Analyzing Secure Memory Architecture for GPUs

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

Ardhi Wiratama Baskara Yudha, Yan Solihin
University of Central Florida
Orlando, Florida
yudha@knights.ucf.edu, Yan.Solihin@.ucf.edu

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

***Abstract*—Wide adoption of cloud computing makes privacy and security a primary concern. Although recent CPUs have integrated secure memory architecture, such support is still missing for GPUs, a key accelerator in data centers.**

In this paper, we explore two secure memory architectures, counter-mode encryption and direct encryption, for GPUs and show that we need to architect secure memory differently from it for CPUs. Our in-depth study reveals the following insights. First, as GPUs are designed for high-throughput computation, its secure memory needs to deliver high bandwidth. Second, with counter-mode encryption, the memory traffic resulting from the metadata, i.e., the counters, MACs (message-authentication codes), and integrity tree, may cause significant performance degradation, even in the presence of metadata caches. Third, the sectored cache structure adopted by GPUs leads to multiple sequential accesses to the same metadata cache line, which necessitates the use of MSHRs (miss-status handling registers) for meta-data caches. Fourth, unlike CPUs, separate/partitioned metadata caches perform better than unified metadata caches on GPUs. The reason is that GPU workloads feature streaming accesses, which cause severe contention in the unified metadata cache and the cached counters and integrity tree nodes may be evicted before being reused. Fifth, the massive-threaded nature of GPUs make them latency-tolerant and the performance impact due to the extra encryption/decryption latency is limited. As a result, direct encryption can be a promising alternative for GPU secure memory. The challenge, however, lies in memory integrity verification as the integrity tree may incur high storage overhead and metadata traffic.

***Index Terms*—GPUs, security, secure memory, memory encryption, memory integrity, metadata cache**

I. Introduction

Cloud computing has become the predominant computing paradigm. With the cloud being a shared resource, it is critical to provide users with sufficient privacy and security guarantees. Toward this end, recent CPUs have integrated hardware-based trusted execution environment (aka *TEE*), such as Intel SGX [2], AMD SEV [11] and ARM TrustZone [18]. A critical building block for TEE is *secure memory engine*, which keeps data in memory encrypted [5], [11] and protects its integrity [5]. However, such secure memory architecture support is missing on GPUs, a key accelerator in clouds for a wide range of workloads including machine learning, scientific computing, 3D rendering, etc. As the system security is determined by its weakest link, we argue that accelerators such as GPUs also need to provide TEE.

Recognizing the needs, recent work, including HIX [8] and Graviton [29] provide isolated execution environment for the programs offloaded to the GPUs. However, their threat model is weaker than CPU TEE, as they do not provide secure memory support and require system software (e.g., GPU driver) to be trusted. In this paper, we explore the design space and implication of supporting secure memory for GPUs. We evaluate both counter-mode and direct encryption, and show that we need to architect secure memory differently from the CPU counterpart. Similar to the CPU TEE (e.g., Intel SGX), the secure memory hardware is placed in the memory controller in our design, as shown in Fig. 1. As GPUs incorporate multiple memory controllers to provide high memory bandwidth, the same secure memory hardware is replicated in each memory controller.

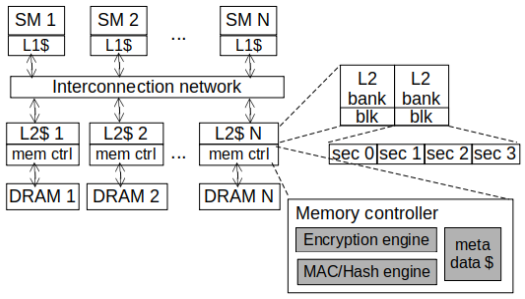

Fig. 1: GPU architecture with secure memory support

We start our investigation with counter-mode encryption and Bonsai Merkle Tree (BMT) [20], which are the state-of-art secure memory architecture on CPUs. Counter-mode encryption is introduced to hide the decryption latency and BMT is used to verify the integrity of data through its counters, which are used to encrypt/decrypt the data along with the secret key. In counter-mode encryption, the security metadata include counters, BMT, and MACs of the ciphertext. To reduce the extra memory accesses for fetching the metadata, on-chip metadata caches are commonly employed. We model the counter-mode encryption and BMT support upon a GPU model based on Nvidia Volta [9] architecture, and identify that hardware-based secure memory architecture can incur significant performance overhead for GPUs. The main reason is that the metadata data accesses generate a lot of memory traffic even in the presence of metadata caches. Given the high bandwidth requirements of GPUs, the additional memory traffic contends for the memory bandwidth and significantly slows down the GPUs performance. Also, we observe that due to the sectored L2 cache structure in GPUs, the streaming

data accesses leads to a high ratio of secondary misses (i.e., misses to the same cache block after it being requested) in the metadata caches, which highlight the importance of the MSHRs to filter out redundant memory requests. Furthermore, due to the nature of high-throughput computation of GPUs, high-throughput cryptographic engines are also needed to produce a balanced system.

Besides counter-mode encryption, we also analyze the alternative design of direct encryption. Without counters, BMT is infeasible. Therefore, we resort to a Merkle Tree (MT) to perform integrity checks. In other words, with direct encryption, the security metadata include MACs and the MT, both of them are used for memory integrity verification. Our analysis reveals the following interesting observations. First, direct encryption itself does not lead to high performance overheads because GPUs are designed to be latency tolerant. Actually, our results show that direct encryption can perform better than counter-mode encryption even with a high encryption latency of 160 cycles. Second, memory integrity protection may become the performance bottleneck due to the memory traffic generated by MAC and MT accesses.

The remainder of the paper is organized as follows. Section II presents the background and specifies the threat model. Section III discusses the related work. Section IV introduces our experimental methodology. Section V analyzes counter-mode encryption and explores different design choices for the metadata caches and AES engines. Section VI focuses on direct encryption and evaluates the performance impact of encryption with and without memory integrity protection. Section VII concludes our paper and summarizes our findings on designing secure memory architecture for GPUs.

II. Background

A. GPU Architecture

Contemporary GPU architecture, as shown in Fig. 1, consists of multiple Streaming Multiprocessors (SMs) to support the SIMT programming model [24]. Each SM has a private L1 cache and all SMs share a multi-banked L2 cache. The L1 caches and L2 cache banks are connected through an interconnection network.

Given the large number of SMs and the high degrees of thread-level parallelism (TLP), GPUs require high memory bandwidth. There are multiple memory controllers on chip and each connects to a few L2 cache banks. To reduce the memory bandwidth consumption, sectored caches are commonly employed in GPUs. Typically, one 128B cache line contains four 32B sectors and each memory access fetches a sector rather than an entire cache line.

B. Threat Model and Scope of Work

The threat model for CPU TEE, such as Intel SGX, assumes two types of threats: compromised system/privileged software (such as the OS and hypervisor) and attackers that have physical access to the remote server and the abilities to snoop/scan and modify data stored in off-chip memory. CPU TEE assumes that the processor chip provides a security boundary, where all on-chip components are assumed to be out of the reach of attackers. In general, TEE requires three major architecture supports: *hardware key management*, *attestation*, and *secure memory engine* (which encrypts and protects the integrity of data stored off the processor chip) [27], [28], [30], [31]. Of these, secure memory incurs the largest performance overheads as it must be active at all time and affects the critical path of load instructions. Due to the enormity of the overheads, recent attempts to extend CPU TEE to GPUs [8], [29] ignore the physical attack threats and enlarge the trust base (e.g., adding GPU memory module to the trust base). In contrast, this research assumes the same threat model in CPU TEE also affects GPUs, and assumes that GPU chip provides the security boundary, where all the data stored in the on-chip resources such as registers and caches are safe. The attackers may have physical access to the GPUs hardware, and have the capability to snoop the GPU memory buses or to scan/tamper the GPU memory content.

The scope of this paper covers the design space of secure memory for GPUs. Hardware key management and attestation have been well studied in previous work (i.e., Graviton [29] and HIX [8]) and are beyond the scope of this paper. Securing the communication channel between CPU and GPU is also outside the scope of this work. Existing solutions for that may include PCI modification [8], or for tighter integration, secure cache coherence protocol [22]. Furthermore, side channel attacks such as timing-based side channel attack [10] are also out of the scope of our work.

C. Security Mechanisms

Memory encryption. The goal of memory encryption is to protect data confidentiality [14], [19], [21], [32]. The cryptographic hardware engines residing in the memory controller are responsible for performing encryption/decryption before the data is sent to/returned from the off-chip memory. Generally, there are two approaches for memory encryption as shown in Fig. 2: counter-mode encryption and direct encryption.

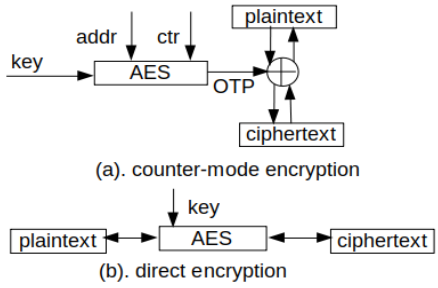

Fig. 2: Counter-mode encryption and direct encryption

Counter-mode encryption can hide the decryption latency by overlapping it with memory reads, and thus offload the decryption latency from the critical path of load instructions. In counter-mode encryption, there is a counter associated with each cache line. When the memory controller is fetching the data from off-chip memory, the corresponding counter and the

block address are used to generate a one-time pad (OTP) using the encryption engine such as AES. The memory controller can recover the plaintext by XORing the data and OTP in one cycle and then supply the data to processors. To guarantee the security, the counters cannot be reused. Hence, at each dirty eviction from the last level cache (LLC), the counter corresponding to the cache line will be incremented. Since the encryption strength of counter-mode memory encryption is conditional upon not reusing the counters, if the attacker can trigger reuse of counters, the encryption will be broken [20], [30]. Therefore, counter-mode encryption *fundamentally relies* on counter integrity protection to provide data confidentiality.

Direct encryption can also be used for the security purpose. In contrast to counter-mode encryption, with direct encryption, confidentiality does not necessarily require integrity protection. The main disadvantage of direct encryption is that it exposes the latency of decryption to the critical path of memory reads, because decryption can only start after the data is fetched from memory [26].

Memory Integrity Verification. As discussed earlier, counter-mode encryption requires integrity protection of counters [20]. Furthermore, MACs are needed to protect against memory tampering attack [30], [33]. As pointed out by Rogers et. al [20], stateful MACs calculated from the ciphertext, block address and the corresponding counter can provide data integrity protection, and the Merkle tree (MT) only needs to cover the counters to protect against counter-replay attacks. This scheme is named BMT and can help to reduce the storage overheads of the MT. Moreover, the BMT is much shallower than the original MT. Any modification to the data or MACs stored in the 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. In contrast, with direct encryption, encryption protection is not dependent on integrity verification; and integrity verification provides additional protection against memory tampering and replay attacks. Integrity protection relies on (1) MAC of the ciphertext and (2) a Merkle/Hash Tree based on the entire memory with a special on-chip register storing the root of the tree [4].

III. Related Work

Recent studies highlight the needs for security support within accelerators. Deep learning (DL) is such an application that attracts special attention [16]. Several recent works [6], [7], [34] aim to extend the existing memory encryption and integrity verification schemes to DL accelerators.

The work [34] by Zuo et al. pointed out that the gap between the throughput of AES engine and the GPU memory bandwidth is the key bottleneck for secure GPU memory. To reduce the overhead of encryption, they proposed: (1) selective memory encryption, (2) co-location of data and the corresponding counter. In comparison, our work differs from this prior work in the following ways. First, our work consider data integrity protection, which is missing in this prior work. Second, we perform a detailed study on metadata cache designs, including the impact from the sectored L2 caches as well as unified metadata caches and separate metadata caches. Third, we exploit pipelined AES engines to overcome the AES throughput limitation and explore the design trade-offs for different AES throughput. Fourth, the proposed solution by Zuo et al. [34] is specific to DL while our work covers a wide range of workloads.

Similar to Zuo's work [34], Hua et al. [6], [7] aimed to provide memory protection for the deep learning workloads running on the accelerators. As mentioned in [7], the overheads of memory encryption and integrity verification come from the extra memory accesses generated by the security metadata. In their system design, they proposed to provide memory encryption and integrity verification based on a coarse granularity of memory objects. Their key observation is that accelerators often explicitly move data between on-chip memory and off-chip DRAM at the object granularity. In comparison, our work provides detailed performance analysis of both counter-mode encryption and direct encryption and explore different metadata cache design choices.

Other works, including HIX [8] and Graviton [29] that aimed to provide the TEE for GPUs, mainly focused on protecting the user program and data from untrusted OS or privileged malicious software (hypervisor), and did not consider the physical attacks, i.e., memory scanning and tampering attacks. In other words, these two works did not provide hardware-based memory encryption and integrity verification protections. The data stored on GPU off-chip memory is plaintext and there is no support for integrity checks. In comparison, our work defines a stronger threat model, and provides detailed performance study on memory encryption and integrity verification for GPUs.

IV. Methodology

We model different secure memory architecture supports using GPGPU-Sim v4.0 [12]. The configuration of our baseline GPU model is shown in Table I, which is based on the Nvidia Volta architecture [9]. In our experiments, we assume that a range of 4GB device memory is protected, and the metadata cache line size is aligned to the data cache line size, which is 128B. The metadata organization and storage overhead are listed in Table II.

For counter-mode encryption, each counter cache line maintains one 128-bit major counter (shared by data blocks within a 16KB memory chunk) and 128 7-bit per block minor counters, thereby covering 128 lines of data. In other words, the ratio between data and counter capacity is 128 and the overall counter blocks take 32MB (=4GB/128) off-chip storage. Using a 64-bit MAC for each 128B data, the overall MAC consumes 256MB storage. As a result of the sectored L2 cache, we use truncated MAC, i.e., 16-bit MAC for each 32B sector. For the BMT, we build a 6-level 16-ary hash tree and its overall capacity is 2.14 MB, excluding the counter blocks that serve as the leaf nodes of the BMT. Therefore, the overall capacity of

TABLE I: 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/partition, 96KB/bank, 6MB in total
DRAM	850MHz, 868GB/s, 32 partitions

TABLE II: Metadata organization and storage

Metadata Type	Counter-mode Encryption	Direct Encryption
Counter	128B/16KB, 7b/blk, 32MB	-
MAC	8B/blk, 2B/sector, 256MB	8B/blk, 2B/sector, 256MB
BMT/MT	16 ary, 6 levels, 2.14MB	16 ary, 7 levels, 17.1MB

metadata is 290.14MB (=32+256+2.14 MB) for counter-mode encryption.

With direct encryption, using the same 64-bit MAC for each 128B data, the overall MAC consumes 256MB memory. As the MT is built upon the 4GB data, we use a 7-level 16-ary tree and the overall capacity is 17.1MB, excluding the MACs that serve as the leaf nodes of the MT. Therefore, the overall memory overhead is 273.1MB (=256+17.1 MB) for direct encryption.

The separate metadata caches, which include the counter cache, MAC cache, and BMT/MT cache, are modeled in each memory partition (i.e., each memory controller). The metadata cache specification is listed in Table III. State-of-the-art secure memory architecture in CPUs uses speculative verification and lazy update for BMT/MT [4]. We also adopt these schemes on GPUs. Speculative verification means that the memory controller can supply the data to the core before the corresponding integrity check is finished. Later on, if there is a failure in integrity verification, an exception would be raised. Lazy update means that only when a counter block or a tree node is evicted from the counter cache or BMT/MT cache, its parent will be updated in the BMT/MT cache.

In our study, we assume pipelined AES engines and pipelined MAC units. This way, the encryption latency or the MAC computation latency would not affect the throughput. With AES-128, 16B of data can come out of a pipelined AES engine each cycle. With the memory clock frequency of 850MHz, the throughput of the pipelined AES engine is 13.6 GB/s (=16B*850MHz). With 2 AES engines in each memory partition, the throughput would match the memory bandwidth, resulting in a balanced design. Therefore, we use 2 pipelined AES engines in each partition by default. We model different AES latencies and a 40-cycle latency for the MAC unit. With counter-mode encryption, the AES latency does not matter as it is hidden by design.

Our benchmarks are selected from the Rodinia-3.1 [1], Parboil [25] and polybench [3] benchmark suites to cover a wide range of workloads. Table IV lists the details of these benchmarks. For each benchmark, we simulate until it has run for 4 million cycles. Table IV also reports the bandwidth utilization and the IPC (instruction-per-cycle) for each benchmark when running on the baseline GPU without secure memory support. In this paper, we categorize the benchmarks which consume more than 50% peak DRAM bandwidth as memory intensive, the benchmarks which consume less than 20% peak DRAM bandwidth as non memory intensive, and the remaining as medium memory intensive.

TABLE III: Metadata cache organization

Counter cache	{2,4,8,16,32,64}KB/Memory Partition, 2KB default, 128B blk, 64 MSHRs, allocate-on-fill policy
Mac cache	{2,4,8,16,32,64}KB/Memory Partition, 2KB default, 128B blk, 64 MSHRs, allocate-on-fill policy
(Bonsai)Merkle Tree cache	{2,4,8,16,32,64}KB/Memory Partition, 2KB default, 128B blk, 64 MSHRs, allocate-on-fill policy
Unified metadata cache	6KB/Memory Partition, 128B blk, 192 MSHRs, allocate-on-fill policy
Hash/Mac latency	40 cycles default
AES engines	{1,2}/Memory Partition, 2 default.

TABLE IV: Benchmarks

Categorization	Benchmark name	Bandwidth utilization	IPC
non memory intensive	heartwall	<1%	1,195.37
	lavaMD	<1%	4,615.23
	nw	<2%	23.90
	b+tree	12%-14%	2,768.61
medium memory intensive	backprop	25%	3,067.61
	cfd	15%-50%	1,076.98
	dwt2d	20%-50%	784.70
	kmeans	40%-45%	97.04
memory intensive	bfs	5%-60%	699.51
	srad_v2	79%- 80%	3,306.82
	streamcluster	78%-80%	1,178.18
	2Dconvolution	53%	2,487.22
	fdtd2d	82%-83%	1,773.95
	lbm	58%	552.12

V. Counter-Mode Encryption

In this section, we perform an in-depth study on extending counter-mode encryption and BMT to GPUs. Different design options, as listed in Table V, are explored to reveal the performance bottlenecks.

A. Performance Overhead

Fig. 3 shows the performance of various GPU secure memory models normalized to the baseline GPU without security support. From the figure, we can see that adding secure memory support may incur significant performance overhead. The normalized IPC is reduced by 65.9% on average using the geometric mean (Gmean), and can be up to 91.06% for memory-intensive workloads such as lbm.

TABLE V: Evaluated designs for counter-mode encryption

Scheme	What It Represents
baseline	Baseline GPU without secure memory support.
secureMem	Baseline GPU with secure memory support using counter-mode encryption and BMT. In Section V-A, no MSHR is modeled, in Section V-B, V-C, V-D, V-E, there are 64 MSHRs for each metadata cache.
secureMem_xMB	secure GPU memory with different L2 capacities upon our baseline GPU with counter-mode encryption and BMT.
0_crypto	secureMem with 0 MAC latency and encryption latency.
perf_mdc	secureMem with perfect metadata caches, i.e., there are no cache misses and write backs.
large_mdc	secureMem with unlimited capacity for metadata caches, meaning that there are only cold misses.
mshr_x	secureMem with different mshr capacity for metadata caches.
separate	secureMem with separate metadata caches in each memory partition/unit.
unified	secureMem with a unified metadata cache, which caches all types of metadata, in each memory partition/unit.

To identify the performance bottleneck, we model different idealistic designs. From Fig. 3, we can see that zero cycle cryptographic operation latency (zero-cycle MAC and AES) does not alleviate the performance overhead. This is expected as GPUs are massively parallel machines and leverage high-degrees of TLP to hide the latency. However, when we model ideal metadata caches or unlimited capacity metadata caches, the GPU performance with secure memory support becomes very close to the baseline. This indicates that the accesses to the security metadata is the performance bottleneck.

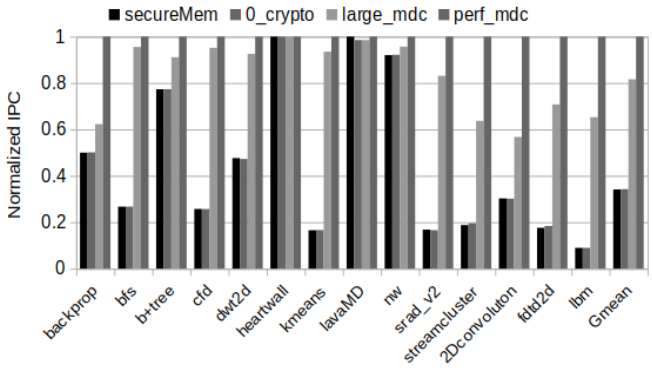

Fig. 3: Normalized IPC of counter mode encryption with Bonsai Merkle tree.

To further analyze the impact of the metadata accesses, in Fig. 4 we compare the amounts of different types of memory requests when secure memory is integrated, i.e., the secureMem model in Table V. The memory-request types include regular data read and write requests (labeled 'data'), counter reads (labeled 'ctr'), MAC reads (labeled 'mac'), BMT reads (labeled 'bmt'), and all the writebacks from the metadata caches (labeled 'wb').

From Fig. 4, we can make the following observations. First, for all benchmarks, the memory requests to fetch MACs from off-chip memory account for a major portion of the memory traffic (25.58% on average). Second, the counter requests also account for a large portion of memory traffic (21.77% on average). Third, some benchmarks, including bfs, b+tree, kmeans, nw and lbm, have a relatively high ratio of memory requests for fetching the BMT blocks from memory. The reason is that these benchmarks have relatively high counter-cache miss rates and the fetched counters need to be validated. Fourth, extra memory traffic due to metadata accesses may not necessarily lead to performance degradation. For example, for the non-memory intensive benchmarks (heartwall, lavaMD and nw), the metadata accesses account for a large portion of the memory traffic (66.07%, 62.71%, 75.41%), but the performance impact is near zero. The reason is that for these benchmarks, the memory bandwidth is under-utilized as shown in Table IV. Therefore, additional traffic does not result in contention for the memory bandwidth. Fifth, some benchmarks, including bfs, dwt2d and lbm, have relatively high amounts of metadata-cache writebacks, which also impact the performance.

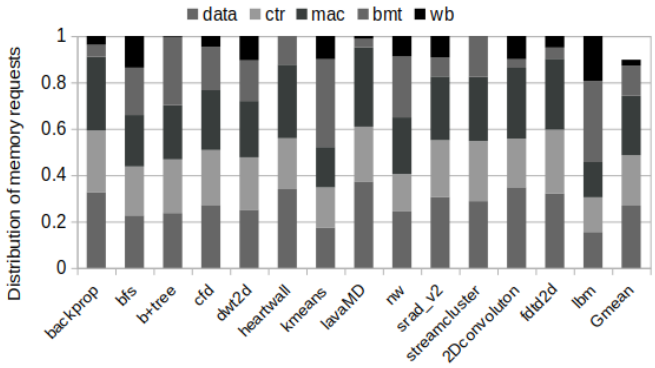

Fig. 4: Distribution of different types of memory requests.

B. MSHRs for Metadata Caches

As shown in Fig. 4, metadata accesses are the main contributor for additional memory traffic. A deeper analysis reveals that many metadata cache misses are secondary misses, meaning that the missed metadata blocks have already been requested but have not returned from the memory. We report the ratio of secondary misses in different benchmarks in Fig. 5. We can see that the secondary misses account for 64.96%, 59.67% and 85.63% for counter/MAC/BMT cache misses on average. And it can be even more than 90% for some memory-intensive benchmarks like streamcluster. The reason is that contemporary GPUs usually employ the sectored cache structures to reduce memory bandwidth requirement. However, the sectored L2 cache combined with streaming data access

pattern will lead to multiple sequential access to the same cache line in metadata cache. Suppose we have a streaming memory access pattern with 4 memory requests {0x0, 0x20, 0x40, x60}. In a non-sectored L2 cache with 128B cache line, one cache miss will be generated, which in turn leads to one counter/MAC cache miss. With a sectored L2 cache (4 sectors and each sector size of 32B), in contrast, four L2 misses are generated, which leads to four counter/MAC cache misses (1 primary and 3 secondary) to the same counter/MAC cache line.

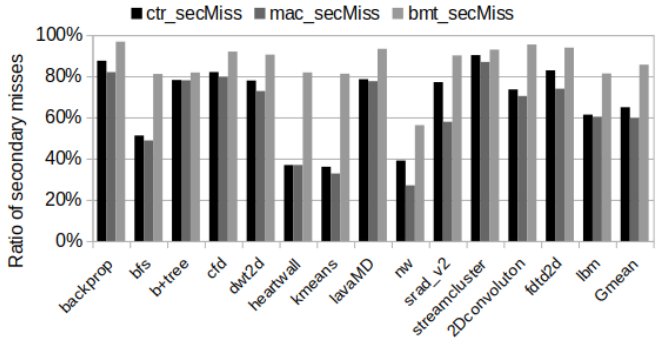

Fig. 5: Amount of secondary misses in metadata caches.

The solution to avoiding the memory traffic generated by the secondary metadata cache misses is to add MSHRs to metadata caches. Here, we show the performance impact with different MSHR sizes for metadata caches and the results are shown in Fig. 6. Due to the metadata cache organization (i.e., one metadata cache line may cover multiple data cache lines/sectors), we assume each MSHR entry can merge at most 512/64/64 requests in the counter/MAC/BMT cache. As we can see from Fig. 6, 64 MSHRs in a metadata cache can be a good choice considering the performance and hardware cost. Hence, we assume 64 MSHRs as the default size in our experiments.

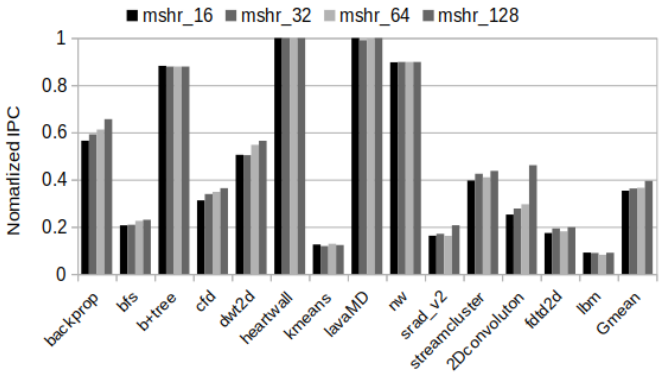

Fig. 6: Normalized IPC of secure memory with different numbers of MSHRs in metadata caches.

C. Metadata Cache Size

In the next experiment, we vary the metadata cache size from 2KB to 64KB. The performance results are shown in Fig. 7. Since our baseline GPU has 32 memory partitions, the overall metadata cache capacity would vary from 192KB (=32x3x2KB) to 6MB (=32x3x64KB).

As expected, enlarging the metadata cache reduces the memory traffic and improves the performance. However, there is still a performance degradation of 46.17% on average even when the overall metadata cache capacity is enlarged to 6MB in total, which is equal to the capacity of our baseline GPU. For memory-intensive benchmarks, the performance overhead can be even more significant. For example, with the 6MB metadata cache, secure memory support still slows down the GPUs by 78.87% for kmeans, 67.82% for srad_v2 and 72.64% for lbm. On one hand, these benchmarks have very high memory bandwidth utilization, therefore any additional memory traffic will incur more contention to the memory system. On the other hand, these benchmarks have many cold misses even in the presence of large metadata caches.

In the baseline GPU, each memory partition has 192KB (=92KB*2) L2 cache capacity, and each counter cache line has 128 minor counters. To maintain the counters for all the L2 cache blocks in this memory partition, the capacity of counter cache should be at least 192KB/128 = 1.5KB. Therefore we use 2KB as the default size for metadata caches in each memory partition.

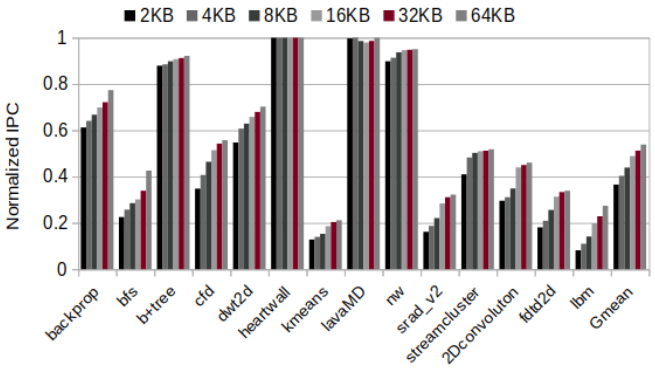

Fig. 7: Normalized IPC for different metadata cache sizes.

D. Unified vs. Separate Metadata Caches

On CPU secure memory, Lehman et al. [15] have studied the metadata cache access patterns and concluded that caching all metadata together in an unified metadata cache is better than caching them in separate metadata caches. In our experiment, we model an unified metadata cache in each memory partition that has the same capacity as the combined capacity of the three separate metadata caches. The performance comparison between the unified and separate metadata caches is shown in Fig. 8.

From Fig. 8 we can see that in contrast to CPUs, separate metadata caches outperform the unified metadata cache on GPUs. To figure out the reason, we report the miss rates of different types of metadata in Fig. 9. From Fig. 9, we can see that with unified metadata cache, the miss rate of different metadata all raises. From 22.77% to 24.03% for encryption

counters, from 31.75% to 31.82% for MACs and from 4.02% to 5.93% for BMT. The reason is, the newly fetched metadata blocks keep on evicting the other metadata blocks from the unified metadata cache due to the streaming access pattern of the GPU workloads, which results in higher metadata misses. Moreover, every evicted counter and BMT node may need to update their parents, which potentially leads to more write updates to the BMT cache and more dirty evictions from BMT nodes. Our evaluation results show that the memory traffic due to metadata writebacks in unified metadata cache can be 1.47X higher than those with separate metadata caches on average.

We also study the reuse distance of metadata in GPU secure memory. We choose the benchmark fdtd2d as our case study and present the reuse distance distribution of its counter and MAC accesses in Fig. 10 and Fig. 11, respectively. With the unified cache, we first collect the metadata access trace from partition 0 and extract sub-traces for each type of metadata. Then we compute the reuse distance based on the sub-traces. The references are grouped into different reuse distance buckets, where [x,y] means the reuse distance between x and y. From the figure, we can see that since GPUs feature streaming data access pattern, the metadata also shows a streaming pattern as most metadata accesses have a reuse distance of zero (i.e., accessing the same metadata cache line). In addition, with unified metadata caches, the counter and MAC accesses show higher numbers of accesses with reuse distances in the range between 65 and 512 while having smaller numbers of accesses with reuse distances between 1 and 8. This indicates that higher capacities are required to capture these reuses using unified caches compared to separate caches. Note that accesses with small reuse distances (e.g., 0) do not always result in cache hits. The reason is that if the first access is a miss and the data has not been fetched, subsequent accesses to the same cache line become secondary misses.

Overall, our conclusion is that for streaming data access patterns, we may either use separate metadata caches or adopt smart replacement policies to avoid the thrashing behavior. Note that the thrashing-avoiding replacement policies proposed for CPU last-level caches may not be readily adopted. The reason is that each newly fetched metadata block may be accessed multiple times as each metadata block protects multiple data blocks. For example, one MAC block contains MACs for 16 data blocks, with perfectly streaming accesses, the MAC block will be reused for 16 times.

E. AES Engine Throughput

To achieve high-throughput computation, GPUs usually have high requirements for the memory bandwidth. For example, our baseline GPU has a peak memory bandwidth of 868 GB/s. So far, we have assumed that each memory partition can be equipped with 2 pipelined AES engines such that the encryption throughput can catch up with the memory bandwidth. In this case, there would be a total of 64 (=2*32) AES engines residing on the GPU chip.

To analyze the performance impact with different numbers of AES engines for each memory partition/unit, we reduce the

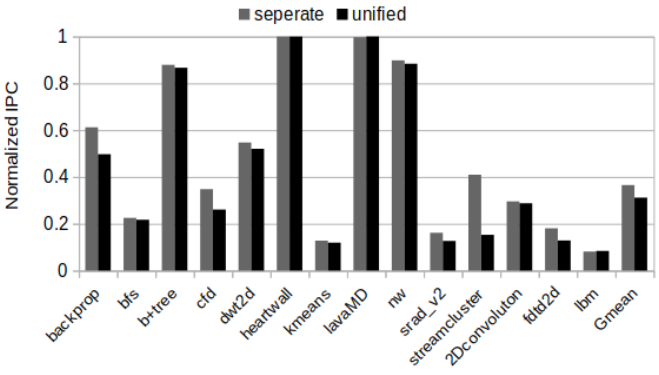

Fig. 8: Normalized IPC of unified metadata caches vs separate metadata caches.

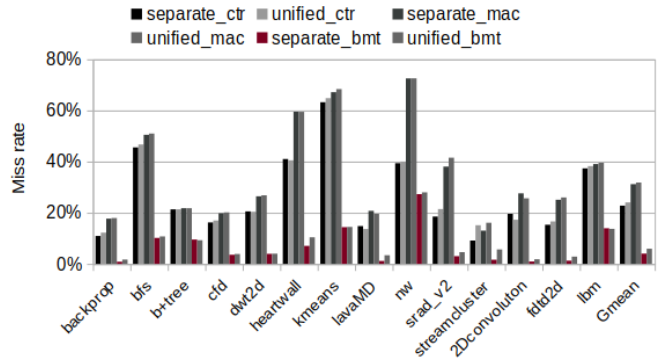

Fig. 9: Miss rates for different types of metadata in unified vs. separate metadata caches.

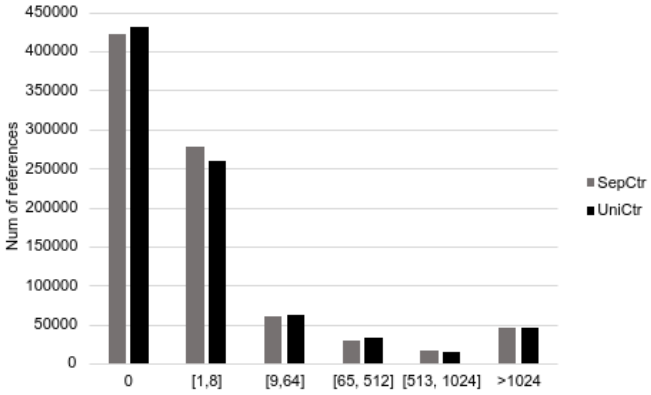

Fig. 10: Reuse distance of counters of the benchmark fdtd2d

number of AES engines in each memory partition/unit from 2 to 1 and the performance results are shown in Fig. 12. From the figure, we can see that although some benchmarks such as b+tree and kmeans observe a little performance degradation, most of the benchmarks are not affected by having just one pipeline AES engine in each memory partition. The reason is that for benchmarks with relatively low memory utilization, neither the memory bandwidth nor the AES throughput is the performance bottleneck. For the workloads with high memory

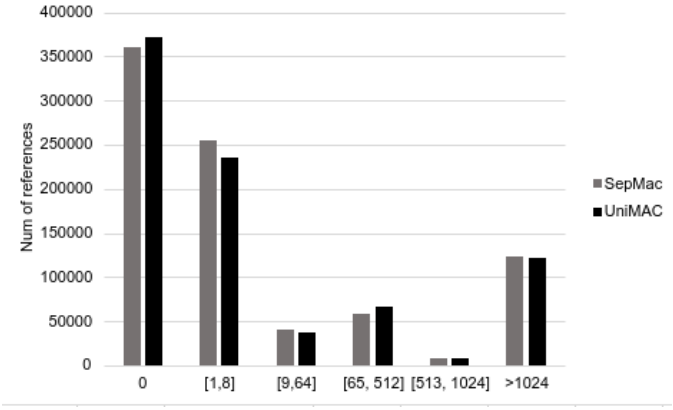

Fig. 11: Reuse distance of MACs of the benchmark fdtd2d

bandwidth utilization, the performance bottleneck is the extra memory traffic generated by metadata accesses. Only after the memory bottleneck is addressed, the limitation due to AES throughput may be exposed. Some memory-intensive benchmarks, such as srad_v2, streamcluster, 2Dconvolution, fdtd2d and lbm also exhibit slight performance improvement when the number of AES engines is reduced from 2 to 1. The reason is that these benchmarks have relatively high memory bandwidth utilization and large numbers of metadata write backs. Delaying some memory accesses due to limited AES throughput leads to a change in warp scheduling decisions as well as the cache access patterns, which results in small performance increase.

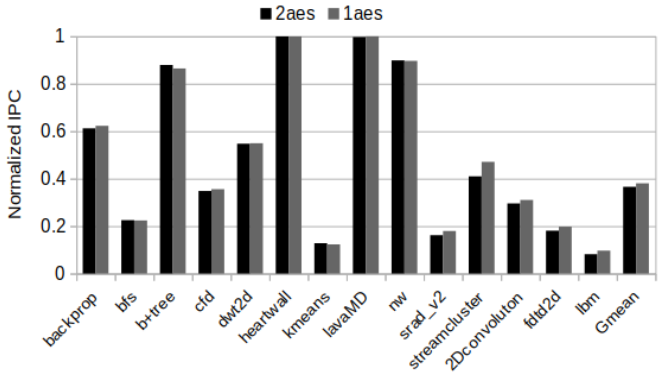

Fig. 12: Normalized IPC with different numbers of AES engines in each memory partition

F. Die Area

GPUs are highly-parallel processors, and most of their die area is dedicated for computational resources such as SIMT cores. For example, our baseline GPU models Nvidia QV_100 (Quadro GV100), which is fabricated using 12nm FinFET Nvidia (FNN) technology with 21.1 billion transistors integrated on a die with the size of 815mm^2 [9]. It integrates 80 SMs and each SM has 64 FP32 cores, 32 FP64 and 8 Tensor cores, which means a total of 5120 FP32 cores, 2560 FP64 cores and 640 Tensor cores.

TABLE VI: Die area of the AES engine

-	Tech	Die Area
JSSC'11 [17]	45nm	0.15mm^2
JSSC'19 [23]	130nm	13241μm^2
JSSC'20 [13]	14nm	4900 μm^2

TABLE VII: Scaled down die area of the AES engine and caches

-	Area(mm^2)/tech	Area(mm^2)/12nm
AES engine	0.0049/14nm	0.0036
64KB cache	0.125821/32nm	0.01769
96KB cache	0.128101/32nm	0.01801

To evaluate the die area required for counter-mode encryption, we estimate the area of both AES engine and the metadata caches. We list the results from prior works on AES design in Table VI and scale the most recent design [13] to the 12nm technology as shown in Table VII. From the table, we can see that the area of one AES engine is estimated as 0.0036 mm^2. As there are 32 memory partitions, the total area for 32/64 AES engines is 0.1152/0.2304 mm^2.

We use CACTI v6.5 to estimate the die area for the caches. As CACTI reports the area estimation using the 32nm technology, we also scale the results down to the 12nm technology, as shown in Table VII. As CACTI does not support modeling of small caches like 2KB, we report the area estimate of 64KB as 64KB is the aggregated capacity of one type of metadata caches in 32 partitions.

To make room for the die area required by the AES engines and metadata caches, we choose to reduce the L2 cache size. Since each L2 bank is 96KB, we also use CACTI to estimate the area. To accommodate 32 AES engines, the L2 cache capacity would need to be reduced by 0.1152 mm^2/0.01801mm^2*96KB = 614KB. Similarly, the metadata caches will occupy 0.01769*3 = 0.05307 mm^2 on chip area, and hence reducing the L2 capacity by 0.05307/0.01801*96KB = 283KB. If we assume MAC units have similar die area compared to AES engines, the security related hardware resources will reduce the L2 capacity by 614+614+298 = 1526KB in total (24.84% L2 cache capacity).

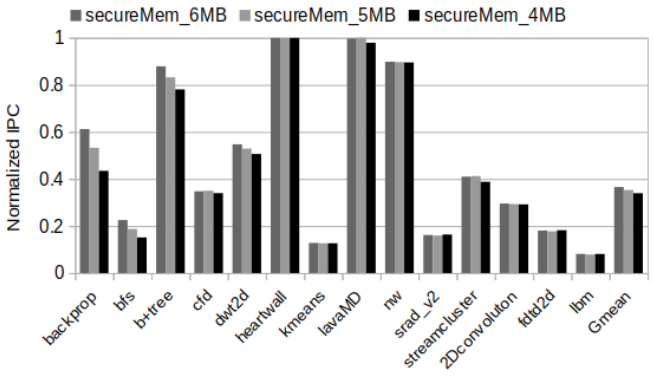

Fig. 13: Normalized IPC with different L2 cache capacities

To evaluate the performance impact of the reduced L2 capacity, Fig. 13 shows the normalized performance for different L2 cache sizes, ranging from 4MB to 6MB. From the figure, we can see that although many benchmarks are not sensitive to the L2 cache capacity, a few of them show relatively high performance degradation. To better understand the reasons, we also report the L2 cache miss rate in Fig. 14. From the figure, we can see that some medium-memory-intensive benchmarks show high sensitivity to L2 capacity. It is expected since computation-intensive benchmarks (e.g., heartwall and lavaMD) have small numbers of L2 accesses whereas highly memory intensive benchmarks such as lbm or fdtd2d have very high L2 miss rates. Changing L2 capacity has little impact on these workloads.

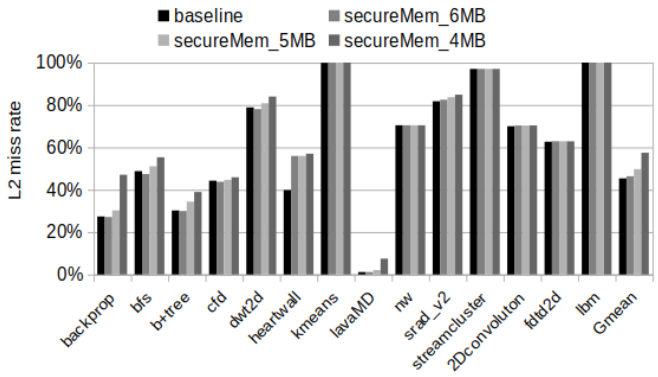

Fig. 14: L2 cache miss rate

VI. Direct Encryption

In this section, we evaluate direct encryption and explore the option of different levels of integrity protection. The designs that we evaluate in this section are listed in Table VIII.

A. Performance Overheads of Direction Encryption

As direct encryption exposes the decryption latency to the critical path of memory read accesses, it can slowdown the performance heavily on CPUs [31]. However, GPUs are designed for high-throughput computation by exploiting the high degrees of TLP. Hence, GPUs are able to tolerate long operation latency. It is expected that adding some encryption latency to the memory access would not hurt the GPU performance significantly.

We model the direct encryption with different encryption/decryption latencies upon our baseline GPU and the performance results are shown in Fig. 15. As expected, direct encryption does not affect the GPU performance much. When the encryption latency varies from 40 cycles to 160 cycles, the IPC slowdown is 1.33%, 3.02%, 5.93%, respectively on average. Some benchmark, e.g., b+tree, nw and streamcluster, show more than 10% slowdowns at the high encryption latency of 160 cycles. The reasons are different among these three benchmarks: (1) benchmark streamcluster suffers from high L2 miss rate (97.02%) as shown in Fig. 14, which leads to

TABLE VIII: Evaluated designs for direct encryption

Scheme	What It Represents
direct_x	direct encryption with different encryption latency of x cycles
ctr	counter-mode memory encryption without any integrity protection
ctr_bmt	counter-mode encryption with BMT to protect counter integrity
ctr_mac_bmt	counter-mode encryption with BMT and MACs
direct_mac	direct encryption with MACs.
direct_mac_mt	direct encryption with MACs and MT.

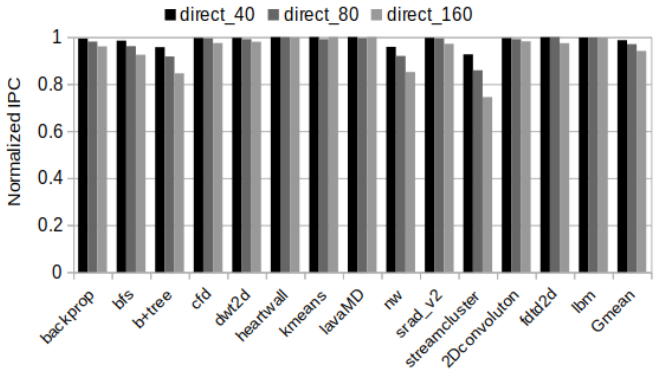

Fig. 15: Normalized IPC of direct encryption with different encryption latencies.

many memory read accesses, (2) benchmark nw is limited by the small kernel (shown in Table IV) such that they do not have enough threads to hide the encryption latency. (3) benchmark b+tree shows interesting results because it's neither bounded by kernel size nor L2 miss rate. Digging deeper, we find that with a high encryption latency of 160 cycles, b+tree suffers from 2.42X dram stall time compared the baseline GPU. Our results show that given longer dram stall time, all the available warps in b+tree suffer 12.2% more pipeline cycles to wait data from memory on average, and thus slowdown the performance. Although high encryption/decryption latency does not impact the GPU performance significantly, we choose 40 cycles as the default encryption/decryption latency, which is consistent with prior work on AES encryption for CPUs [15].

B. Direct Encryption *vs.* Counter-mode Encryption

To better understand the trade-offs between direct encryption and counter-mode encryption, we examine their performance in Fig. 16. From Fig. 16, we can make two observations. First, as discussed above, the performance impact of direct encryption is almost negligible. Second, compared with direct encryption, counter-mode encryption without integrity checks can lead to a relatively high performance overhead (33.06% on average, and up to 66.44% for the memory-intensive workload lbm). As we studied in section V, the main reason is that counter-mode encryption will generate additional memory traffic to fetch/store counters from/to the off-chip main memory.

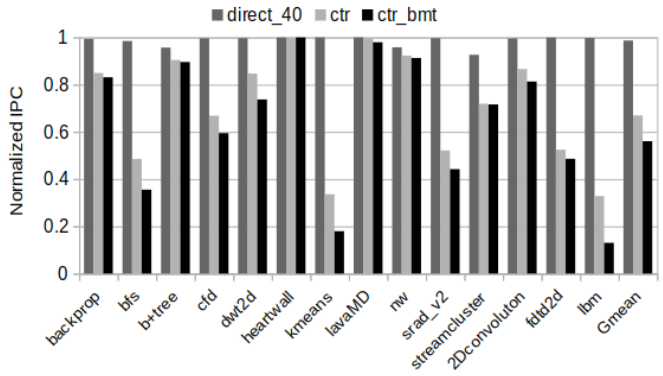

Fig. 16: Normalized IPC of direct encryption and counter-mode encryption

Moreover, as we discussed in section II-C, counter-mode encryption *fundamentally relies* on counter integrity protection to provide data confidentiality. The reason is that the counters are used for encryption and decryption. Without integrity checks of the counters, the GPU cannot tell whether a counter has been altered by the attacker or not. In this case, an attacker may be able to manipulate the counters to recover the plaintext. It can be illustrated as follows. Let us use P as the plaintext, C as the ciphertext, and K as the secret key of AES. Then the ciphertext is generated by $C = E_K(A||Ctr) \oplus P$, where A is the address of the memory block and Ctr is the counter. P can be recovered with $C \oplus E_K(A||Ctr)$ if Ctr can be controlled by the attacker. Hence, with counter-mode encryption, BMT is needed anyway to provide integrity protection to the counters stored in the off-chip memory.

As we can see from Fig. 16, adding the integrity protection to counters further increases the performance overhead, 43.94% on average for all the benchmarks in our study.

In summary, our evaluation suggests that if the memory only needs to be encrypted, direct encryption would be a better choice for GPUs.

C. Integrity Protection

Memory encryption can provide data confidentiality. However, it cannot provide memory integrity protection when the attackers have the ability to tamper the memory contents. With counter-mode encryption, data integrity protection is provided by stateful MACs, and counter integrity protection is provided by BMT. With direct encryption, one can choose to build MAC upon cyphertext with/without a Merkle Tree (MT). The MT is needed to prevent replay attacks and can be built with the MACs as its leaf nodes.

We model these schemes and compare the performance results in Fig. 17. For fairness, we assume the same on chip resource used by the metadata caches. In counter-mode encryption, as mentioned in Section V, a 2KB on-chip metadata cache is used for each type of metadata and a total of 6KB on-chip resource is used in each memory partition. In direct encryption with MAC, we model a MAC cache with the size of 6KB. In direct encryption with both MAC and MT, we model a 3KB MAC cache and 3KB MT cache (the MACs will not access the MT cache as they are cached in the MAC cache).

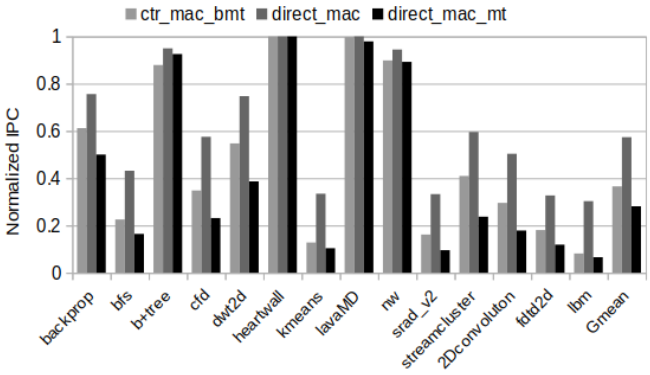

Fig. 17: Normalized IPC of direct encryption and counter-mode encryption with integrity protection

From Fig. 17, we can make two observations. First, given the same on-chip resource for metadata caches, direct encryption with MAC can still perform better than counter-mode encryption scheme with BMT (42.65% **vs.** 63.45% IPC slowdown on average). Second, MT can have high performance impact upon direct encryption. Combined with MACs, it can slowdown the GPUs IPC by 71.87% on average. The reason is that with the same memory protection range, the height of MT (7-level 16-ary tree) is higher than the BMT (6-level 16-ary tree). On one hand, a higher hash tree means that the integrity verification process (every newly fetched MAC block must be authenticated) needs to traverse a longer path and potentially incurs more memory traffic to fetch/store the tree nodes from/to off-chip memory. On the other hand, a higher hash tree, although just 1 level higher, has much larger capacity (due to the arity), which leads to high contention for the MT cache.

VII. Conclusions

In this paper, we perform detailed performance analysis of different secure memory architecture designs for accelerators like GPUs. From our study, we conclude that we need to architect GPU secure memory differently from it in CPUs. Our key observations include: (1) Due to the latency-hiding capability, direct encryption can be a better alternative to counter-mode encryption for GPUs if only data confidentiality is needed. (2) The AES throughput limitation can be reasonably addressed with one pipelined AES engine in each memory partition. (3) To support integrity verification, both counter-mode encryption and direct encryption need to be further optimized to reduce the performance overhead. In either design, the key bottleneck is the memory traffic generated by security metadata accesses. (4) Metadata caches can reduce the memory traffic and separate metadata caches can be a better choice than unified metadata caches for GPUs. (5) The use of sectored L2 cache on GPUs necessities MSHRs for the metadata caches to reduce the memory traffic.

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] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, , and K. Skadron., "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, 2009.

[2] V. Costan and S. Devadas, "Intel sgx explained," Cryptology ePrint Archive, Report 2016/086, 2016.

[3] cott Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to gpu codes," in *To Appear In Proceedings of Innovative Parallel Computing*, 2009.

[4] B. Gassend, G. E. Suh, D. Clarke, M. van Dijk, and S. Devadas, "Caches and hash trees for efficient memory integrity verification," in *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, 2003.

[5] S. Gueron, "A memory encryption engine suitable for general purpose processors," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 204, 2016. [Online]. Available: http://eprint.iacr.org/2016/204

[6] W. Hua, M. Umar, Z. Zhang, and G. E. Suh, "Guardnn: Secure dnn accelerator for privacy-preserving deep learning," *CoRR*, vol. abs/2008.11632, 2020. [Online]. Available: https://arxiv.org/abs/2008.11632

[7] W. Hua, M. Umar, Z. Zhang, and G. E. Suh, "Mgx: Near-zero overhead memory protection with an application to secure dnn acceleration," *CoRR*, vol. abs/2004.09679, 2020. [Online]. Available: https://arxiv.org/pdf/1804.06826.pdf

[8] I. Jang, A. Tang, T. Kim, S. Sethumadhavan, and J. Huh, "Heterogeneous isolated execution for commodity gpus," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*, 2018.

[9] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the nvidia volta gpu architecture via microbenchmarking," *CoRR*, vol. abs/1804.06826, 2018. [Online]. Available: https://arxiv.org/pdf/1804.06826.pdf

[10] Z. H. Jiang, Y. Fei, and D. Kaeli, "A complete key recovery timing attack on a gpu," in *Symposium on High Performance Computer Architecture (HPCA)*, 2016.

[11] D. Kaplan, J. Powell, and T. Woller, "Amd memory encryption," 2016.

[12] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers., "Accel-sim: An extensible simulation framework for validated gpu modeling," in *proceedings of the 47th IEEE/ACM International Symposium on Computer Architecture(ISCA)*, 2020.

[13] R. Kumar, V. Suresh, M. Kar, M. A. Anders, H. Kaul, A. Agarwal, S. Hsu, G. K. Chen, R. K. Krishnamurthy, V. De, and S. K. Mathew, "A 4900$-\mu m$ 2 839-mb/s side-channel attack-resistant aes-128 in 14-nm cmos with heterogeneous sboxes, linear masked mixcolumns, and dual-rail key addition," *IEEE Journal of Solid-State Circuits (Volume: 55, Issue: 4, April 2020)*, 2020.

[14] R. B. Lee, P. C. S. Kwan, J. P. McGregor, J. Dwoskin, and Zhenghong Wang, "Architecture for protecting critical secrets in microprocessors," in *32nd International Symposium on Computer Architecture (ISCA'05)*, 2005.

[15] T. S. Lehman, A. D. Hilton, and B. C. Lee, "MAPS: understanding metadata access patterns in secure memory," in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2018, Belfast, United Kingdom, April 2-4, 2018*. IEEE Computer Society, 2018, pp. 33–43.

[16] Y. Liu, Y. Xie, and A. Srivastava, "Neural torjans," in *2017 IEEE International Conference on Computer Design (ICCD)*, 2017.

[17] S. K. Mathew, F. Sheikh, M. Kounavis, S. Gueron, A. Agarwal, S. K. Hsu, H. Kaul, M. A. Anders, and R. K. Krishnamurthy, "53 gbps native gf(2 4) 2 composite-field aes-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessors," in *IEEE Journal of Solid-State Circuits*, 2011.

[18] S. Pinto and N. Santos, "Demystifying arm trustzone: A comprehensive survey," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 130:1–130:36, 2019. [Online]. Available: https://doi.org/10.1145/3291047

[19] B. Rogers, , M. Prvulovic, and Y. Solihin, "Effective data protection for distributed shared memory multiprocessors," in *in Proceedings of the International Conference of Parallel Architecture and Compilation Techniques (PACT)*, 2006.

[20] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "Using address independent seed encryption and bonsai merkle trees to make secure processors os- and performance-friendly," *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2007.

[21] B. Rogers, C. Yan, S. Chhabra, M. Prvulovic, and Y. Solihin, "Single-level integrity and confidentiality protection for distributed shared memory multiprocessors," in *in Proceedings of the 14th International Symposium on High Performance Computer Architecture (HPCA-14*, 2008.

[22] B. Rogers, C. Yan, M. Prvulovic, and Y. Solihin, "Single-level integrity and confidentiality protection for distributed shared memory multiprocessors," in *HPCA*, 2008.

[23] A. Singh, M. Kar, S. K. Mathew, A. Rajan, V. De, and S. Mukhopadhyay, "Improved power/em side-channel attack resistance of 128-bit aes engines with random fast voltage dithering," *IEEE Journal of Solid-State Circuits*, 2019.

[24] Y. Solihin, *Fundamentals of Parallel Multicore Architecture*. London: Chapman and Hall/CRC, 2015.

[25] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W. mei W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," in *IMPACT Technical Report*, 2009.

[26] G. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "Efficient memory integrity verification and encryption for secure processors," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, 2003.

[27] G. E. Suh, C. W. O'Donnell, and S. Devadas, "Aegis: A single-chip secure processor," *IEEE Design Test of Computers*, 2007.

[28] D. L. C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural support for copy and tamper resistant software," in *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[29] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on gpus," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*, 2018.

[30] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving cost, performance, and security of memory encryption and authentication," in *Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA)*, 2006.

[31] J. Yang, Y. Zhang, and L. Gao, "Fast secure processor for inhibiting software piracy and tampering," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, 2003.

[32] Y. Zhang, L. Gao, J. Yang, and R. Gupta, "Senss: Security enhancement to symmetric shared memory multiprocessors," in *in Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2005.

[33] Y. Zou and M. Lin, "Fast: A frequency-aware skewed merkle tree for fpga-secured embedded systems," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2019.

[34] P. Zuo, Y. Hua, L. Liang, X. Xie, X. Hu, and Y. Xie, "Sealing neural network models in secure deep learning accelerators," *CoRR*, vol. abs/2008.03752, 2020. [Online]. Available: https://arxiv.org/pdf/1804.06826.pdf