPUs. Graviton assumes that the system software stack, including GPU drivers, the OS, and hypervisor, cannot be trusted and the attackers have physical access to the hardware. To protect against software attacks, the GPU management operations are offloaded to the GPU command processors instead of being relegated to the GPU drivers, which run within the untrusted host kernel space. Furthermore, Graviton provides new primitives, such as secure memory copy, to prevent attackers from snooping upon the PCIe bus. HIX, alternatively, mainly focuses on protecting the I/O path between the hardware and software. Although it also keeps the GPU driver out of the trust boundary by isolating it from the kernel space, it relies on the secure CPU enclaves to protect the refactored GPU drivers.

A key observation is that the threat models of previous works like Graviton [33] and HIX [14] are significantly weaker than that of the CPU TEE. Specifically, they include the GPU device memory in the trusted computing base (TCB). The GPU device memory is left unprotected, i.e., no memory encryption or integrity protection are provided. As a result, the device memory is vulnerable to attacks, which may include eavesdropping or tampering of data in memory. Examples include passive eavesdropping between GPU and its device memory [32], cold boot attacks [11], rowhammer attacks [21], etc. In particular, Graviton assumes that the GPU device memory is soldered within the GPU package and thus assumes physical attacks on memory are out of reach. However, due to the high cost of 3D stacked memory, discrete GDDR memory remains to be widely used in commercial GPUs, including the latest Nvidia Ampere GPUs [2]. Discrete GDDR memory chips are readily accessible as it is a common practice to replace faulty GDDR-memory chips. Thus, GPU device memory will still be vulnerable to physical attacks. Moreover, with the new attack scheme like rowhammer [21], even the content of stacked memory can also be altered by attackers. Hence, we argue that secure memory support is needed for GPUs to provide strong confidentiality and integrity protection.

ICS '21, June 14–17, 2021, Virtual Event, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8335-6/21/06…$15.00
https://doi.org/10.1145/3447818.3460374

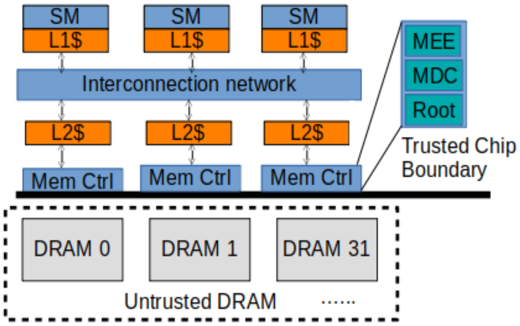

Figure 1: Secure GPU architecture with the trust boundary as the GPU chip

In this paper, we explore the architectural design of secure memory for GPUs. The overall secure GPU architecture is shown in Fig. 1. The GPU chip forms the trust boundary and each memory partition on the GPU chip integrates a memory encryption engine (MEE) [10], metadata caches (MDC), and a special register storing the root of the integrity tree.

One may consider adopting CPU secure memory architecture [4, 34, 35] for GPUs. However, we argue that CPU secure memory support cannot be directly adopted to the GPUs. As pointed out in our earlier study [38], the main reason is that the accesses to the security metadata, including encryption counters, message authentication codes (MACs) and integrity trees, often lead to bandwidth contention, which degrade GPU performance. This paper builds upon the findings from our previous study and leverages the recommended designs including the MDC architecture and the pipelined MEEs to further reduce the performance overhead. In particular, we make the following new observations: (1) GPU secure memory needs to support partitioned memory organization, which is used in contemporary GPUs [2] to achieve high memory bandwidth. As a result of partitioned memory, memory blocks are interleaved to different memory partitions and physical address based security metadata requires either cross partition communication and/or redundant storage, which worsens the bandwidth contention problem. (2) Sectored caches are often used in GPUs as a bandwidth saving technique but the security metadata, especially the split counters, are not well suited for such sectored organizations. (3) MAC verification presents a new trade off for sectored data caches. On one hand, if a MAC is computed for each cache line, all the sectors in a cache would be needed for MAC verification, defeating the purpose of sectored data caches. On the other hand, if a MAC is computed for each sector, higher MAC storage and bandwidth would be resulted. If we truncate the MAC to reduce the storage and bandwidth overhead, the strength of MAC verification might be compromised.

Based on our observations, we propose a simple yet effective scheme, partitioned and sectored security metadata (PSSM). PSSM has two components. First, to address the issues with partitioned memory organization, we propose to use the *"partition-local addresses"* (or simply local addresses) within each partition to generate the metadata. In partitioned memory organizations, physical addresses are mapped to partition ids and partition offsets using some carefully crafted hash functions. We refer to the partition offsets as partition-local addresses. Using local addresses to generate the security metadata essentially protects each partition independently: each partition has its own local counters and its own integrity tree with an on-chip root. Therefore, as shown in Fig. 1, a root register is added in each memory partition. Second, we propose to reorganize the metadata so as to make them friendly to the sectored cache structure. In addition, given the high performance impact from MAC accesses, we choose to have one MAC for each cache line instead of one MAC per sector. In other words, a sectored L2 cache is essentially forced to behave like a non-sectored one.

To summarize, the contributions of this paper are as follows:

- We show that conventional physical address-based metadata are not compatible with GPU partitioned memory. Naive adoption of physical address-based metadata results in metadata replication and/or cross partition communication.
- We propose to use the partition-local address to organize the encryption counters and build an integrity tree for each memory partition to address the problem.
- Based on the GPU memory access pattern, we propose to reorganize the security metadata and make them friendly to the sectored cache structure to reduce the memory bandwidth contention.
- We recognize that MAC accesses can be a major performance overhead and show that the design of one MAC per sector incurs higher performance overhead than one MAC per cache line, although it makes a sectored L2 cache to behave like a non-sectored one.
- We present detailed evaluation and show that PSSM can reduce the GPU secure memory performance overhead from 59.22% to 16.84% on average. If we only consider GPU memory encryption, the overhead is reduced from 29.53% to only 5.18%.

The remainder of the paper is organized as follows. Section 2 presents the background on secure memory and specifies the threat model. Section 3 discusses our main observations and motivations of our work. Section 4 introduces our architecture design. Section 5 evaluates the performance of our design. Section 6 discusses the related work. And Section 7 concludes our work.

2 BACKGROUND

2.1 Thread Model and Scope of Our Work

Hardware-based CPU TEEs, e.g., Intel SGX, assume two types of threats [10]: compromised system software (such as the OS and/or hypervisor) and physical attacks including bus snooping and memory scanning/tampering attacks. The processor chip forms the trust computing boundary (TCB) [19], and all on-chip components are assumed to be out of the reach for attackers. In other words, all the data going off the processor chip need to be protected. This protection scheme requires three major architecture supports [6]: hardware key management, remote attestation, and secure memory. Among them, secure memory incurs the highest performance overhead because it needs to be in operation all the time, encrypting/decrypting and verifying the integrity of all the data stored in off-chip main memory, and affects the critical-path delay of load instructions.

This paper assumes that the same physical attacks considered in CPU TEEs can also affect GPUs. In contrast to previous work [14, 33], we exclude the GPU memory module from the trust boundary, and assume that the GPU chip forms the security boundary. In other words, we consider that GPU memory is vulnerable to the conventional memory bus snooping and memory tampering attacks if the attackers have the physical control of the server.

Our focus is on exploring the secure memory architecture for GPUs. Hardware key management and remote attestation have been studied in previous work including Graviton [33], HIX [14] and Intel SGX [10]. Also, secure GPU context isolation and management as well as PCIe bus protection scheme have been addressed in previous works like Graviton [33], and we assume such support when designing our secure memory architecture for GPUs. Furthermore, side channel attacks [7, 22, 23] such as timing-based side channel attacks are out of the scope of our work.

2.2 Memory Encryption

Memory encryption is one of the key functionalities of secure memory. It performs encryption/decryption for all the traffic between last-level caches and off-chip DRAM. The state-of-the-art memory encryption uses counter-mode encryption. A counter value is maintained for each memory block/cache line. At the time when a data request is fetched from off-chip memory, the block address (physical address typically) and the counter value are combined to generate a one-time pad (OTP). And the memory controller recovers the plaintext with a single-cycle XOR operation. This encryption/decryption process is illustrated in Fig. 2.

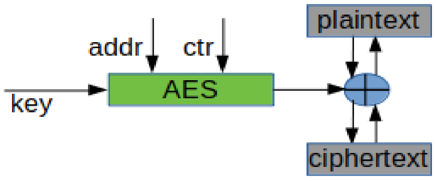

Figure 2: Counter-mode encryption

Counters in counter-mode encryption can be monolithic [10] or split [4, 34]. The benefits of split counters is that they have low storage overhead. With split counters, a major counter is shared by the memory blocks within a physical memory page, and one minor counter is maintained for each memory block. At the time of each memory (read/write) access, the major counter and the minor counter are concatenated to form the counter used for encryption/decryption. With monolithic counters, each memory block has one counter. Counter-mode encryption fundamentally relies on counter integrity protection to provide data confidentiality [34].

2.3 Memory Integrity

The integrity of the off-chip data can be protected by a combination of MACs and integrity/hash trees. MACs can protect against memory tampering attacks where value in memory is changed or moved to a different location, but the memory can still be impacted by the replay attack because the attacker can replay both data and its MAC to pass the MAC verification logic. In early designs [19, 30, 35], a Merkle/Hash tree built on top of the entire memory is used to protect against replay attacks. Later on, Rogers et al. [26, 34] proposed a stateful MAC, which includes the ciphertext, block address, and the counter value of the block into the MAC calculation, and reduces the replay attack surface to only the encryption counters. A variant of Merkle Tree, named Bonsai Merkle Tree (BMT), was proposed to cover the encryption counters to prevent counter-replay attacks and the BMT is much smaller and shallower than the hash/integrity tree built upon the memory data blocks. Any modification to the data or MACs stored in off-chip memory can be detected by comparing the MACs when the data is fetched from memory. Similarly, any modification or replay of the counters can be detected by traversing the BMT and comparing the hash values in the BMT nodes stored on chip, either the on-chip BMT cache or the on-chip BMT root.

Alternatively, a counter tree is used in Intel SGX [10] to protect the integrity of counters. The advantage of the counter tree is that the updates of the nodes along the update path from the leave to the root can be done in parallel, resulting in low latency of integrity tree updates at the cost of multiple MAC engines. As shown in Section 3, GPUs are not sensitive to the latency of cryptographic operations. Therefore, we use BMTs in this work.

3 MOTIVATIONS

3.1 Performance Impacts of Naive Design

To pinpoint the performance bottlenecks of adopting CPU secure memory to GPUs directly, we first perform a detailed performance analysis. The simulation methodology is presented in Section 5.1. We model the secure memory architecture with split-counter mode encryption, combined with MAC and BMT for integrity protection. A 64KB cache is added for each type of security metadata. There are 32 memory partitions in our baseline GPU model, thus each memory partition is equipped with a 2KB counter cache, a 2KB BMT cache, and a 2KB MAC cache. We assume the metadata cache (MDC) line size of 128B, the same as the data cache line size. A split-counter block with 128B consists of 1×128-bit major counter and 128×7-bit minor counters (SC_128) is adopted in this naive design, as shown in Fig. 7. This way, one counter block is used for 128 data blocks or a major counter is shared among 128 data blocks.

Fig. 3 reports the instruction per cycle or IPC (higher is better) of secure memory designs normalized to the baseline GPU without secure memory support. In the figure, the design directly adopted from CPU secure memory is labeled **secureMem**. Two idealistic (but infeasible) designs are also included to determine performance bottlenecks. **large_mdc** represents an ideal design with unlimited MDC capacity, while **0_crypto** represents an ideal design with zero-latency cryptographic operations (i.e., zero encryption and MAC computation latencies). In all these designs, the metadata are generated using the physical addresses. In other words, the addresses used in counter mode encryption and MACs are physical addresses of the data blocks.

From Fig. 3, we can see that directly adopting the CPU secure memory design results in a performance slowdown for GPU of 59.66% on average (geometric mean). The performance degradation is even higher for memory-intensive benchmarks like lbm (91%)

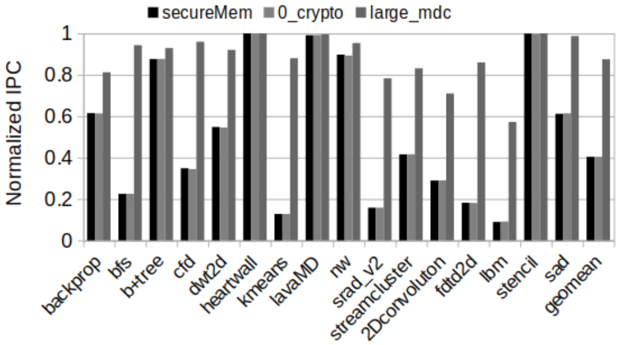

Figure 3: Normalized performance of secure memory designs to the baseline GPU without secure memory support.

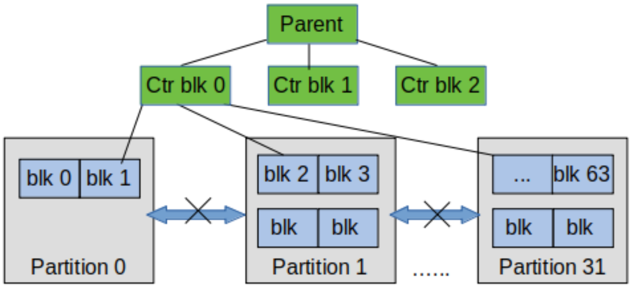

Figure 4: A single (128B) counter block corresponds to 128 data blocks in 32 partitions. Similarly, one BMT node requires multiple counter blocks.

and srad_v2 (84%). Zero cryptographic latency does not improve the performance of secure memory, primarily because GPUs being designed to be latency tolerant. On the other hand, with unlimited MDC sizes, the average performance is more than doubled and approaches the baseline GPU without secure memory, indicating that the memory accesses to fetch/store security metadata from/to off-chip memory is the main performance bottleneck. Moreover, we can observe that even with unlimited MDC capacity, where only cold misses on metadata remain, the performance slowdown can still be significant, 13% on average. The reason is that the performance of GPUs is highly bounded by memory bandwidth and we would better avoid any additional memory bandwidth contention.

3.2 Problem Diagnosis

As presented in Section 3.1, even with unlimited MDCs, there is a performance gap between the baseline GPU and the one with secure memory support. By analyzing the MDCs in different partitions, we discover that the partitioned memory structure and memory interleaving used in contemporary GPUs lead to redundant metadata being fetched to and stored in the MDCs in different partitions. This implies that some memory bandwidth is wasted due to unnecessary data transfers. The main reason for the redundant metadata among different memory partitions is that the state-of-the-art split-counter mode encryption organizes the counters based on data blocks' physical addresses. However, with partitioned memory structure, memory blocks within the same physical page are mapped to different memory partitions so as to avoid the partition camping problem [1][36]. In our baseline GPU, pseudo random memory interleaving is employed and the interleaving granularity is 256B (2 memory blocks or cache lines with the block/line size of 128B). With split counters, one counter block contains the counters for many data blocks mapped to different partitions. Since each memory partition has its own memory controllers and cannot access other partitions directly, there is a question of how to maintain the split counters.

The problem is further illustrated with Fig 4. Here, we simply assume the sequential interleaving scheme and the interleaving granularity is 2 memory blocks. With split counters, one 128B counter block protects 128 data blocks and these data blocks are distributed across the 32 memory partitions. As a result, the same counter block needs to be accessed by all the 32 memory partitions and it may be stored in the counter cache in each partition, leading to significant redundancy. Besides the redundancy in counter caches, another key problem is, which memory partition should be used to accommodate this counter block in off-chip memory? There are two options.

Option 1: Redundant counters We can store several copies of the counter block in different partitions such that each partition can access the its own copy. For example, a copy of counter block 0 can be stored in off-chip memory corresponding to partition 0, another stored in partition 1, etc. However, the issue is the coherence among the multiple copies of the same counter block. If there is one minor counter overflow in one partition, the major counter would need to be updated for all the copies of the counter block. But there is no communication channel across different partitions to support such an operation. The remedy would be adding an interconnect network across different partitions, which would incur high hardware cost.

Option 2: Single copy of counters We can also store all the metadata in one partition. For example, all counter blocks can be stored in partition 1. When partition 2 received a memory request, which needs to access its counter block, the memory controller in partition 2 would have to access partition 1, thereby requiring the interconnects across different partitions.

Note that the BMT nodes share the same problems as the counters since the BMT is built on top of the counter blocks. In other words, a BMT node needs to be accessed by multiple partitions.

3.3 Coarse-Grain Interleaving

To solve the counter redundancy and coherence problem, one possible solution is to enlarge the interleaving granularity to larger memory chunks like page-level memory interleaving [39]. For example, if four consecutive memory pages in the physical memory space (assuming 4KB pages) can be assigned to the same memory partition, all the 128 data blocks (16kB = 128*128B) corresponding to a 128B counter block would reside in the same memory partition. However, the problem of such coarse interleaving granularity is *partition camping* [1], which means that multiple streaming multiprocessors (SMs) may try to access the data from the same memory partition, resulting in contention at the partition and low memory bandwidth utilization overall.

To evaluate the performance impact of coarse interleaving granularity, we model the memory interleaving at 1-page and 4-page granularity without secure memory and normalize the performance to the IPC of baseline GPU without secure memory. The results are shown in Fig. 5. As we can see from Fig. 5, 1-page interleaving slows down GPU performance by 13% on average compared to the baseline GPU, which uses 2-block/256B interleaving. Moreover, 1-page interleaving cannot solve the counter storage problem as one counter block still contains the counters from more than one partition. In comparison, 4-page interleaving can eliminate the counter storage problem entirely, as all the data blocks, whose counters are in the same counter block, reside in the same partition. The overheads, however, are increased: 4-page interleaving reduces GPU performance by almost 29% on average due to more severe partition camping.

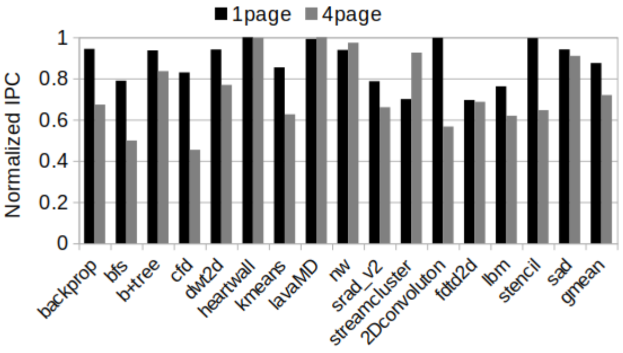

Figure 5: The IPC of different page interleaving granularities, normalized to the baseline GPU with 256B interleaving and without secure memory support.

Even with coarse-grain interleaving, the problem with the BMT remains. Some BMT nodes, especially those in high levels of the tree, are still needed by different partitions, as one BMT node may have several children nodes that span over multiple partitions.

3.4 Sectored MDC

Sectored caches are commonly used for commercial GPUs to reduce memory bandwidth consumption. As pointed out in Section 3.1, the key performance bottleneck of GPUs with secure memory support is the bandwidth contention due to metadata accesses. Therefore, we expect that the GPU performance can benefit from sectored MDCs. In our baseline GPU, the data caches (both L1 and L2) use 128B cache lines and each cache line has 4 sectors, i.e., each sector has a size of 32B. Similarly, we model the MDC with 4 sectors in each cache line. Fig. 6 shows the performance comparison of secure GPU with sectored and regular (i.e., non-sectored) MDCs of the same capacity and set associativity. As expected, we can clearly see that the performance of sectored MDC is significantly better than the non-sectored MDC. As such, we conclude that sectored MDCs are preferred for GPU secure memory design.

However, there is a problem with separating split-counters into sectors as shown in Fig. 7. With a 128B counter block being divided into 4 sectors, the major counter (128b) along with 18 minor counters (7b each) is usually stored in the first sector (32B) while the remaining minor counters are stored in other sectors. If an L2 cache miss leads to a counter access to the sector other than the first, the memory controller still needs to issue 2 memory requests to fetch 2 sectors containing the major counter and the corresponding minor counter to recover the counter value needed for memory encryption/decryption. To avoid such additional memory transactions, the counter block organization needs to be redesigned such that it can be friendly to the sectored cache structure.

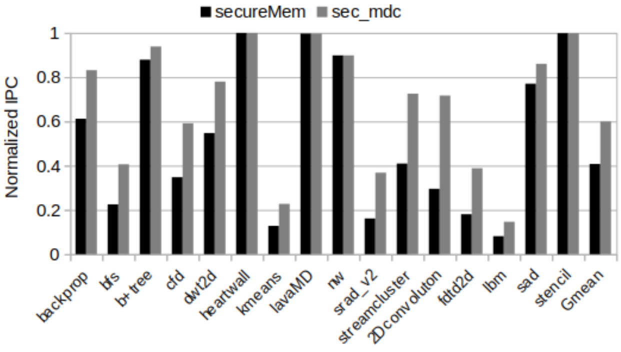

Figure 6: Performance comparison of secure GPU with non-sectored MDCs (labeled 'secureMem') and sectored MDCs (labeled 'sec_mdc').

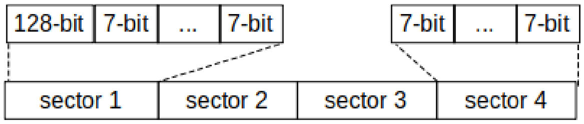

Figure 7: A Split-counter block of 128B, containing 1×128-bit major counter and 128×7-bit minor counters. When split into 4 sectors, the first sector contains the major counter and some minor counters.

3.5 Sectored Data Cache and MAC Verification

In CPU secure memory, a MAC is calculated for each cache line. On GPUs, sectored data caches are commonly used to save bandwidth consumption. However, secure memory presents a new trade off between regular data bandwidth and metadata bandwidth consumption. On one hand, if a MAC is generated for each line, MAC computation and verification would require all the sectors in the cache line, which would force a sectored cache to operate like a non-sectored one. On the other hand, if one MAC is used for each sector, the amount of MAC data will be highly increased, which would lead to high MAC data storage and bandwidth overheads. One partial solution is to truncate the MAC size for a sector. Although this might compromise the strength of MAC verification, we consider the performance impact of such MAC truncation in our evaluation.

As the design of one MAC per cache line would essentially make a sectored cache to behave like a non-sectored one, we examine the performance difference between the sectored and non-sectored caches for GPUs without secure memory. In our experiment, we compare a non-sectored L2 cache with a sectored L2 while keeping

L1 caches as sectored. The reason is that sectored L1 caches help reduce bandwidth pressure on the L1-L2 interconnect network. We report the IPCs of non-sectored L2 cache designs normalized to the sectored L2 cache design in Fig. 8. The label 'nL2_N' denotes the model of non-sectored L2 cache where each L2 MSHR (miss status handling register) can merge up to N requests that missed in the L2 cache and each request is a cache line. In our baseline GPU with sectored L2 cache, each L2 MSHR can merge 4 L2 miss requests and each request is one sector (i.e., 32B) rather than a cache line. The structure of such a MSHR is illustrated in Fig. 9 (a), where the primary miss (labeled 'priMiss') is the first request to the cache line/sector and the secondary misses (labeled 'secMiss') are the merged requests. If a request finds a matching MSHR but all its entries have been occupied, the request will be stalled and block the subsequent requests.

As we can see from Fig. 8, the performance of non-sectored L2 is very close to it of the sectored L2 for most benchmarks when N is 16 or larger. For N being 4, the sectored L2 has significantly higher performance for several memory intensive benchmark. The reason is that the non-sectored L2 design suffers many more MSHR stalls due to the merging granularity. Considering a streaming-like access sequence, A1, A2, A3, A4, A5, all of which are misses and map to different sectors in the same cache line: A1 and A5 to sector 1, A2 to sector 2, A3 to sector 3, and A4 to sector 4. With a sectored L2, A1, A2, A3, and A4 reside in different MSHRs and A5 merges with A1, as illustrated in Fig. 9 (b). With a non-sectored L2, A1, A2, A3, and A4 reside in the same MSHR and A5 is blocked since the matching MSHR is fully occupied, as illustrated in Fig. 9 (c). After fixing this MSHR stall with higher N values, the performance of the non-sectored L2 is very close to the sectored one. One exception is benchmark kmeans, for which sectored accesses reduce the overall bandwidth and a sectored L2 has better performance as a result of poor spatial locality of the benchmark. On the contrary, the benchmark lbm exhibits strong spatial locality and the non-sectored L2 designs show higher performance than the sectored one.

In summary, we can see that although a sectored L2 is beneficial for certain workloads, the performance of a non-sectored L2 cache is very close the sectored one on average. Note that for a sectored cache with one MAC per cache line, the MSHRs would not be a bottleneck since the requests are processed at the sector granularity.

4 ARCHITECTURE DESIGN

4.1 Overall Architecture

From our performance study in Section 3, we observed that CPU secure memory scheme cannot be directly adopted to GPU without losing much performance. To adapt the secure memory architecture design for GPUs, we propose partitioned and sectored security metadata (PSSM). PSSM has two simple yet effective components. First, it uses the partition-local addresses, which are the offsets within a partition, instead of physical addresses to construct the security metadata. Second, it reorganizes the split counter blocks to make them friendly to sectored caches. Our overall architecture design is shown in Fig. 1. With partitioned memory, each memory partition has its own memory controller. The memory encryption engine (MEE) [10] and MDCs are integrated into the on-chip memory controller. PSSM eliminates the metadata redundancy and coherence problem, and thus we can store the security metadata locally in each memory partition. There is also no need for cross-partition communication.

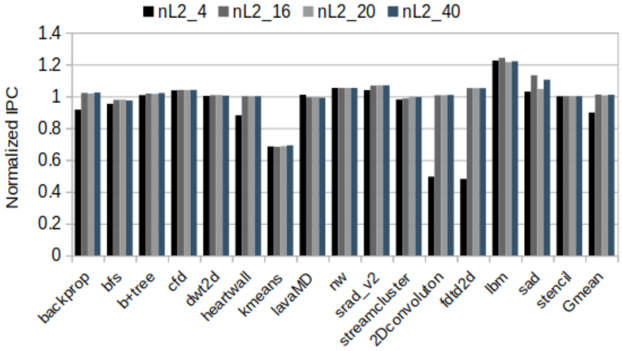

Figure 8: The IPC of a non-sectored L2 cache with different numbers of request merges in an L2 MSHR normalized to the baseline sectored L2.

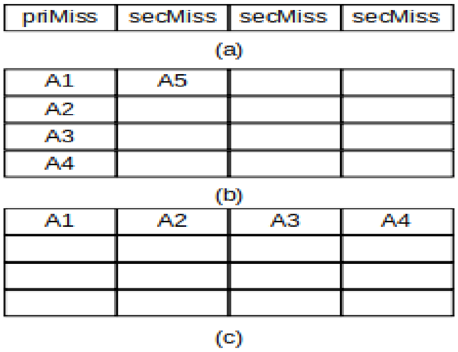

Figure 9: The L2 MSHRs. (a) The structure of an L2 MSHR. (b) The MSHR state of a sectored L2 after the access sequence A1-A5. (c) The MSHR state of a non-sectored L2 cache after the same access sequence A1-A5.

The MEE operates as an extension to the memory controller. It contains the AES encryption engines and hash/MAC engines. All the L2-to-DRAM requests are forwarded to the MEE, and the MEE will encrypt/decrypt data before sending/fetching it to/from the off-chip memory. To verify data integrity, the MEE will also generate additional memory transactions to validate the MAC for each data block, and traverse/update the integrity tree. A special register (labeled as **Root** in Fig. 1) is used for the root of the integrity tree. Also, a special off-chip memory region is reserved in each memory partition to store the security metadata including counters, MACs and intermediate integrity tree nodes.

4.2 Using Local Addresses for Security Metadata

With a partitioned memory structure, each memory access will be mapped to a partition. If this access misses in the L2 cache banks of the partition, the access goes to the off-chip memory through the memory controller of the partition. There is a mapping function, which converts the physical address into a partition id and a partition offset. In our design, we propose to use the partition offset, which we refer to as local address, to generate/organize the security metadata.

In Fig. 10, we illustrate the difference between physical and local addresses with an example of sequential interleaving across 32 memory partitions and the interleaving granularity is 2 memory blocks. With a 4KB page size, 32 memory blocks within the same physical page are mapped to 16 partitions, i.e., blk0, blk1, ..., blk15, where the blkids are physical addresses. With sequential interleaving, the partition id can be computed as (physical address / partition granularity) % number of partitions or blkid/2%32 whereas the partition offset is (physical address / number of partitions / partition granularity) * partition granularity + physical address % partition granularity), or (blkid/64)*2 + blkid % 2, where '/' is integer division and '%' is remainder. PSSM uses local addresses for security metadata. As shown in Fig. 10, a local page contains 32 blocks with consecutive local addresses while their physical addresses are not consecutive.

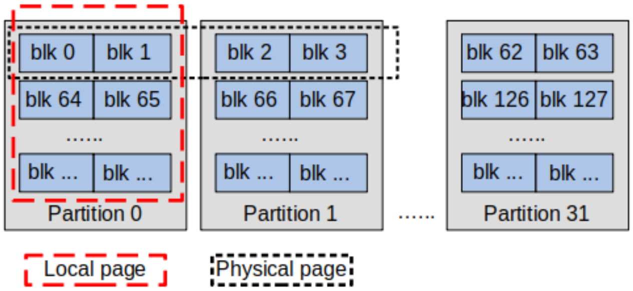

Figure 10: Physical and local addresses in partitioned memory organization.

Using local addresses to organize the counter blocks, memory blocks within the same local page will share one counter block. PSSM stores the counter blocks locally within each partition memory. Therefore, the BMT is also constructed solely based on the counter blocks within the same partition. In other words, each partition has its own BMT with the root stored in the corresponding on-chip memory controller.

The benefits of PSSM are: (1) correctness are ensured because all the metadata redundancies (counter/bmt node) are eliminated and there are no coherence problems, and (2) smaller and shallower integrity trees compared to a single BMT for all partitions.

Our baseline GPU uses pseudo random interleaving [25] memory. There are also other memory interleaving schemes, e.g., sequential interleaving, prime-module interleaving [18], skewed-interleaving [12], etc. A common feature of memory interleaving is the nature of bijection [25], which means that with local addresses and partition ids, the corresponding physical addresses can be computed, and vise versa. Also, the cost of address transformation between a physical address and a local address is usually minor.

4.3 Making Metadata Friendly to Sectored Caches

As discussed in section 3, sectored MDCs are preferred than non-sectored ones because their bandwidth-saving effects. A MAC cache line and a BMT cache line can be easily broken into smaller sectors since the MAC size and the hash values are a multiple of bytes (e.g., 8). A counter cache line or a counter block, however, is not friendly to sectored caches as discussed in Section 3.4. In PSSM, a single major counter is divided into multiple major counters and a major counter is shared by a smaller number of minor counters. Taking a 4-way sectored block as an example in Fig. 11, for each sector of 32B, there is a 32-bit major counter and 32 7-bit minor counters and we refer to this counter block design as SC_32. In other words, one sector in a counter block corresponds to 32 data blocks.

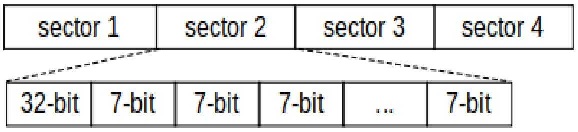

Figure 11: Sectored split-counter design: each sector has $1 \times$ 32-bit major counter and 32×7-bit minor counters.

Our sectored counter design is inspired from the memory write characteristics of GPU applications. In Fig. 12, we present the distribution of data access instructions, including shared memory read (labeled 'load_sm'), shared memory write (labeled 'store_sm'), global memory read (labeled 'dram_read'), and global memory write (labeled 'dram_write'). The results clearly show that global memory writes only account for a very small portion of overall memory access instructions. As a result, although PSSM uses a smaller major counter (32 bits) than the design in Fig. 7 (128 bits), there would not be a problem with the potential major counter overflows. Moreover, compared to the design in Fig. 7, the PSSM counter block design in Fig. 11 has lower overhead of a minor counter overflow: each minor counter overflow leads to 32 blocks to be re-encrypted rather than 128 blocks in Fig. 7.

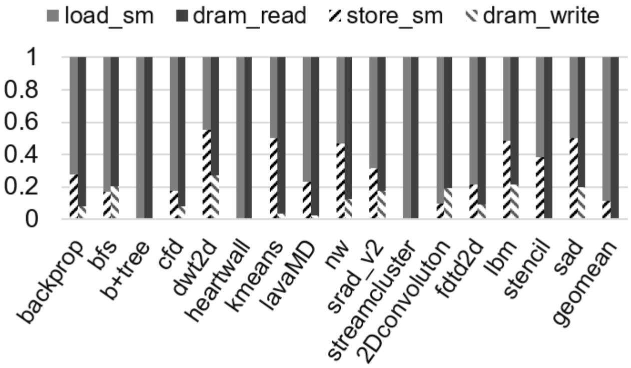

Figure 12: The distribution of different memory access instructions.

4.4 Encryption and MAC Engine

In counter-mode encryption, the choice of pad value is critical for security because pad reuse can lead to information leakage. In other words, the pad value must be unique. In conventional CPU secure memory, to ensure the temporal uniqueness, which means that pads are unique over time for each memory block, a counter value is maintained for each memory block as a pad component and is incremented on each write back. To maintain the spatial uniqueness, the physical block address is also included to form the encryption pads.

In PSSM, the encryption/decryption is performed at the granularity of a cache sector. It can be illustrated with Fig. 13. However, given that local addresses can be the same across different memory partitions and lead to pad-reuse, PSSM includes the partition id and sector id into the encryption pad. Let us use P as the plaintext, C as the ciphertext, and K as the secret key of AES. The memory encryption can be denoted as

$$C = E_K(local\_addr||Ctr||pid||sec\_id) \oplus P$$

where $local\_addr$ is the local address of the memory block, Ctr is the combination of major counter and minor counter, pid is the memory partition id, sec_id is the sector id within the cache line. The size of each input field is shown in Fig. 13. Depending on the number of memory partitions and partition granularity, the sizes of each field for the encryption input are adjusted accordingly.

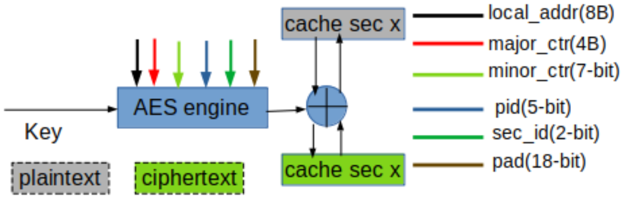

Figure 13: The encryption/decryption process in PSSM. The input to the AES encryption engine ensures encryption seed uniqueness, both temporally and spatially.

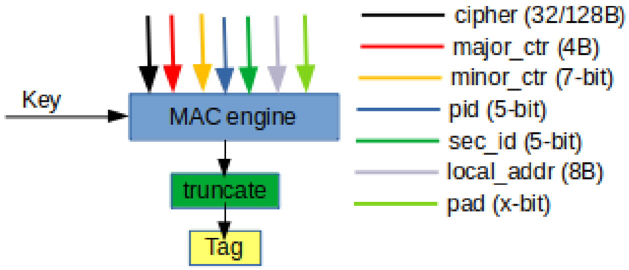

Figure 14: The MAC generation process in PSSM. The MAC computation output is truncated to 64/32 bits. Sector id is used when a MAC is generated for each sector.

In PSSM, the MAC may be calculated based on a cache sector or a cache line. The input fields and MAC calculation process are illustrated with Fig. 14. PSSM includes the sector id, partition id, and the local address into the MAC calculation to make the MAC location dependent. Note that depending on what MAC algorithm being used, the padding bits of the MAC engine can be different.

Table 1: Baseline GPU Configuration

SM config	80 SMs, 1132 MHz
Register File	256KB/SM, 20MB in total
L1 D-Cache	32KB/SM
Shared Memory	96KB/SM
L2 cache	2 banks per memory partition, each L2 cache bank is 96KB, 6MB in total
DRAM	850MHz, 32 partitions, 868GB/s, pseudo random memory interleaving.

4.5 Bandwidth for Accessing MACs

Among different types of metadata, accessing MACs incur high memory bandwidth (See Section 5.2). To address this performance bottleneck, PSSM may opt to truncate the MAC value to a smaller size. In our default setup, a MAC of 8B is used for each 32B sector/128B cache line. In our experiments, we evaluate the performance when we truncate the MAC value to 4B. We think that the truncated MAC is sufficient for GPU security for a few reasons. First, any random modification only has a very small chance of producing a hash collision due to the nature of computation resistance of the underlying hash function. With 4B MAC, an attacker only has $\frac{1}{2^{32}}$ or less than 1 in a billion chance to successfully bypass the MAC verification by randomly changing any bits of the data in off-chip memory. Second, unlike CPUs which may run long-running server applications, most GPU applications execute short-running kernels. Every instance of kernel execution uses a different session key hence the attacker cannot succeed across different kernel executions. Hence, the attacker must succeed in producing a hash collision within a single kernel lifetime. Some attacks, such as rowhammer, takes a while to succeed, e.g., 0.64 second on average to flip a single bit in the RAMbleed attack [8]. Therefore, as long as a single kernel executes for less than 1 hour, the chance of thousands of bit flips producing a hash collision is still much less than one in a million. Furthermore, once MAC mismatch is detected, GPU will be rebooted hence the attacker for practical purposes can only modify memory once before detection.

5 EVALUATION

5.1 Methodology

We evaluate our designs with GPGPU-Sim v4.0 [17]. Our baseline GPU configuration is shown in Table 1, which is modeled based on the Nvidia Volta architecture [15].

In our experiments, we assume that a range of 4GB device memory is protected. The detailed MDC and MEE organization is listed in Table 2. The MDCs are sectored by default unless otherwise specified. The different secure memory designs that we evaluate in our experiments are listed in Table 4. In one of our experiments, we also relax our threat model and only focus on GPU memory encryption without integrity protection. Table 5 lists the schemes we

Table 2: MDC and MEE Organization

Counter cache	2KB / memory partition, 128B blk, 4-way sectored, 256 MSHRs, allocate-on-fill policy
Mac cache	2KB / memory partition, 128B blk, 4-way sectored, 256 MSHRs, allocate-on-fill policy
Bonsai Merkle Tree cache	2KB / memory partition, 128B blk, 4-way sectored, 256 MSHRs, allocate-on-fill policy
Hash/Mac latency	40 cycles
AES engines	1 pipelined AES/memory partition

Table 3: Benchmarks

Categorization	Benchmark name	Bandwidth utilization
non memory intensive	heartwall	<1%
	lavaMD	<1%
	stencil	<1%
	sad	5%-7%
	nw	<2%
	b+tree	12%-14%
medium memory intensive	backprop	25%
	cfd	15%-50%
	dwt2d	20%-50%
	kmeans	40%-45%
memory intensive	bfs	5%-60%
	srad_v2	79%- 80%
	streamcluster	78%-80%
	2Dconvolution	53%
	fdtd2d	82%-83%
	lbm	58%

evaluate for GPU memory encryption, including different designs using split-counters and one using monolithic counters.

We use 16 benchmarks from a wide range of benchmark suites including Rodinia 3.1 [3], Parboil [28] and polybench [9]. Table 3 elaborates the details of these benchmarks. For each benchmark, we simulate 4 million cycles. The reason is that previous works [20] have pointed out that the variation of statistic counters becomes very small after the kernel is simulated for 2 million cycles for these benchmarks. Table 3 also classify the benchmarks based on their bandwidth utilization when running on the baseline GPU without secure memory support. We report normalized IPCs in our evaluation with the baseline as the GPU with sectored data caches and without secure memory support.

5.2 Performance

Overall Performance We evaluate our PSSM designs for both one MAC per sector and one MAC per cache line MAC with different MAC sizes, and report the performance results (normalized to the baseline GPU) in Fig. 15. From the figure, we can make the following observations. First, compared with the baseline secure memory design, labeled 'secureMem', our PSSM scheme improve the performance significantly. The average performance overhead is reduced from 59.22% to 42.03% for PSSM_sL2_8B_sMdc, 31.09% for PSSM_sL2_4B_sMdc, 19.06% for PSSM_nL2_8B_sMdc and 16.84% for PSSM_nL2_4B_sMdc. Second, a sectored L2 with one MAC per cacheline outperforms sectored L2 caches with one MAC per sector MAC. The reason is that the MAC storage overhead of the one-MAC-per-sector design is much higher (N times) that of one-MAC-per-cacheline design, where N is the number of sectors in a cache line. Consequently, the bandwidth requirement of MAC accesses is much higher in one-MAC-per-sector designs, leading to more severe memory bandwidth contention. Third, truncating the MAC size from 8B to 4B further reduces the memory bandwidth contention, resulting in a 2.2% reduction in performance overhead on average.

Table 4: Evaluated designs for GPU secure memory with both memory encryption and integrity verification.

Scheme	What It Represents
secureMem	Baseline GPU with secure memory, and the security metadata is organized with physical address. Sectored L2 cache and 2B MAC per sector.
PSSM_sL2_xB_sMdc	secure GPU memory with our PSSM design, the L2 cache is sectored, and the MAC is **x** bytes per sector.
PSSM_nL2_xB_sMdc	secure GPU memory with our PSSM design, the L2 cache is sectored (but behaving like non-sectored) since the MAC is **x** bytes per cache line.

Table 5: Evaluated designs for GPU memory encryption.

Scheme	What It Represents
SC_128_nMdc	Encrypted GPU memory with split counters. The major counter is 128-bit and the encryption counters are organized with physical address and the counter cache is non-sectored.
PSSM_Mono_Ctr_sMdc	Encrypted GPU memory with 32-bit monolithic counters. The counters are organized with local addresses and the counter cache is sectored.
PSM_SC_128_nMdc	Encrypted GPU memory with split counters. The counters are organized with local addresses. The major counter is 128-bit and the counter cache is non-sectored.
PSSM_SC_32_sMdc	Encrypted GPU memory in split counters. The counters are organized with local addresses. The major counter is 32-bit and the counter cache is sectored.

To better understand the performance impacts, we also present the MDC miss rates of our PSSM_nL2_sMdc design in Fig. 16. As we can see from the figure, the MAC cache has high miss rate for most workloads and potentially contribute to more memory bandwidth consumption, especially for the memory intensive benchmarks.

Remaining Performance Bottleneck As we can see from Fig. 15, even with our best design, there is still some performance overhead , 16.84% on average for PSSM_nL2_SC_32_sMdc. To better understand where the remaining overhead comes from, we model ideal MDCs under different scenarios. These ideal designs will limit one specific MDC resource and make the other MDC resource unlimited (meaning that there are only cold misses for these types of

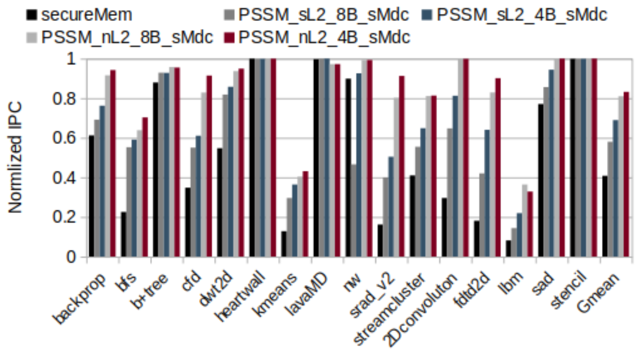

Figure 15: Normalized IPC of different secure GPU memory designs.

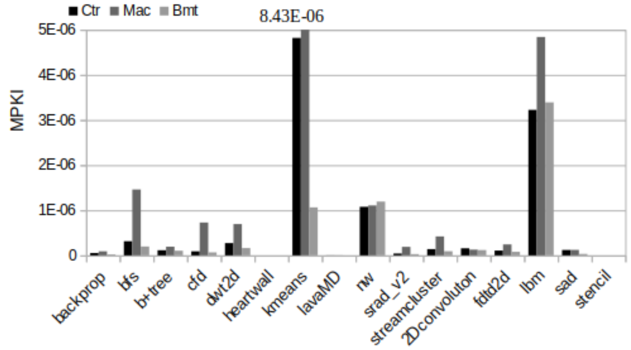

Figure 16: MDC miss rates (in MPKI) of the PSSM_nL2_4B_sMdc design.

metadata). For example, the label **smallCtr** models a 2KB counter cache per partition while the MAC cache and the BMT cache have unlimited capacity. Similarly, the label **smallMac** and **smallBmt** means the MAC cache or BMT cache is 2KB while the others are unlimited. We present the results in Fig. 17. From Fig. 17, we can see that the MAC accesses remain to be the main bottleneck. The reason is that the MACs incur the most storage overhead among all three types of metadata. At every memory read or write, the corresponding MACs must be accessed to verify the data or updated and any MAC cache miss would lead to additional bandwidth pressure.

Memory Encryption For the systems where data confidentiality is the main concern, we may forego data integrity protection to reduce the performance and the hardware overhead. With this application scenarios in mind, we evaluate different GPU memory encryption designs as listed in Table 5 and the results are shown in Fig. 18.

From Fig. 18, we can make the following observations. First, compare with the direct adoption of the split-counter mode encryption from the CPU (labeled 'SC_128_nMdc'), our PSSM designs can improve the performance significantly. The main reason is that the counter redundancy is totally eliminated and the memory bandwidth consumption for the counters is reduced significantly.

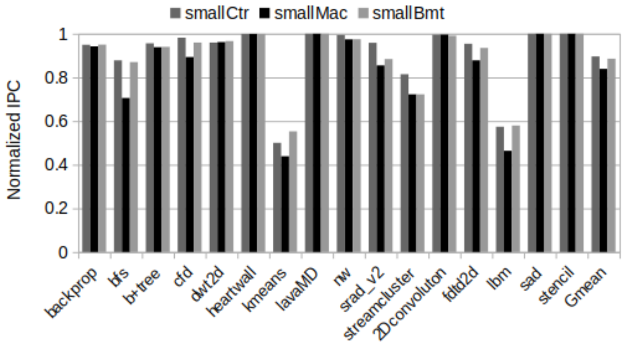

Figure 17: Normalized IPC of the PSSM_nL2_4B_sMdc design with different ideal MDCs.

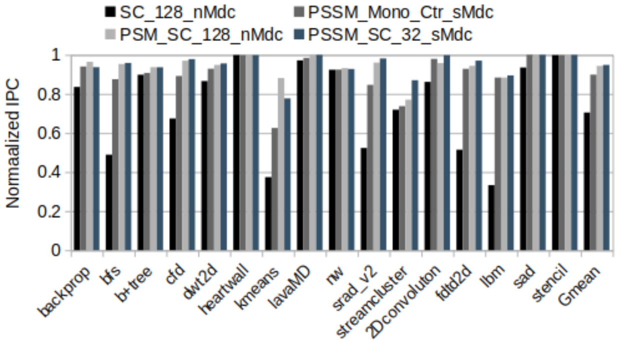

Figure 18: Normalized IPC of different GPU memory encryption schemes.

Second, with our PSSM design, split- counter schemes perform better than the one using monolithic counters. The reason is that with monolithic counters, each 128B block needs a 32-bit counter. In comparison, with split counters, one 128B counter block is shared by 128 128B data blocks. Therefore, one 128B data block requires 8 bits as its counter storage overhead. The 4X higher counter storage in the monolithic counter scheme leads to higher pressure on the counter cache and subsequently higher bandwidth consumption. Third, with split counters in our PSSM design, our sectored counter organization, i.e., SC_32, performs better than the SC_128 organization on average. Hence, we conclude that if the focus is on data confidentiality, the PSSM_SC_32 design would be the choice.

6 RELATED WORK

Secure Memory: An early prototype of secure memory design was proposed in Execution only machine (XOM) [19]. XOM defines the basic threat models (bus snooping, memory tampering), key exchanges and hardware based memory encryption/integrity protection. The TCB is limited to only the processor chip in an XOM machine. Later on, AEGIS [29] proposed by Suh et al. uses integrity trees to prevent replay attacks. Yang et al. proposed one-time pad encryption that offloads the encryption/decryption latency from the critical path [35]. Suh et al. proposed efficient memory integrity

protection with the schemes of speculative verification and lazy update of integrity tree [30]. Rogers et al. proposed Bonsai Merkle Tree (BMT) [26] based on the observation of stateful MACs and reduced the replay attack surface to only encryption counters.
Compact Integrity Tree: Previous works also recognize that the encryption counters and integrity tree of secure memory can incur high performance overhead. Yitbarek et al. [37] proposed to improve the counter storage overhead by compressing the encryption counters with delta encoding. Their work assumes monolithic counters. With delta encoding, the reference counter for the memory blocks within one memory page is extracted and the remaining delta value is maintained for each memory block. Taassori et al. [31] identified that paging overhead of sensitive pages can be very high due the limited size of Enclave Page Cache (EPC), and proposed a variable arity unified tree (VAULT) organization to reduce the height of the integrity tree, which leads to a more compact and smaller integrity tree. Saileshwar et. al [27] proposed the morphable counter design and their observation was that the memory blocks within the same page is not uniformly updated. Hence, they proposed to dynamically change the number of bits used for each minor counter and thus each counter block can hold more minor counters. In other words, each counter block can protect more data blocks. Their design also leads to a more compact integrity tree without hurting the system security. These approach can be compatible with our PSSM design and the only change is that the encryption counters and integrity tree should be organized with local addresses rather than physical addresses.
GPU TEEs: Recognizing the needs for GPUs TEE, recent works including Graviton [33], HIX [14], Telekine [13] and Common Counters [16] try to integrated TEE for GPUs from both software and hardware perspectives. Graviton and HIX have been discussed earlier in the paper. Telekine partitions a GPU application into a sensitive part and a non-sensitive part and guarantees that secret-dependent behaviors can only run on GPU TEE and trusted clients. Common counters integrate the hardware-based memory protection upon Graviton. Their key observation was that the streaming access pattern of GPU applications usually updates a consecutive memory chunk in an uniform manner, which results in common counter values for these memory chunks. By compressing and storing only one copy of these common counter values, the memory bandwidth for fetching/storing encryption counters can be saved. Common counters are complementary to our work and can be combined to further reduce the performance overhead of GPU secure memory. Our earlier work [38] presents a detailed performance study on GPU secure memory architecture. Besides pointing out the performance bottlenecks due to meatadata accesses, it explores MDC organizations, the MSHRs for MDCs, the MEE designs, and different encryption designs (counter-mode and direct encryption) to support GPU secure memory. Our paper uses the recommended designs from this work as the baseline in our experiments.

7 CONCLUSIONS

In this paper, we propose architectural designs for secure memory support on GPUs. Our performance analysis identifies that the partitioned memory architecture of GPUs is not compatible with conventional secure metadata and the existing counter block organization is not friendly to sectored cache structures. We propose Partitioned and Sectored Security Metadata, which is a simple-yet-effective approach to (a) use partition-local addresses for metadata and (b) reorganize the split-counter block to make it fit with sectored caches.Our results show that our proposed scheme effectively reduces the performance overheads of secure memory support on GPUs. Our study also reveals that even with our proposed scheme, memory bandwidth contention due to metadata accesses remains a performance bottleneck for memory intensive workloads. For systems only requiring data confidentiality, the metadata is reduced to only counters. In such a case, our proposed scheme incurs only 5.18% performance overhead on average.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their valuable comments. For this work, the NSCU team is funded in part by NSF grants 1717550 and 1908406, and the UCF team is supported in part by NSF grant 1908079 and by UCF.

REFERENCES

[1] Ashwin M. Aji, Mayank Daga, and Wu-chun Feng. 2011. Bounding the effect of partition camping in GPU kernels. In *Proceedings of the 8th Conference on Computing Frontiers, 2011, Ischia, Italy, May 3-5, 2011*, Calin Cascaval, Pedro Trancoso, and Viktor K. Prasanna (Eds.). ACM, Italy, 27. https://doi.org/10.1145/2016604.2016637

[2] Hartwig Anzt, Yuhsiang M. Tsai, Ahmad Abdelfattah, Terry Cojean, and Jack J. Dongarra. 2020. Evaluating the Performance of NVIDIA's A100 Ampere GPU for Sparse and Batched Computations. In *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, PMBS@SC 2020, Atlanta, GA, USA, November 12, 2020.* IEEE, USA, 26–38. https://doi.org/10.1109/PMBS51919.2020.00009

[3] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization, IISWC 2009, October 4-6, 2009, Austin, TX, USA.* IEEE Computer Society, USA, 44–54. https://doi.org/10.1109/IISWC.2009.5306797

[4] Siddhartha Chhabra, Brian Rogers, Yan Solihin, and Milos Prvulovic. 2009. Making secure processors OS- and performance-friendly. *ACM Trans. Archit. Code Optim.* 5, 4 (2009), 16:1–16:35. https://doi.org/10.1145/1498690.1498691

[5] Intel Corporation. 2019. *Intel® 64 and IA-32 Architectures Software Developer's Manual (325462-071US).* Technical Report. Intel Corporation, USA.

[6] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptol. ePrint Arch.* 2016 (2016), 86. http://eprint.iacr.org/2016/086

[7] Yiwen Gao, Hailong Zhang, Wei Cheng, Yongbin Zhou, and Yuchen Cao. 2018. Electro-magnetic analysis of GPU-based AES implementation. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018.* ACM, USA, 121:1–121:6. https://doi.org/10.1145/3195970.3196042

[8] DAN GOODIN. 2019. *RAMBleed side-channel attack works even when DRAM is protected by error-correcting code.* Retrieved June 11, 2019 from https://arstechnica.com/information-technology/2019/06/researchers-use-rowhammer-bitflips-to-steal-2048-bit-crypto-key/

[9] Scott Grauer-Gray and John Cavazos. 2010. Optimizing and Auto-tuning Belief Propagation on the GPU. In *Languages and Compilers for Parallel Computing - 23rd International Workshop, LCPC 2010, Houston, TX, USA, October 7-9, 2010. Revised Selected Papers (Lecture Notes in Computer Science, Vol. 6548)*, Keith D. Cooper, John M. Mellor-Crummey, and Vivek Sarkar (Eds.). Springer, USA, 121–135. https://doi.org/10.1007/978-3-642-19595-2_9

[10] Shay Gueron. 2016. Memory Encryption for General-Purpose Processors. *IEEE Secur. Priv.* 14, 6 (2016), 54–62. https://doi.org/10.1109/MSP.2016.124

[11] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. 2008. Lest We Remember: Cold Boot Attacks on Encryption Keys. In *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA.* USENIX Association, USA, 45–60. http://www.usenix.org/events/sec08/tech/full_papers/halderman/halderman.pdf

[12] DAVID T. HARPER and J. ROBERT JUMP. 1987. Vector Access Performance in Parallel Memories Using a Skewed Storage Scheme. *IEEE Trans. Computers* C-36, 5 (1987), 1440 – 1449. https://doi.org/10.1109/TC.1987.5009496

[13] Tyler Hunt, Zhipeng Jia, Vance Miller, Ariel Szekely, Yige Hu, Christopher J. Rossbach, and Emmett Witchel. 2020. Telekine: Secure Computing with Cloud GPUs. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, Ranjita Bhagwan and George Porter (Eds.). USENIX Association, USA, 817–833. https://www.usenix.org/conference/nsdi20/presentation/hunt

[14] Insu Jang, Adrian Tang, Taehoon Kim, Simha Sethumadhavan, and Jaehyuk Huh. 2019. Heterogeneous Isolated Execution for Commodity GPUs. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck (Eds.). ACM, USA, 455–468. https://doi.org/10.1145/3297858.3304021

[15] Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele Paolo Scarpazza. 2018. Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking. *CoRR* abs/1804.06826 (2018). arXiv:1804.06826 http://arxiv.org/abs/1804.06826

[16] Zhen Hang Jiang, Yunsi Fei, and David R. Kaeli. 2016. A complete key recovery timing attack on a GPU. In *2016 IEEE International Symposium on High Performance Computer Architecture, HPCA 2016, Barcelona, Spain, March 12-16, 2016.* IEEE Computer Society, USA, 394–405. https://doi.org/10.1109/HPCA.2016.7446081

[17] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. 2020. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In *47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020, Valencia, Spain, May 30 - June 3, 2020.* IEEE, Spain, 473–486. https://doi.org/10.1109/ISCA45697.2020.00047

[18] Duncan H. Lawrie and Chandra R. Vora. 1982. The Prime Memory System for Array Access. *IEEE Trans. Computers* 31, 5 (1982), 435–442. https://doi.org/10.1109/TC.1982.1676020

[19] David Lie, Chandramohan A. Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John C. Mitchell, and Mark Horowitz. 2000. Architectural Support for Copy and Tamper Resistant Software. In *ASPLOS-IX Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, USA, November 12-15, 2000.* ACM Press, USA, 168–177. https://doi.org/10.1145/356989.357005

[20] Zhen Lin, Michael Mantor, and Huiyang Zhou. 2018. GPU Performance vs. Thread-Level Parallelism: Scalability Analysis and a Novel Way to Improve TLP. *ACM Trans. Archit. Code Optim.* 15, 1 (2018), 15:1–15:21. https://doi.org/10.1145/3177964

[21] Onur Mutlu. 2017. The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser. *CoRR* abs/1703.00626 (2017). arXiv:1703.00626 http://arxiv.org/abs/1703.00626

[22] Seonjin Na, Sunho Lee, Yeonjae Kim, Jongse Park, and Jaehyuk Huh. 2021. Common Counters: Compressed Encryption Counters for Secure GPU Memory. In *2021 IEEE International Symposium on High Performance Computer Architecture, HPCA 2021, Seoul, South Korea, Feb. 27 — Mar. 3.* IEEE Computer Society, South Korea.

[23] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael B. Abu-Ghazaleh. 2018. Rendered Insecure: GPU Side Channel Attacks are Practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018.* ACM, Canada, 2139–2153. https://doi.org/10.1145/3243734.3243831

[24] Sandro Pinto and Nuno Santos. 2019. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.* 51, 6 (2019), 130:1–130:36. https://doi.org/10.1145/3291047

[25] B. Ramakrishna Rau. 1979. Interleaved Memory Bandwidth in a Model of a Muyltiprocessor Computer System. *IEEE Trans. Computers* 28, 9 (1979), 678–681. https://doi.org/10.1109/TC.1979.1675436

[26] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. 2007. Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-40 2007), 1-5 December 2007, Chicago, Illinois, USA.* IEEE Computer Society, USA, 183–196. https://doi.org/10.1109/MICRO.2007.16

[27] Gururaj Saileshwar, Prashant J. Nair, Prakash Ramrakhyani, Wendy Elsasser, José A. Joao, and Moinuddin K. Qureshi. 2018. Morphable Counters: Enabling Compact Integrity Trees For Low-Overhead Secure Memories. In *51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2018, Fukuoka, Japan, October 20-24, 2018.* IEEE Computer Society, Japan, 416–427. https://doi.org/10.1109/MICRO.2018.00041

[28] John A. Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and Wen mei W. Hwu. 2009. *Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing.* Technical Report. Champaign, IL USA.

[29] G. Edward Suh, Dwaine E. Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. 2003. AEGIS: architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the 17th Annual International Conference on Supercomputing, ICS 2003, San Francisco, CA, USA, June 23-26, 2003*, Utpal Banerjee, Kyle A. Gallivan, and Antonio González (Eds.). ACM, USA, 160–171. https://doi.org/10.1145/782814.782838

[30] G. Edward Suh, Dwaine E. Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. 2003. Efficient Memory Integrity Verification and Encryption for Secure Processors. In *Proceedings of the 36th Annual International Symposium on Microarchitecture, San Diego, CA, USA, December 3-5, 2003.* IEEE Computer Society, USA, 339–350. https://doi.org/10.1109/MICRO.2003.1253207

[31] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. 2018. VAULT: Reducing Paging Overheads in SGX with Efficient Integrity Verification Structures. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018.* ACM, USA, 665–678. https://doi.org/10.1145/3173162.3177155

[32] Jitendra K. Tugnait. 2016. Detection of Active Eavesdropping Attack by Spoofing Relay in Multiple Antenna Systems. *IEEE Wirel. Commun. Lett.* 5, 5 (2016), 460–463. https://doi.org/10.1109/LWC.2016.2585549

[33] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: Trusted Execution Environments on GPUs. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, Andrea C. Arpaci-Dusseau and Geoff Voelker (Eds.). USENIX Association, USA, 681–696. https://www.usenix.org/conference/osdi18/presentation/volos

[34] Chenyu Yan, Daniel Englender, Milos Prvulovic, Brian Rogers, and Yan Solihin. 2006. Improving Cost, Performance, and Security of Memory Encryption and Authentication. In *33rd International Symposium on Computer Architecture (ISCA 2006), June 17-21, 2006, Boston, MA, USA.* IEEE Computer Society, USA, 179–190. https://doi.org/10.1109/ISCA.2006.22

[35] Jun Yang, Youtao Zhang, and Lan Gao. 2003. Fast Secure Processor for Inhibiting Software Piracy and Tampering. In *Proceedings of the 36th Annual International Symposium on Microarchitecture, San Diego, CA, USA, December 3-5, 2003.* IEEE Computer Society, USA, 351–360. https://doi.org/10.1109/MICRO.2003.1253209

[36] Yi Yang, Ping Xiang, Jingfei Kong, and Huiyang Zhou. 2010. A GPGPU compiler for memory optimization and parallelism management. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2010, Toronto, Ontario, Canada, June 5-10, 2010*, Benjamin G. Zorn and Alexander Aiken (Eds.). ACM, Canada, 86–97. https://doi.org/10.1145/1806596.1806606

[37] Salessawi Ferede Yitbarek and Todd M. Austin. 2018. Reducing the overhead of authenticated memory encryption using delta encoding and ECC memory. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018.* ACM, USA, 35:1–35:6. https://doi.org/10.1145/3195970.3196102

[38] Shougang Yuan, Ardhi Wiratama Baskara Yudha, Yan Solihin, and Huiyang Zhou. 2021. Analyzing Secure Memory Architecture for GPUs. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS.* IEEE Computer Society.

[39] Zhao Zhang, Zhichun Zhu, and Xiaodong Zhang. 2000. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 33, Monterey, California, USA, December 10-13, 2000.* ACM/IEEE Computer Society, USA, 32–41. https://doi.org/10.1109/MICRO.2000.898056