RCoal: Mitigating GPU Timing Attack via Subwarp-based Randomized Coalescing Techniques

Gurunath Kadam
College of William and Mary
Williamsburg, VA
gakadam@email.wm.edu

Danfeng Zhang
Penn State University
State College, PA
zhang@cse.psu.edu

Adwait Jog
College of William and Mary
Williamsburg, VA
adwait@cs.wm.edu

***Abstract*—**

Graphics processing units (GPUs) are becoming default accelerators in many domains such as high-performance computing (HPC), deep learning, and virtual/augmented reality. Recently, GPUs have also shown significant speedups for a variety of security-sensitive applications such as encryptions. These speedups have largely benefited from the high memory bandwidth and compute throughput of GPUs. One of the key features to optimize the memory bandwidth consumption in GPUs is intra-warp memory access coalescing, which merges memory requests originating from different threads of a single warp into as few cache lines as possible. However, this coalescing feature is also shown to make the GPUs prone to the correlation timing attacks as it exposes the relationship between the execution time and the number of coalesced accesses. Consequently, an attacker is able to correctly reveal an AES private key via repeatedly gathering encrypted data and execution time on a GPU.

In this work, we propose a series of defense mechanisms to alleviate such timing attacks by carefully trading off performance for improved security. Specifically, we propose to randomize the coalescing logic such that the attacker finds it hard to guess the correct number of coalesced accesses generated. To this end, we propose to randomize: a) the granularity (called as subwarp) at which warp threads are grouped together for coalescing, and b) the threads selected by each subwarp for coalescing. Such randomization techniques result in three mechanisms: fixed-sized subwarp (FSS), random-sized subwarp (RSS), and random-threaded subwarp (RTS). We find that the combination of these security mechanisms offers 24- to 961-times improvement in the security against the correlation timing attacks with 5 to 28% performance degradation.

***Index Terms*—GPUs, Hardware Security, Coalescing**

I. Introduction

Graphics Processing Units (GPUs) are becoming an inevitable part of every computing system because of their ability to provide fast and energy-efficient computation. Given such ability, GPUs are also now being used to accelerate a variety of cryptographic algorithms. For example, the popular Advanced Encryption Standard (AES) algorithm [21] is known to achieve significant speedups on GPUs compared to CPUs [6], [9], [17], [23] as the AES algorithm exposes abundant thread-level parallelism to leverage high bandwidth and compute throughput of GPUs. With such increasing popularity of GPUs to accelerate security-sensitive applications, it is imperative to keep GPUs secure against a variety of side-channel attacks and other security vulnerabilities.

In this paper, we specifically focus on the correlation-based timing attacks on GPUs. In general, a correlation-based timing attack exploits the relationship between the secret data and its impact on the processing time of an application: the attacker sends a large number of data samples to calculate the correlation between the actual processing time and the secret data. Among the *guessed* values for the secret data, the one leading to the highest correlation is the actual secret data. Notably, the recent work from Jiang et al. [10] demonstrated a correlation-based timing attack on a remote GPU server. They exploited two observations. First, the last round private key byte directly affects the number of coalesced memory accesses in the last round and can be calculated deterministically given the encrypted text. Second, the number of coalesced accesses in the last round is correlated with the total execution time. With these two observations, an attacker can recover each key byte by picking the value that best correlates with the recorded total execution time from the remote GPU server[1].

The goal of this paper is to design low-overhead defense mechanisms to thwart timing attacks that exploit the memory coalescing in GPUs. To this end, a straightforward solution is to eliminate the correlation between the number of coalesced accesses and the total execution time by disabling the memory access coalescing mechanism completely. However, since the memory access coalescing is one of the key features in GPUs that optimizes the memory bandwidth consumption, the disabling of coalescing will incur a heavy performance due to increase in the number of memory accesses [10], [15], [16], [28]. To provide a better trade-off between security and performance, we propose *RCoal*, a series of three tunable coalescing mechanisms to guard against correlation-based timing attacks.

The first mechanism focuses on tuning the granularity at which threads are coalesced together, thereby increasing the number of coalesced accesses at a finer granularity. We call this technique as fixed-sized subwarp (FSS) defense mechanism, where the size of subwarp determines the coalescing granularity. FSS mechanism helps to reduce the correlation between the coalesced accesses and total execution time by reducing the variance in the coalesced accesses. Building on the first mechanism, the second mechanism focuses on randomly changing the size of each subwarp. We call this technique as random-sized subwarp (RSS) defense mechanism where the size of each subwarp affects the attacker's ability to correctly determine the number of coalesced accesses. The

[1]Section II presents more details on the attack.

final mechanism focuses on randomly changing the thread elements of each subwarp. We call this technique as random-threaded subwarp (RTS) defense mechanism as the coalescer picks random thread elements to form a subwarp. RTS can be applied to both FSS and RSS to further hinder the attacker's ability to determine the number of coalesced accesses correctly.

To the best of our knowledge, this is the first work to thwart timing attacks in GPUs via randomized coalescing techniques. In summary, this paper makes the following contributions:

- We generalize the correlation-based timing attack on GPUs and show that the regularity and determinism in memory access coalescing is a major security vulnerability.
- We propose three novel coalescing mechanisms to mitigate the timing attacks arising from memory access coalescing. These mechanisms revolve around carefully changing the size, number, and thread elements of a subwarp to reduce the correlation between the number of coalesced accesses and the total execution time.
- We present a detailed information-theoretical analysis to show that our randomized coalescing mechanisms can improve the GPU security by 24 to 961 times. Our extensive simulation results confirm the theoretical results and demonstrate that the improved security can be achieved at a performance loss of 5 to 28%.
- We propose a new metric called $RCoal\_Score$ that provides an opportunity for hardware engineers to tune the security and performance trade-off as per their requirements. We discuss two such security-performance trade-off designs and conclude that RSS and RTS mechanisms provide significant advantages towards performance and security, respectively.

II. Background

In this section, we briefly introduce a) the baseline GPU architecture and the process of memory access coalescing, b) the anatomy of AES encryption, and c) the baseline timing attack assumed in this paper.

A. Baseline GPU Architecture

Overview. Figure 1 shows a high-level schematic of the GPU architecture. A typical GPU consists of multiple cores, called as streaming multiprocessors (SMs) in NVIDIA terminology. Each SM takes advantage of the Single Instruction, Multiple Threads (SIMT) programming paradigm [14] to schedule multiple threads on its processing elements (PEs). These threads are scheduled at the granularity of a *warp*, which is essentially a collection of (usually 32) individual threads that execute a single instruction on the PEs in a lock step manner. Each SM can execute multiple warps concurrently in a multiplexed manner to hide the long global memory latencies and improve the utilization of core resources (e.g., register file, scratchpad memory). All SMs are connected to global memory partitions via an on-chip interconnect. In this paper, we evaluate the proposed techniques on a GPU architecture simulated using a cycle accurate GPU simulator – GPGPU-Sim [1]. More details on the simulated architecture are given in Table I.

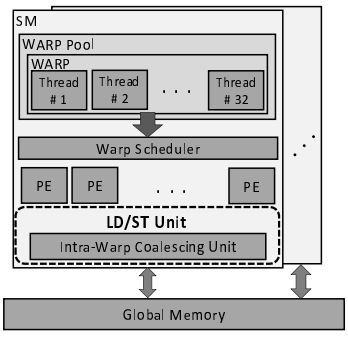


Fig. 1: Overview of Baseline GPU Architecture.

TABLE I: Key configuration parameters of the simulated GPU configuration.

Core Features	1400MHz core clock, SIMT width = 32 (16 × 2)
Resources / Core	32KB shared memory, 32KB register file, 15 SMs 32 threads/warp, one subwarp per coalescing unit
Features	immediate post dominator based branch divergence handling
Memory Model	6 GDDR5 Memory Controllers (MCs), FR-FCFS scheduling 16 DRAM-banks, 4 bank-groups/MC, 924 MHz memory clock Global linear address space is interleaved among partitions in chunks of 256 bytes [4] Hynix GDDR5 Timing [7], t_{CL} = 12, t_{RP} = 12, t_{RC} = 40, t_{RAS} = 28, t_{CCD} = 2, t_{RCD} = 12, t_{RRD} = 6
Interconnect	1 crossbar/direction, 1400MHz interconnect clock, islip VC and switch allocators

Memory Access Coalescing. One of the effective ways to improve the collective performance of the concurrently executing threads on GPUs is to optimize the global memory bandwidth. To this end, several techniques such as intra-warp memory access coalescing, inter- and intra-warp request merging via miss status handling registers (MSHRs), sectoring [28], and L1/L2 caching have been proposed for GPUs. In this work, we focus on intra-warp memory access coalescing technique, which merges multiple memory requests from different threads of the same warp in to as few cache line sized coalesced memory accesses as possible.

The coalescing unit (part of LD/ST unit of the SM) performs the agglomeration of memory requests from the threads in a warp at a *subwarp* level, where the number of subwarps is an architectural parameter. If the threads of a particular subwarp request nearby data within a contiguous block of the memory, their requests are coalesced together to avoid redundant accesses. Therefore, if the memory access size, subwarp size, and thread-data pattern (e.g., if/when thread to table index mapping is known) are known, the number of memory accesses can be calculated accurately. As per CUDA programming guide [24], the scalar threads from the same warp can be coalesced together (subwarp size of 1), at a half-warp basis (subwarp size of 2) or at a quarter-warp basis (subwarp size of 4). The subwarp size is decided based on the size of the memory request from each thread. The generated coalesced accesses are serviced at the rate that matches with the underlying cache/memory bandwidth. To correctly simulate the number of coalesced accesses as that of in the baseline attack model (explained later in the section), we assume subwarp size to be 1 in our baseline architecture.

To understand the effect of subwarps on coalescing, consider an example with warp comprising of four threads under two different cases employing the number of subwarps (*num-subwarp*) as 1 and 2, respectively, as shown in Figure 2. We assume that four threads generate four accesses and if perfectly coalesced will generate one coalesced access (memory block). When all the threads are considered together for coalescing (i.e., Case 1: *num-subwarp* is 1), only three coalesced accesses are generated as the requests from the second and third thread

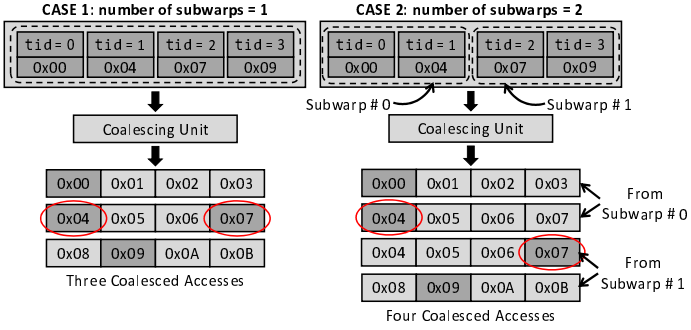


Fig. 2: Effect of subwarps on memory coalescing.

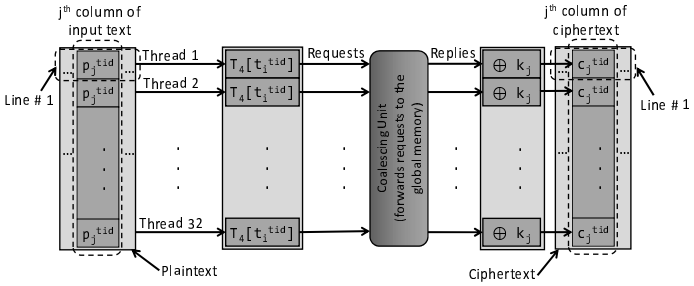


Fig. 3: Last round execution of AES-128 algorithm. The t_i in $T4[t_i]$ represents the index of the table lookup operation. k_j and c_j represent the jth byte of the last round key and ciphertext, respectively. tid is the thread id within a warp.

are coalesced into one request. When *num-subwarp* is 2 (Case 2), the coalescing is performed independently for each subwarp. Consequently, two coalesced accesses per subwarp (in total four) are generated.

B. AES Encryption

Basics. The Advanced Encryption Standard (AES) [21] is a widely used symmetric-key algorithm. The AES standard specifies 128, 192, and 256 bits as the standard key lengths. Without losing generality, we focus on AES-128, which employs a 128-bit key to encrypt the plaintext. AES-128 algorithm consists of 10 rounds each with its own round key of 16 bytes, which is generated from the encryption key. In each round, subBytes() transformation (details of other transformation can be found in prior works on AES [6], [9], [17], [23]) performs a table look-up operation on the substitution (S-box) table. In the last round, a table look-up operation is performed on the T_4 S-box table followed by bitwise XOR operation with the last round key. This operation is expressed by Equation 1 for the j^{th} byte of output ciphertext (c_j) and i^{th} input state of the last round (t_i, table lookup index) [6], [10]. $T_4[\,]$ represents the last round S-box table look-up operation whose result is XORed with j^{th} byte of the last round key (k_j).

$$c_j = T_4[t_i] \oplus k_j \tag{1}$$

GPU Implementation of AES Encryption. A CUDA implementation of AES divides the plaintext across multiple parallel threads to improve GPU throughput. Each thread performs encryption on one line (block) of the plaintext. Therefore, each warp consists of 32 threads performing 32 different encryptions. The line to thread mapping is sequential and deterministic in the baseline implementation. If the size of the plaintext exceeds 32 lines, then it is divided sequentially among several warps. For example, a plaintext with 1024 lines will employ 32 warps each executing 32 lines of the plaintext. Figure 3 shows the encryption process for the last round on 32 threads of a single warp. Each thread performs encryption of a byte (p_j) of the input text, where j varies from 1 to 16. All threads of the warp work in a lock-step manner and perform the same table look up operation ($T_4[t_i]$) with different values of t_i. The accesses are coalesced together by the coalescing unit, and when the replies come back, all threads use the same last round key (k_j) to generate one column of the ciphertext c_j as per Equation 2. In Equation 2, tid is the thread index.

$$c_j^{tid} = T_4\left[t_i^{tid}\right] \oplus k_j \tag{2}$$

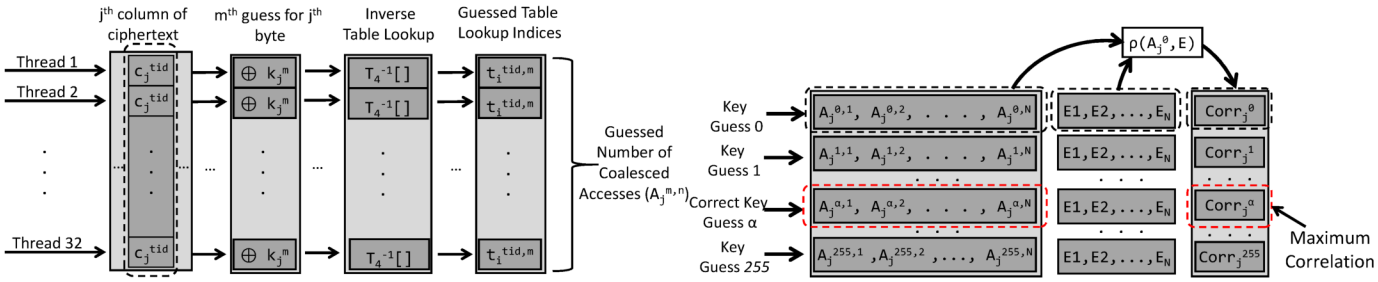
C. Baseline Timing Attack

In this paper, we use the correlation timing attack proposed by Jiang et al. [10] as the baseline attack. The attack model assumes that the attacker sends a large number of plaintexts to a remote GPU AES encryption server. The attacker collects the ciphertexts and records the total execution time for each plaintext. The goal is to correctly find all 16 last round key bytes by exploiting a key observation that there is a high correlation between the number of memory accesses and the total execution time on GPU. The baseline attack targets the last round key since it is the most vulnerable round and key expansion is invertible (i.e., it is possible to derive the original private key from any round key) [22]. The observation is that each table lookup index in the last round can be computed from a byte of the last round key (k_j) and the corresponding byte of ciphertext (c_j), independent of other ciphertext bytes (as shown in Equation 3). Thus, the attacker is able to observe the security leakage separately at per-byte level.

$$t_i = T_4^{-1}[c_j \oplus k_j] \tag{3}$$

Figure 4 shows the attack process for recovering the j^{th} last round key byte (k_j). The attack process has two major steps. The first step involves a *guessed* key value k_j^m where m ranges from 0 to 255. According to Equation 3, the table lookup index of each thread ($t_i^{tid,m}$) can be computed, as shown in Figure 4a. Once the indices are obtained for all threads, the attacker can calculate the expected number of coalesced accesses ($A_j^{m,n}$) for the n^{th} plaintext with the known and deterministic behavior of coalescing (in our configuration, 16 consecutive table elements are mapped sequentially to the same memory block). This particular attack assumes *num-subwarp* to be 1 (i.e., all threads in the warp are processed together for coalescing). This first step is repeated for all possible 256 key byte guesses for the j^{th} byte and for N plaintext samples. As a result, a memory access matrix is generated as shown in Figure 4b. Each row of the matrix corresponds to the number of guessed memory accesses for a particular key guess (m) across N plaintext samples (A_j^m).

The second step involves calculating the correlation



(a) Guessing the Coalesced Accesses from the Table Look up Indices **(b) Guessing the Last Round Key**

Fig. 4: Overview of the process of guessing one of the correct last round key byte (k_j). $A_j^{m,n}$ is the number of memory requests for m^{th} guess of the j^{th} last round key byte using n^{th} plaintext. n varies between 1 to N, where N is the number of plaintext samples. m varies from 0 to 255 and j varies from 1 to 16.

between each row (key guess) of the memory access matrix with the last round execution time (E) to encrypt each plaintext (collected by the attacker). Since both the total and last round execution time correlate with last round coalesced accesses (shown in Figure 5), the guessed key value (α) is correct for k_j if it has the maximum correlation value ($corr_j^{\alpha}$) with E. For the rest of the paper, we assume a stronger attack with the capability of accessing last round execution time as compared to the realistic attack, which is weaker due to the noise in the total execution time.

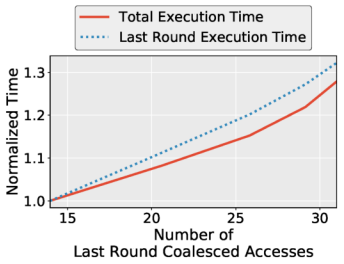
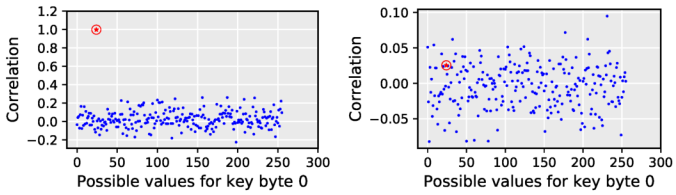


Fig. 5: Relationship between Last Round and Total Execution Time.

III. Motivation and Goals

The primary reason behind the success of the baseline correlation timing attack is the deterministic behavior of memory access coalescing that allows accurate calculation of the coalesced accesses generated. To verify this on our GPGPU-Sim based simulation environment, we plot the correlation values ($corr_j^m$) of all 256 possible values of m for 0^{th} key byte (k_0, j=0). We calculate this correlation value between the coalesced accesses from the attack and the execution time of the last round of AES-128. From Figure 6a, we observe that the correlation value is the highest (highlighted in red and encircled) for the correct value of the 0^{th} key byte among all other guess values. We observe this trend for all 16 last round key bytes indicating that we can successfully guess all of them.

As a first step towards defending against the baseline attack, we aim to eliminate the relationship between the number of coalesced accesses and the last round execution time by disabling the coalescing mechanism. As a result, the number of coalesced accesses will always be 32 (i.e., the worst case scenario) from a warp with 32 threads. We executed the same baseline attack with coalescing disabled to find that there is no correlation between the number of coalesced accesses and the last round execution time. Consequently, we could not successfully guess any of the key byte. Figure 6b shows the plot of correlation values against the possible values of the 0^{th} key byte. The correlation of the correct key byte is very close to zero, so as that of other key guesses.



(a) Coalescing Enabled **(b) Coalescing Disabled**

Fig. 6: Effect of Coalescing on the Recovery of 0^{th} Last Round Key Byte (k_0): a) Recovery is Successful when Coalescing is Enabled, b) Recovery is Unsuccessful when Coalescing is Disabled.

Although disabling the coalescing is an effective technique to defend against the baseline correlation timing attack, absence of memory access coalescing degrades the GPU performance and energy efficiency significantly [10], [15], [16], [28]. Our own experiments show that the performance degrades by up to 178% for AES-128 algorithm encrypting plaintext of 1024 lines. Also, the data movement (i.e., the number of memory accesses) increases by 2.7×. Therefore, disabling the coalescing is not an attractive solution from the perspective of GPU efficiency.

In this paper, our goal is to design randomized coalescing techniques to carefully balance the security and performance trade-offs. Our techniques exploit two primary shortcomings of the GPU AES implementation that lead to the successful correlation timing (baseline) attack. First, all threads of a warp are grouped in a single subwarp for coalescing. As a result, the calculation of number of coalesced accesses becomes straightforward: a) determine the requested table look up indices, and then b) given that the table elements are sequentially mapped to the memory blocks and the size of each block is known, determine the number of memory blocks (coalesced accesses) required. Second, because all threads of the warp were considered together for coalescing, the order in which the threads are grouped together had no impact on the coalescing. However, if coalescing is performed at a subwarp-level (with number of subwarp being more than one), the order of grouping the threads would affect the total number of coalesced accesses depending on which threads fall into the

same subwarp. To address these two shortcomings, we focus on the following three randomized coalescing aspects to weaken the correlation between the coalesced accesses calculated by the baseline attack and the execution time from the encryption.

• **Number of Subwarps:** We choose the number of subwarps that is unknown to the baseline attacker. The benefit of using subwarps is that the attacker may not be able to correctly estimate the number of coalesced accesses. Further, with a large number of subwarps, the variance in the number of coalesced accesses decreases, entailing more number of plaintext samples to establish a weak correlation. This weak correlation reduces the information leakage over the timing channels. We call this defense mechanism as *Fixed Subwarp Size (FSS)*, as the size of subwarp chosen by the defense mechanism is fixed.

• **Size of Subwarps:** In case the attacker knows the number of subwarps (or calculates it based on the timing information), we aim to increase the strength of the defense mechanism by randomizing the number of threads per subwarp such that the total number of threads per warp still remains 32. This randomness makes the number of coalesced accesses harder to estimate (same reasoning as FSS) even if the number of subwarps is known to the attacker. We call this defense mechanism as *Random Subwarp Size (RSS)* as the size of each subwarp is chosen randomly.

• **Thread Elements of Subwarps:** Our last mechanism is focused on further enhancing the GPU security by randomizing the thread elements of each subwarp (Random-threaded Subwarp (RTS)). It introduces additional randomness in the number of coalesced accesses generated. Note that RTS can be combined with both FSS and RSS defense mechanisms.

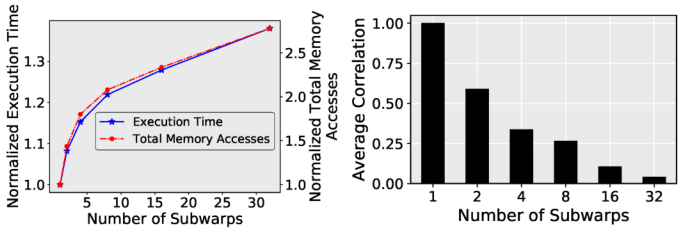
IV. Subwarp Based Defense Mechanisms

In this section, we discuss a series of subwarp-based defense mechanisms that are designed to weaken the deterministic memory coalescing logic in GPUs. By doing so, the baseline attack that leverages the knowledge of memory coalescing logic will find it difficult to correctly guess the last round key bytes, thereby improving the security of the GPU-based systems.

A. Fixed-sized Subwarps (FSS)

In the baseline attack, the attacker assumes that the number of subwarps (*num-subwarp*) is 1, and hence, all threads are processed together for coalescing. In our first defense mechanism, fixed size subwarps (FSS), we break this assumption by choosing a value of *num-subwarp* that is unknown to the attacker. In order to understand the impact of *num-subwarp* on performance, consider Figure 7a. We find that the total execution time increases with increase in the value of *num-subwarp*. It is because a large *num-subwarp* leaves few threads for being coalesced together thereby reducing coalescing possibilities across the threads within a warp. This leads to increased number of coalesced accesses resulting in the performance loss.

Advantages of FSS. Although FSS has disadvantage in terms of performance, we find that such a mechanism can improve the GPU security against the baseline attack. It is because a value of *num-subwarp* other than 1 will generate different number of coalesced accesses than the baseline attack, which



(a) Execution Time **(b) Average correlation**

Fig. 7: Performance of FSS enabled AES with respect to number of subwarps: a) Execution Time and Total Memory Accesses per plaintext with increasing number of subwarps, b) Average of correlations between the last round execution time and the last round memory accesses for all key bytes. The last round memory accesses are calculated assuming correct values of a key byte and the number of subwarps for coalescing to be one.

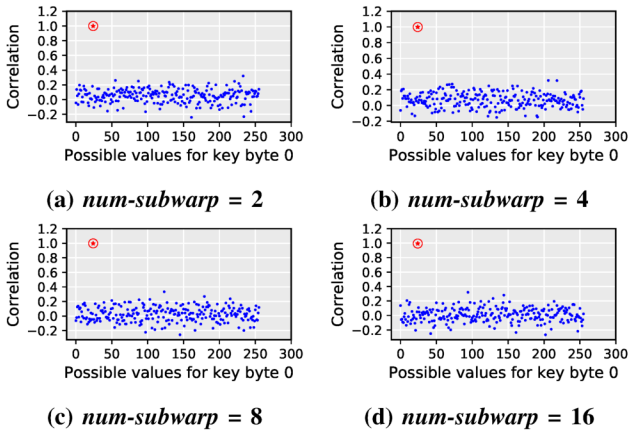
assumes *num-subwarp* to be 1. Therefore, the attacker will find it hard to guess the correct key byte as the correlation between the estimated number of coalesced accesses and the execution time reduces. To understand this further, we evaluate FSS-enabled GPU under the baseline attack. Figure 7b shows the average correlations for the correct guesses of all 16 key bytes of the last round key. As expected, we observe that the correlation between the last round execution time and coalesced accesses calculated from the attack reduces with the increase in the value of *num-subwarp*. Therefore, a high number of samples would be required to correctly guess the last round keys depending on the *num-subwarp* value.

Limitations of FSS. We evaluate the security of FSS mechanism when the attacker knows or correctly calculates the value of *num-subwarp*. For example, the calculation can be done based on the significant execution time differences across *num-subwarp* values (Figure 7). By repeatably measuring the execution time for encryption for a plaintext, an attacker can determine which *num-subwarp* is used by the remote GPU server. We call this new attack as "FSS Attack", where the attacker first calculates the number of last round coalesced accesses generated per subwarp. Next, since the last round execution time correlates with the last round coalesced accesses across a complete warp, the attacker sums up the last round coalesced accesses across all subwarps in a warp. Algorithm 1 illustrates the steps to calculate the number of last round coalesced accesses per warp.

We evaluate the effectiveness of FSS-enabled GPU under the FSS-attack in Figure 8, which illustrates that the attacker is able to establish a high correlation between the number of coalesced accesses and the last round execution time using the FSS attack. Using Algorithm 1, the attacker can calculate the last round memory accesses across the whole warp as observed during the encryption. Therefore, the attacker can establish a high correlation between the calculated number of last round coalesced accesses and the observed last round execution time to successfully recover the last round key. For *num-subwarp* = 32 (not shown), the variation in the numbers of last round coalesced accesses generated across all plaintexts drops to 0. Subsequently, the correlation between the number of last round

Algorithm 1 Algorithm for FSS attack to calculate the number of last round coalesced accesses for a given key byte guess while considering *num-subwarp*.

$k_j \leftarrow guess\_value$
$last\_round\_mem\_accesses \leftarrow 0$
for $i = 0 \rightarrow$ *num-subwarp* **do**
  $mem\_accesses\_subwarp[i] \leftarrow 0$
for $grp = 0 \rightarrow$ *num-subwarp* **do**
  for $i = 0 \rightarrow \frac{32}{num\text{-}subwarp}$ **do**
    $holder[i] \leftarrow 0$
% comment: $line$ represents plaintext line
% comment: LEN represents the total number of lines in the plaintext
  for $line = \frac{grp*LEN}{num\text{-}subwarp} \rightarrow \frac{(grp+1)*LEN}{num\text{-}subwarp}$ **do**
    $holder[T4^{-1}[cipher[line][j] \oplus k_j] >> 4] + +$
  for $i = 0 \rightarrow \frac{32}{num\text{-}subwarp}$ **do**
    if $holder[i]! = 0$ **then**
      $mem\_accesses\_subwarp[grp] + +$
for $i = 1 \rightarrow$ *num-subwarp* **do**
  if $mem\_accesses\_subwarp[i]! = 0$ **then**
    $last\_round\_mem\_accesses \leftarrow last\_round\_mem\_accesses + mem\_accesses\_subwarp[i]$



(a) ***num-subwarp*** **= 2** **(b)** ***num-subwarp*** **= 4**

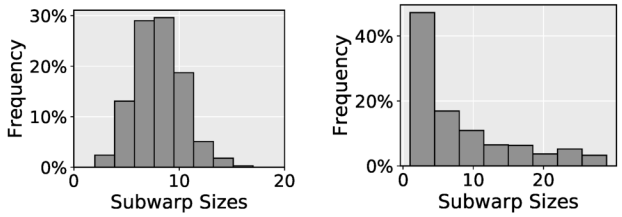
(c) ***num-subwarp*** **= 8** **(d)** ***num-subwarp*** **= 16**

Fig. 8: Fixed Size Subwarp (FSS) mechanism against FSS attack.

coalesced accesses from Algorithm 1 and the observed last round execution time also drops to 0. Therefore, FSS enabled GPU is immune to the correlation timing attacks only when *num-subwarp* = 32 but at the cost of performance. *In summary, we conclude that the stand alone FSS-enabled GPU cannot provide adequate security against the generalized correlation timing attacks. Therefore, improved defense mechanisms are required.*

B. Random-sized Subwarp (RSS)

In Random-sized Subwarp (RSS) defense mechanism, the size of each subwarp is randomly chosen by the hardware. It implies that the coalescing unit considers different numbers of threads per subwarp for coalescing together. This results in increased randomness in the number of last round coalesced accesses generated per warp leading to reduction in correlation. We consider two distributions to generate sizes of subwarps: normal and skewed. Figure 9 shows these distributions for 1000 plaintexts and with the assumption of *num-subwarp* = 4. In the normal distribution case, the mean of the distribution is close to that of the FSS scenario (32/*num-subwarp*). According to empirical results (not shown), this implies that security and performance of RSS with normal distribution is similar to that of FSS.



(a) Normal Distribution. **(b) Skewed Distribution.**

Fig. 9: Subwarp size distribution of RSS for *num-subwarp* = 4.

In order to improve security and performance over FSS, we consider skewed size distribution for the RSS mechanism that leads to significant differences in the subwarp sizes. This has two benefits. First, due to the mismatch in the subwarp sizes, the attacker will find it hard to correctly calculate the last round coalesced accesses using Algorithm 1. Second, the skewed distribution also results in an improved performance, since the opportunities for coalescing increases with the subwarp size. Further, we ensure that the skewed distribution considers all possible subwarp size combinations equally likely and no subwarp is empty (a formalization can be found in Section V-B3). In summary, we use skewed distribution for RSS to improve security and performance.

C. Random-threaded Subwarp (RTS)

In addition to the size and number of subwarps, we consider an additional level of randomness that comes from the choice of threads that form a particular subwarp. By random allocation of the threads to different subwarps, we eliminate the in-order mapping of threads to the subwarp. We find that such random formation of subwarps significantly changes the number of expected coalesced accesses as the threads processed for coalescing in a subwarp are chosen randomly. We define this technique as Random-threaded Subwarp (RTS).

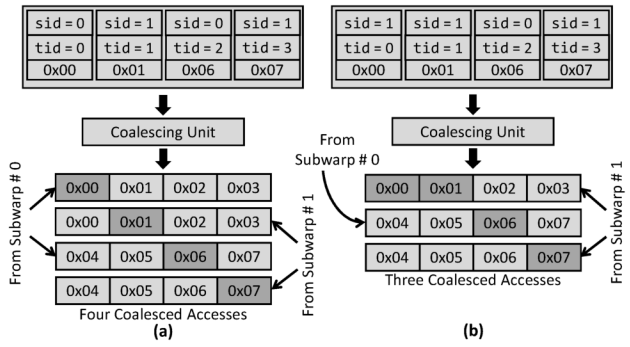


Fig. 10: Effects of different defense mechanisms on coalescing for *num-subwarp* = 2: a) FSS+RTS and b) RSS+RTS. sid **represents subwarp id and** tid **represents thread id.**

RTS can be applied on top of both FSS and RSS, called as FSS+RTS and RSS+RTS, respectively. Extending the example used in Section II-A to study the impact of subwarps on coalescing, Figures 10a and 10b illustrate examples of FSS+RTS

and RSS+RTS with 4 threads and 2 subwarps, respectively. In the case of FSS+RTS, the size of both subwarps is 2 but threads are not mapped in order. For example, subwarp 0 (sid = 0) has two threads 0 and 2 (tid = 0 and 2) instead of threads 0 and 1. Therefore, four coalesced accesses are generated. In the case of RSS+RTS, sizes of the subwarp are different: 1 and 3. Consequently, the mapping of one of threads is changed (i.e., tid = 0 is now mapped to sid = 1) leading to total three coalesced accesses. In summary, we find that RSS can help in reducing the number of coalesced accesses while providing randomness (along with RTS) for better security.

D. Implementation Details

In order to implement the proposed subwarp based defense mechanisms, we modify the coalescing unit to allow flexibility in processing of threads for memory access coalescing. Figure 11 shows a schematic of the memory coalescing unit (MCU) of GPU (the additional hardware logic for security is shaded). As described by Leng et al. [16], each MCU contains a multi-entry pending request table (PRT). Each entry in the PRT table stores the thread index (tid), the base and offset addresses of the memory requests from the threads, and their sizes. An entry is logged when a memory request is issued from a thread. We add an additional subwarp-id (sid) field to identify which threads should be coalesced together. The subwarp-id and thread-id mapping is set by the hardware logic at the beginning of the application execution and does not change during the execution. The logic is dependent on the adopted defense mechanism. In case of FSS and RSS, the bits are set based on the chosen value of *num-subwarp* and the sizing mechanism. The subwarp-ids are allotted in order, that is, first group of threads will belong to the first subwarp with sid set to 0 and so on. For RTS, the available $sids$ are allotted randomly to the threads in a warp. The additional hardware overhead of our mechanisms is related to the addition of subwarp-id field to each PRT entry. The number of concurrent warp scheduler per SM in our case is two. Therefore, for each SM, the nominal overhead would be 32 × 2 × 5 bits (to represent 32 maximum possible values of sid) = 320 bits.

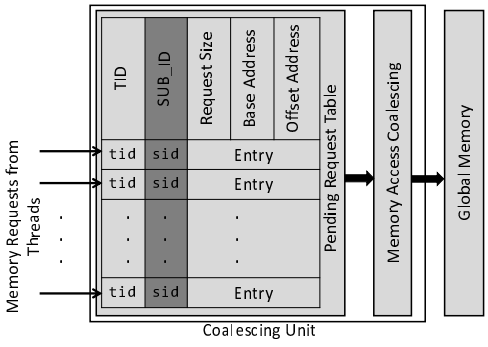


Fig. 11: Modified Coalescing Unit to realize FSS, RSS, and RTS defense mechanisms. The additional hardware required is the field to store subwarp-id (sid) for each thread.

E. Corresponding Attacks

Similar to the FSS attack, which generalized our baseline attack, we assumed that the attacker is aware of the details of our defense mechanisms implemented on GPU. Therefore, for each defense mechanism, we modified Algorithm 1 to mimic the respective defense mechanism on the attacker's side. For example, against the RSS+RTS enabled GPU, the corresponding attack algorithm simulates RSS-like subwarp size distribution along with random allocation of threads to subwarps within a warp as in RTS. We assume corresponding attacks in the rest of the paper.

V. Theoretical Security Analysis

A. Analytical Model

To measure the security strength of the defense mechanisms introduced in Section IV, we inspect a natural metric of the (expected) number of samples needed to successfully launch the correlation timing attack.

To estimate that, we use T to represent the *measurement vector*, a vector of the encryption times for a sample set using the actual key. For the j^{th} last round key byte k_j, we use $\widehat{U}_{k_j^m}$ to represent the *estimation vector* of k_j^m, a vector of the coalesced accesses for the same sample set if $0 \leq m \leq 255$ were the actual value of k_j. The correlation attack essentially tries to find the value $\widehat{m}$ that maximizes the correlation with the measure vector:

$$\widehat{m} = \arg\max_{m}(\rho(T, \widehat{U}_{k_j^m}))$$

We follow the derivation in [20], [31] to estimate the number of needed samples, S, for a successful attack as follows:

$$S = 3 + 8 \times \Big(\frac{Z_\alpha}{\ln\big(\frac{1+\rho(T,\widehat{U})}{1-\rho(T,\widehat{U})}\big)}\Big)^2 \approx \frac{2 \times Z_\alpha^2}{\rho^2(T,\widehat{U})} \tag{4}$$

where $\widehat{U}$ is a short hand for $\widehat{U}_{k_j^{\widehat{m}}}$, ρ represents the correlation and Z_α is the quantile of the standard normal distribution for α, the desired success rate of an attack. With $\alpha = 0.99$, $2 \times Z_\alpha^2$ is approximately 11. Z_α is proportional to α. So the smaller α is, the smaller S is (i.e., fewer samples are needed).

To estimate $\rho^2(T, \widehat{U})$, we observe that (as shown in Figure 5) the total execution time of AES is proportional to the number of last-round coalesced accesses. Hence, we can draw on the latter in the analytical model[2]. Hence, let us assume that U is the actual vector of number of coalesced accesses from the lookup of table T_4 with respect to the key byte k_j (Equation 1 in Section II-B). We can rewrite Equation 4 as

$$\begin{aligned} S \propto \frac{1}{\rho^2(U,\widehat{U})} &= \Big(\frac{\mu(U \times \widehat{U}) - \mu(U)\mu(\widehat{U})}{\sigma(U)\sigma(\widehat{U})}\Big)^{-2} \\ &= \Big(\frac{\mu(U \times \widehat{U}) - \mu^2(U)}{\sigma^2(U)}\Big)^{-2} \end{aligned} \tag{5}$$

where μ and σ, as standard, respectively represent the mean and standard deviation of a random variable. The last equation is true since U and $\widehat{U}$ are identically distributed.

[2]We note that using the number of coalesced accesses rather than the execution time assure a lower bound on the number of samples since the later is noisier than the former.

B. Analysis of Defense Mechanisms

To make the analysis general, we assume there are in total M subwarps and N threads. Moreover, we assume that each lookup table may map to R memory blocks. As discussed in Section II, our configuration has $N = 32$ and $R = 16$.

We first define three useful definitions.

Definition 1: Given m threads, if each thread accesses one of n memory blocks in a uniform way, then the number of coalesced accesses, $\mathfrak{N}_{m,n}$, obeys the following distribution:

$$P(\mathfrak{N}_{m,n} = i) = \frac{1}{n^N} \frac{n!}{(n-i)!} \left\{ {m \atop i} \right\}$$

where $\left\{ {m \atop i} \right\}$ denotes the Stirling number of the second kind. Here, $\left\{ {m \atop i} \right\}$ represents the ways of partitioning m threads into i non-empty subsets; $\frac{n!}{(n-i)!}$, i-permutations of n, represents the ways of forming i non-empty subsets from n memory blocks.

It is infeasible to compute Equation (5) by enumerating all possible mappings from threads to memory blocks since there are in total R^N possibilities ($16^{32} = 2^{128}$ when $N = 32$ and $R = 16$). However, we note that with RTS, the number of coalesced accesses only depends on the *frequency* of the R memory blocks, which is defined as follows.

Definition 2: For R memory blocks and n threads, we define a *frequency set* $\mathcal{F}$ as

$$\{(f_1, \ldots, f_R) \mid f_1 + \cdots + f_R = n\}$$

where $f_i \in \mathcal{F}$ represents the frequency of accessing the i-th memory block among the n threads.

Given a frequency vector $F \in \mathcal{F}$, we note that the "contribution" of each memory block to the number of last-round coalesced accesses U is independent. Hence,

Definition 3: Given a frequency sequence $F \in \mathcal{F}$ and a vector $C = \{c_1, \cdots, c_m\}$ that specifies the capacity of each subwarp, if each thread uniformly accesses one of the $|F|$ memory blocks, then the number of coalesced accesses, written as $\mathfrak{M}_{F,C}$, satisfies

$$\mu(\mathfrak{M}_{F,C}) = \sum_{f_i \in F} \sum_{c_j \in C} (1 - C_{f_i}^{S-c_j} / C_{f_i}^{S})$$

where C_n^m denotes the binomial coefficient and $S = \sum_{1 \le j \le n} c_j$.
Here, $C_{f_i}^{S-c_j} / C_{f_i}^{S}$ is the probability that the j-th subwarp is empty and $\mu(\mathfrak{M}_{F,R})$ is the sum of the expectations for each subwarp and each memory block.

Next, we derive the (normalized) samples needed for a successful attack for each defense mechanism. We skip the theoretical analysis for the RSS mechanism since it requires enumerating all possible mappings from threads to memory blocks rather than the frequency set, making it infeasible for the calculation. Instead, we provide the empirical results for the RSS mechanism in Section VI.

1) FSS: With sufficiently random plaintexts, the probability that one thread accesses one of the R memory blocks is $1/R$. Hence, for each subwarp with size N/M, the number of coalesced accesses is $\mathfrak{N}_{N/M,R}$. Since each subwarp is independent, we have

$$\mu(U) = M \times \mu(\mathfrak{N}_{N/M,R}) \qquad \sigma(U) = M \times \sigma(\mathfrak{N}_{N/M,R})$$

TABLE II: Security analysis results with $N = 32$ and $R = 16$, where N is the number of threads and R is the number of memory blocks. Here, M is the number of subwarps and S is the number of samples normalized to FSS with $M = 1$ case.

	ρ			S (normalized)		
M	FSS	FSS+RTS	RSS+RTS	FSS	FSS+RTS	RSS+RTS
1	1.00	1.00	1.00	1	1	1
2	1.00	0.41	0.20	1	6	25
4	1.00	0.20	0.15	1	24	42
8	1.00	0.09	0.11	1	115	78
16	1.00	0.03	0.05	1	961	349
32	0.00	0.00	0.00	∞	∞	∞

For $\mu_{U \times \widehat{U}}$, we note that given any sequence of memory blocks being accessed by threads, U is identical to $\widehat{U}$. Hence, $\mu(U \times \widehat{U}) = \mu(U^2) = \sigma^2(U) + \mu^2(U)$.

2) FSS+RTS: The random permutation does not affect $\mu(U)$ and $\sigma(U)$. For $\mu(U \times \widehat{U})$, $(U|F)$ and $(\widehat{U}|F)$ are independent and identical for any $F \in \mathcal{F}$. Hence, the term is equivalent to

$$\sum_{F \in \mathcal{F}} P(F) \mu^2(U|F) \tag{6}$$

Here, $P(F)$ is the probability of seeing the frequency vector F. Among all R^N combinations of N memory accesses, $C_N^{f_1} C_{N-f_1}^{f_2} \cdots C_{N-\sum_{1 \le j \le R-1} f_j}^{f_R} = \frac{(N)!}{\Pi_{f_i \in \mathcal{F}} f_i!}$ match F. Hence, we have $P(F) = \frac{(N)!}{\Pi_{f_i \in \mathcal{F}} f_i!} \times \frac{1}{R^N}$. Moreover, $\mu(U|F)$ is the same as $\mu(\mathfrak{M}_{F,\{N/M, \cdots, N/M\}})$ since each subwarp has size N/M.

3) RSS+RTS: We use U_i to represent the coalesced accesses of the i-th subwarp. With RSS, U_i and U_j are not independent. Hence, we cannot compute $\sigma(U)$ as for FSS.

However, given the size of each subwarp, U_i and U_j are independent for any $1 \le i, j \le M$. We use $\mathcal{W} = \{(w_1, \cdots, w_M) \mid \sum_{1 \le i \le M} w_i = N \wedge \forall_{1 \le i \le M}.\ w_i \neq 0\}$ to denote all possible non-empty sizes of subwarps under RSS. Due to uniformity, $P(W) = \frac{1}{|\mathcal{W}|}$ for any $W \in \mathcal{W}$.

For $\mu(U)$, we have $\mu(U) = \sum_{W \in \mathcal{W}} P(W) \mu(U|W) = \sum_{W \in \mathcal{W}} P(W) \sum_{w_i \in W} \mu(U_i|w_i)$ where $\mu(U_i|w_i)$ is the same as $\mu(\mathfrak{N}_{w_i,R})$. For $\sigma(U)$, we know $\sigma^2(U) = \mu(U^2) - \mu^2(U)$ and

$$\begin{aligned} \mu(U^2) &= \sum_{W \in \mathcal{W}} P(W) \mu(U^2|W) \\ &= \sum_{W \in \mathcal{W}} P(W) \big(\sum_{1 \le i \le M} \sigma^2(U_i|w_i) + \mu^2(U|W) \big) \end{aligned}$$

Here, $\sigma^2(U_i|w_i) = \mu(U_i^2|w_i) - \mu^2(U_i|w_i)$ and $\mu(U|W) = \sum_{1 \le i \le M} \mu(U_i|w_i)$ due to independence. We note that $(U_i|w_i)$ is $\mathfrak{N}_{w_i,R}$ and $(U_i^2|w_i)$ is $(\mathfrak{N}_{w_i,R})^2$. So these terms can be computed via Definition 1.

For $\mu(U \times \widehat{U})$, we can reuse Equation 6 since with RTS, $(U|F)$ and $(\widehat{U}|F)$ are independent and identical. Similar to FSS+RTS, $\mu(U|F) = \sum_{W \in \mathcal{W}} P(W) \mu(\mathfrak{M}_{F,W})$ in this case.

C. Results

We use a Python script to compute the correlation and normalized sample size for a successful attack. The results are summarized in Table II.

As expected, when $M = 32$, we have $\rho = 0$ and $S = \infty$ because in this case, each thread is mapped to one subwarp

and hence, $U = 32$ regardless of the last-round key. Otherwise, FSS is the least secure ($\rho = 1, S = 1$), where the key can be revealed easily (as shown in Figure 8). For both FSS+RTS and RSS+RTS, increasing the number of subwarps reduces ρ and increases S. We note that FSS+RTS is more secure than RSS+RTS for $M = 8$ and 16 though the latter adds randomness to the subwarp size. We hypothesize the reason for this improved security is that one of the subwarps has large size under RSS+RTS most of the times (see, Figure 9). In this case, the correlation between measurement vector and estimation vector is higher than that under FSS+RTS. Moreover, the empirical results are consistent with the evaluation (Section VI).

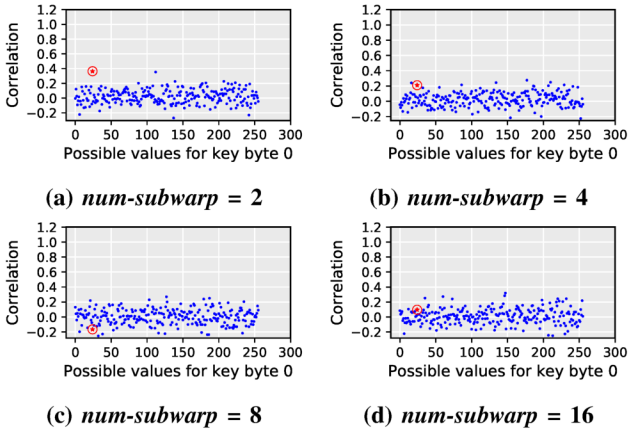
Since, in practice, the attacker may observe only the noisy total execution time rather than the last-round coalesced accesses as assumed for the theoretical analysis, the absolute value of needed samples for a successful attack is very large. We note that the FSS mechanism with $M = 1$ is the same as the (baseline) architecture used in [10]. As reported in [10], one million timing samples are needed (if the timing data measured is clean) in this case, and the samples can be collected within 30 minutes. Hence, we estimate that under FSS+RTS with $M = 16$, around one billion samples (refer Table II) are needed for a successful attack. Although such an attack is theoretically possible, it is not practical since collecting timing samples alone may take (30 minutes $*$ 961 $\approx$) 20 days.

VI. Experimental Analysis of Security and Performance

In this section, we present empirical results to support the theoretical results discussed in Section V. We first analyze the security of each mechanism by assessing the key recovery ability using the scatter plots and by inspecting the reduction in the correlation values. Subsequently, we discuss the effects of the proposed mechanisms on performance and data movement. All the results are collected on a GPU architectural simulator, GPGPU-Sim [1]. Note that it is impractical to execute the attack experiments with a large number of plaintexts on a simulator. However, because of the less noisy environment in the simulators compared to the real hardware, we were able to demonstrate the baseline attack with 100 plaintext samples (each with 32 lines) in Section III. Therefore, for a fair comparison, we use the same number of samples to demonstrate the effectiveness of our defense mechanisms.

A. Effect on Security

FSS+RTS Attack on FSS+RTS enabled GPU. Figure 12 shows four scatter plots each with a different value of *num-subwarp*. Each scatter plot shows the correlation values between the last round execution time and the number of last round coalesced accesses calculated from the FSS+RTS attack algorithm for all the guessed values of the key byte 0. We notice that as *num-subwarp* increases the last round key byte recovery gets difficult as opposed to the standalone FSS defense mechanism. This enhancement in the security is due to the random noise added by the RTS mechanism. Although the FSS+RTS attack implements the random thread allocation in the attack algorithm, it is hard to correctly match the thread allocation order to the one used during the encryption. We conclude that the randomization in the thread allocations allows FSS+RTS to improve GPU security.

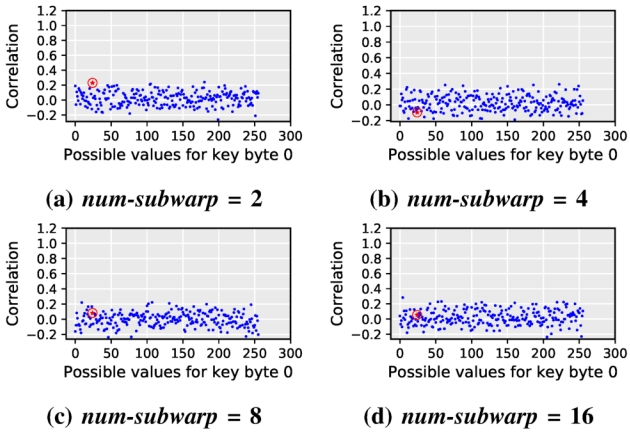


(a) ***num-subwarp*** **= 2** **(b)** ***num-subwarp*** **= 4**

(c) ***num-subwarp*** **= 8** **(d)** ***num-subwarp*** **= 16**

Fig. 12: FSS+RTS defense mechanism against FSS+RTS attack.

RSS Attack on RSS enabled GPU. Figure 13 shows four scatter plots each with a different value of *num-subwarp*. Each scatter plot shows the correlation values between the last round execution time and the number of last round coalesced accesses calculated from the RSS attack algorithm for all the guessed values of the key byte 0. For *num-subwarp* greater than 2, we observe that the key byte recovery is difficult as the correlation value for the correct guess is no longer the highest. The drop in the correlation value against the RSS attack is due to the random nature of the subwarp sizing employed in RSS defense mechanism. This random subwarp sizing is changed between the plaintexts and is hard to mimic during the correlation timing attack. Therefore, with the random sizing of the subwarps, the RSS defense mechanism offers improved security as compared to the FSS defense mechanism.



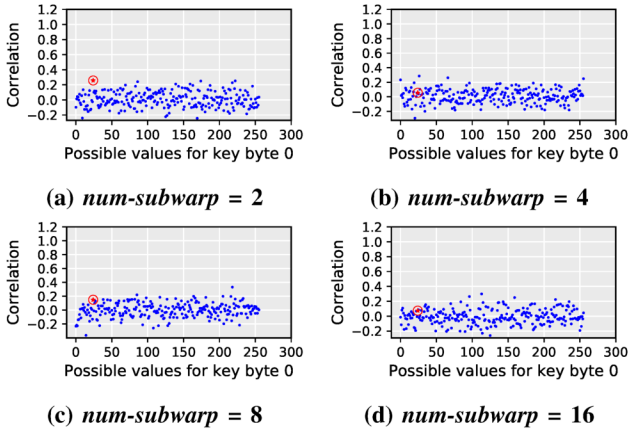
(a) ***num-subwarp*** **= 2** **(b)** ***num-subwarp*** **= 4**

(c) ***num-subwarp*** **= 8** **(d)** ***num-subwarp*** **= 16**

Fig. 13: RSS defense mechanism against RSS attack.

RSS+RTS Attack on RSS+RTS enabled GPU. Figure 14 shows four scatter plots each with a different value of *num-subwarp*. Each scatter plot shows the correlation values between the last round execution time and the number of last round coalesced accesses calculated from the RSS+RTS attack algorithm for all the guessed values of the key byte 0. Similar

to FSS+RTS and RSS defense mechanisms, we notice that the recovery of the correct value of the key byte is difficult with the RSS+RTS defense mechanism for *num-subwarp* greater than 2. The RSS+RTS leverages the randomness in the subwarp sizing and in the thread allocation to the subwarps, which is very difficult to replicate in the RSS+RTS attack. We conclude that RSS+RTS offers security benefits over the FSS defense mechanism.



(a) ***num-subwarp*** **= 2** **(b)** ***num-subwarp*** **= 4**

(c) ***num-subwarp*** **= 8** **(d)** ***num-subwarp*** **= 16**

Fig. 14: RSS+RTS defense mechanism against RSS+RTS attack.

Security Comparisons. Figure 15 compares the security offered by the different defense mechanisms proposed in this work using the average correlation. As noted in Section IV, the FSS defense mechanism fails to reduce the correlation as the FSS attack can correctly calculate the last round coalesced accesses. For FSS+RTS, RSS and RSS+RTS defense mechanisms, we observe a decrease in correlation for *num-subwarp* = 2 and 4. We observe slight fluctuations in the respective correlations for RSS and RSS+RTS defense mechanisms for *num-subwarp* = 8 and 16 due to increased randomness in the coalescing. This randomness affects the incorrect guesses of the key bytes as well and results in an overall improved security. Also, we notice that RSS+RTS outperforms all other defense mechanisms for *num-subwarp* = 2 and 4, while FSS+RTS outperforms rest of the defense mechanisms for *num-subwarp* = 8 and 16. For *num-subwarp* = 2 and 4, the RSS+RTS introduces randomness in the coalescing at subwarp sizing as well as at thread to subwarp allocation level. Therefore, the correlation values decreases more in RSS+RTS than in FSS+RTS. However, for *num-subwarp* = 8 and 16, the variance in the last round coalesced accesses is lower in the case of FSS+RTS compared to RSS+RTS. It is because FSS+RTS has more subwarps with the same size compared to RSS+RTS. These findings are corroborated by the theoretical analysis (Table II).

B. Effect on Performance and Data Movement

Figure 16 shows the execution time and the total number of memory accesses with respect to *num-subwarp* for each defense mechanism. In Figure 16a, we notice an increase in the total memory accesses with respect to *num-subwarp*. This increase in the memory accesses is attributed to the subwarp based defense mechanisms – FSS and RSS – which reduce

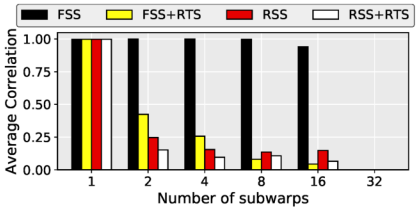
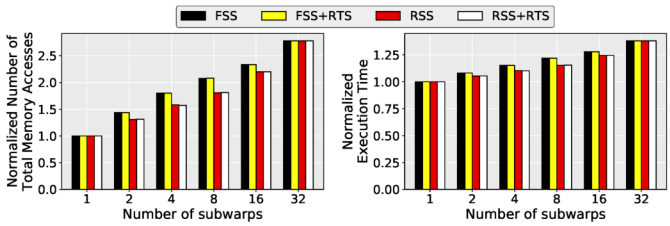


Fig. 15: Comparison between the security offered by FSS, FSS+RTS, RSS and RSS+RTS based on the average of correlations between the last round coalesced accesses for all key bytes and the last round execution time observed during the encryption. The last round coalesced accesses are calculated using the corresponding attacks.

the possibility of memory accesses coalescing by dividing the threads of a warp into different subwarps. Therefore, we observe an increase in the execution time as the *num-subwarp* increases (Figure 16b). We make two more observations. First, the RTS mechanism does not affect the performance. Although the order of the thread allocation to the subwarps dictates the number of coalesced accesses in a subwarp and hence across the entire warp, the overall effect on performance averages itself out over a large number of plaintexts.



(a) Number of Memory Accesses **(b) Execution Time**

Fig. 16: Performance and Data Movement Comparisons between FSS, FSS+RTS, RSS, and RSS+RTS.

Second, the RSS-based mechanisms (RSS and RSS+RTS) show a slightly lower increase in the memory accesses compared to the FSS-based mechanisms (FSS and FSS+RTS). This is because the skewed distribution of subwarp sizes in the RSS-based mechanisms (Section IV-B) increases the possibility of a few subwarps to be larger than others. Therefore, on average, the RSS and RSS+RTS defense mechanisms perform better than the FSS and FSS+RTS defense mechanisms.

C. Evaluating the Trade-off Between Security and Performance

We define the $RCoal\_Score$ metric, as per Equation 7, to allow hardware engineers to achieve a trade-off between the security and performance as per design requirements.

$$RCoal\_Score = \frac{S^a}{execution\_time^b} \tag{7}$$

In the above equation, S is the square of the inverse of the average correlation values calculated from the attack as shown in Figure 15. The parameters (a and b) can be set by the hardware engineer to put an appropriate emphasis on either security or performance. For example, Figure 17a shows the $RCoal\_Score$ values for a security-oriented system with $a = 1$

and $b = 1$. We note that FSS+RTS with *num-subwarp* = 8 and 16 is best suited for improving GPU security, albeit with a considerable loss in the performance. For a performance-oriented system, we set $a = 1$ and $b = 20$, as shown in Figure 17b. In this case, for *num-subwarp* = 8 and 16, RSS+RTS scores higher than FSS+RTS since it offers an improvement in the performance at a moderate loss in security.

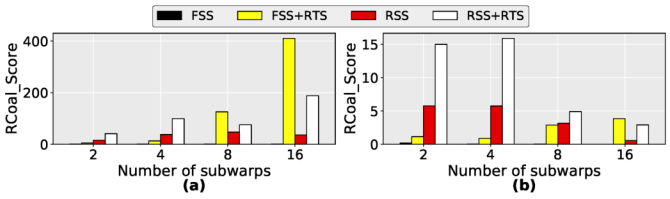


Fig. 17: Comparison between the FSS, FSS+RTS, RSS and RSS+RTS defense mechanisms based on the RCoal score against the corresponding attacks: a) Security-oriented system with $a = 1$ and $b = 1$, b) Performance-oriented system with $a = 1$ and $b = 20$.

D. Case Study: Plaintext with 1024 Lines

We evaluate the scalability of the subwarp based defense mechanisms by increasing the plaintext size to 1024 lines. To negate the ill-effects of the warp scheduling noise during the security evaluation of the defense mechanisms, we correlate the last round coalesced accesses calculated from the corresponding attacks with the last round coalesced accesses observed during the encryption. It is evident that if the attacker is able to correctly estimate the last round coalesced accesses during the attack, then the correlation will be highest for the correct guess of the key byte leading to a successful recovery of the key. We discuss the security and the performance of each mechanism with respect to *num-subwarp*.

Security. Figure 18a shows the average correlation for all key bytes of the last round key for each defense mechanism. As expected, we notice that the average correlation decreases for FSS+RTS, RSS and RSS+RTS mechanisms for *num-subwarp* greater than 1. We conclude that our defense mechanisms improve security on GPUs encrypting large plaintexts as well.

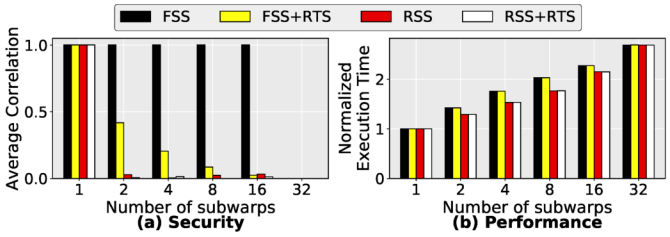


Fig. 18: Effects of the defense mechanisms on security against the corresponding attacks and performance with respect to the number of subwarps for plaintext with 1024 lines: a) Average of correlations between the last round coalesced accesses from the attack and the execution for all key bytes, b) Execution time (normalized to the case when *num-subwarp*=1) with respect to the number of subwarps.

Performance. Figure 18b shows the execution time for each mechanism normalized to the baseline case of *num-subwarp* set to 1. As in the case of plaintext with 32 lines, we note that the RTS mechanism does not affect the execution time. Also, the the execution time increases with *num-subwarp*. Additionally, as earlier, RSS-based mechanisms increase the coalescing possibilities and deliver better performance than the FSS-based mechanisms. In conclusion, we observe that RSS+RTS mechanism offers an improved security with performance degradation in the range of 29 to 76% for *num-subwarp* = 2, 4, and 8. This indicates that the defense mechanisms presented in this work scale well with the plaintext size.

VII. Discussion and Future Work

In the context of RCoal, we discuss the following two future research directions.

- The current implementation of RCoal spans over the entire execution of the AES and assumes that all rounds are equally vulnerable [25]. The advantage of such an implementation is that it does not require software support to identify the vulnerable portions (rounds) of the code. To enhance the performance further, RCoal can be limited only to the vulnerable part of the code. However, that would require software support to correctly identify the vulnerable portions of the code and hardware support to frequently turn coalescing on and off based on which warps are executing the vulnerable code at a given time. We leave the development of such hardware/software support as a part of the future work.
- We presented a series of defense mechanisms that focused only on the intra-warp coalescing techniques. Therefore, we disabled other bandwidth conserving optimizations in GPUs (e.g., MSHRs and caches). However, we believe our proposed intra-warp coalescing will be more effective if randomization is employed at all levels of the memory hierarchy. We leave the development of these randomization techniques as a part of the future work.

VIII. Related Work

To the best of our knowledge, this is the first work that proposes randomized coalescing mechanisms to thwart timing attacks in GPUs. In this section, we list works relevant to ours.

Timing attacks. Cryptographic algorithms implemented on CPUs have been the major targets of timing attacks. Those attacks exploit the fact that key-dependent memory accesses, such as table-lookups in AES, affect the memory access patterns and hence, the status of data cache. Hence, an attacker may infer private keys by observing the execution time of either a cryptographic algorithm (e.g., [2], [3], [5], [26]), or his own application if the data cache is shared (e.g., [5], [8], [35], [37]).

Pietro et al. [27] identified that the memory leaks are possible at various levels of GPU memory hierarchy, especially at software-managed scratchpad memory and register file. A recent work [11] exploits a new fine-grained timing channel caused by bank conflicts in a GPU's shared memory. A complete AES key recovery timing attack was first demonstrated on a commercial GPU architecture by Jiang et al. [10]. We have already extensively discussed this attack and proposed defense mechanisms that trade-off performance for security.

Timing channel mitigation. Several hardware-based timing attacks have been proposed in the context of CPUs [18], [19], [26], [32], [33], [34], [36]. Among those works, more related

are mechanisms based on randomization [19], [32], [33], [34]. Most of these works randomize the memory-to-cache mapping or the cache replacement policy, while our work proposes to randomize the coalescing behavior.

Coalescing and Bandwidth Saving Techniques in GPUs. Kloosterman et al. [15] proposed warp-pool, an enhanced inter-warp sharing mechanism to reduce global memory accesses. Rhu et al. [28] proposed cache sectoring mechanism to reduce unnecessary data fetches from global memory. A series of warp scheduling techniques [12], [13], [29], [30] have been proposed to reduce cache misses and improve memory bandwidth utilization. None of these works focused on hardware security issues, as we do in this paper.

IX. Conclusions

Our findings confirm that the deterministic nature of the coalescing logic is a major cause of security vulnerability in GPUs. To address this vulnerability, we propose a series of defense mechanisms that allow the coalescing logic to randomly change the number of coalesced accesses. Specifically, we propose to randomize: a) the granularity at which intra-warp coalescing is performed in the baseline architecture, and b) allocation of the thread elements per subwarp. Our theoretical and empirical results show that our randomized coalescing defense mechanisms significantly improve the GPU security at a modest performance loss.

Acknowledgment

The authors would like to thank the anonymous reviewers and the members of Insight Computer Architecture Lab at the College of William and Mary for their feedback. This material is based upon work supported by the National Science Foundation (NSF) grants (#1657336 and #1717532) and a start-up grant from the College of William and Mary. This work was performed in part using computing facilities at the College of William and Mary which were provided by contributions from the NSF, the Commonwealth of Virginia Equipment Trust Fund and the Office of Naval Research.

References

[1] A. Bakhoda *et al.*, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, April 2009, pp. 163–174.

[2] A. Bogdanov *et al.*, "Differential cache-collision timing attacks on AES with applications to embedded cpus," in *Topics in Cryptology–CT-RSA 2010*, ser. Lecture Notes in Computer Science, J. Pieprzyk, Ed., 2010, vol. 5985, pp. 235–251.

[3] J. Bonneau and I. Mironov, "Cache-collision timing attacks against AES," in *Cryptographic Hardware and Embedded Systems - CHES 2006*, ser. Lecture Notes in Computer Science, L. Goubin and M. Matsui, Eds. Springer Berlin Heidelberg, 2006, vol. 4249, pp. 201–215.

[4] GPGPU-Sim v3.2.1. Address mapping. Available: {http://gpgpu-sim.org/manual/index.php5/GPGPU-Sim_3.x_Manual#Memory_Partition}

[5] D. Gullasch *et al.*, "Cache games—bringing access-based cache attacks on AES to practice," in *Proc. IEEE Symp. on Security and Privacy (S&P)*, 2011, pp. 490–505.

[6] O. Harrison and J. Waldron, "AES Encryption Implementation and Analysis on Commodity Graphics Processing Units," in *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems*, ser. CHES '07, 2007.

[7] Hynix. Hynix GDDR5 SGRAM Part H5GQ1H24AFR Revision 1.0. Available: {http://www.hynix.com/datasheet/pdf/graphics/H5GQ1H24AFR(Rev1.0).pdf}

[8] G. Irazoqui *et al.*, "Wait a minute! A fast, cross-VM attack on AES," in *Research in Attacks, Intrusions and Defenses*, ser. Lecture Notes in Computer Science, A. Stavrou *et al.*, Eds., 2014, vol. 8688, pp. 299–319.

[9] K. Iwai *et al.*, "Aes encryption implementation on cuda gpu and its analysis," in *2010 First International Conference on Networking and Computing*, Nov 2010, pp. 209–214.

[10] Z. H. Jiang *et al.*, "A complete key recovery timing attack on a GPU," in *HPCA*, 2016.

[11] Z. H. Jiang *et al.*, "A Novel Side-Channel Timing Attack on GPUs," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*. ACM, 2017, pp. 167–172.

[12] A. Jog *et al.*, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," in *ASPLOS*, 2013.

[13] O. Kayiran *et al.*, "Neither More Nor Less: Optimizing Thread-level Parallelism for GPGPUs," in *PACT*, 2013.

[14] D. Kirk and W. W. Hwu, *Programming Massively Parallel Processors*. Morgan Kaufmann, 2010.

[15] J. Kloosterman *et al.*, "Warppool: Sharing requests with inter-warp coalescing for throughput processors," in *MICRO*, 2015.

[16] J. Leng *et al.*, "GPUWattch: Enabling Energy Optimizations in GPGPUs," in *ISCA*, 2013.

[17] Q. Li *et al.*, "Implementation and analysis of AES encryption on GPU," in *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*. IEEE, 2012.

[18] X. Li *et al.*, "Sapper: A language for hardware-level security policy enforcement," in *ASPLOS*, 2014.

[19] F. Liu and R. B. Lee, "Random fill cache architecture," in *MICRO*, 2014.

[20] S. Mangard, "Hardware countermeasures against dpa–a statistical analysis of their effectiveness," in *Cryptographers' Track at the RSA Conference*. Springer, 2004, pp. 222–235.

[21] F. P. Miller *et al.*, *Advanced Encryption Standard*. Alpha Press, 2009.

[22] M. Neve and J.-P. Seifert, "Advances on access-driven cache attacks on aes," in *Selected Areas in Cryptography*, vol. 4356. Springer, 2006, pp. 147–162.

[23] N. Nishikawa *et al.*, "High-performance symmetric block ciphers on cuda," in *2011 Second International Conference on Networking and Computing*, Nov 2011, pp. 221–227.

[24] NVIDIA, "Programming Guide." Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#axzz3DAGrtrOq

[25] D. A. Osvik *et al.*, "Cache attacks and countermeasures: The case of aes," in *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, ser. CT-RSA'06, 2006.

[26] D. Page, "Partitioned cache architecture as a side-channel defense mechanism," in *Cryptology ePrint Archive, Report 2005/280*, 2005. Available: http://eprint.iacr.org/2005/280.pdf

[27] R. D. Pietro *et al.*, "Cuda leaks: A detailed hack for cuda and a (partial) fix," *ACM Trans. Embed. Comput. Syst.*, 2016.

[28] M. Rhu *et al.*, "A Locality-Aware Memory Hierarchy for Energy-Efficient GPU Architectures," in *MICRO*, 2013.

[29] T. G. Rogers *et al.*, "Cache-Conscious Wavefront Scheduling," in *MICRO*, 2012.

[30] T. G. Rogers *et al.*, "Divergence-Aware Warp Scheduling," in *MICRO*, 2013.

[31] K. Tiri *et al.*, "An analytical model for time-driven cache attacks," in *International Workshop on Fast Software Encryption*. Springer, 2007, pp. 399–413.

[32] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *ISCA*, 2007.

[33] Z. Wang and R. B. Lee, "A novel cache architecture with enhanced performance and security," in *MICRO*, 2008.

[34] M. Yan *et al.*, "Secure hierarchy-aware cache replacement policy (sharp): Defending against cache-based side channel atacks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 347–360.

[35] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, 2014, pp. 719–732.

[36] D. Zhang *et al.*, "A hardware design language for timing-sensitive information-flow security," in *ASPLOS*, 2015.

[37] Y. Zhang *et al.*, "Cross-tenant side-channel attacks in PaaS clouds," in *Proc. ACM Conf. on Computer and Communications Security (CCS)*, 2014, pp. 990–1003.