Side-channel Attacks with Multi-thread Mixed Leakage

Yiwen Gao[1] and Yongbin Zhou[2,3]

[1]National University of Singapore, 3 Research Link, #03-02 i4.0 Building, Singapore 117602
e-mail: gaoywin@gmail.com
[2]SKLoIS, IIE, CAS, Beijing 100195, China
[3]School of Cyber Security, UCAS, Beijing 100049, China

Abstract – **Side-channel attacks are one of the greatest practical threats to security-related applications, because they are capable of breaking ciphers that are assumed to be mathematically secure. Lots of studies have been devoted to power or electro-magnetic (EM) analysis against desktop CPUs, mobile CPUs (including ARM, MSP, AVR, etc) and FPGAs, but rarely targeted modern GPUs. Modern GPUs feature their special and specific single instruction multiple threads (SIMT) execution fashion, which makes their power/EM leakage more sophisticated in practical scenarios. In this paper, we study side-channel attacks with leakage from SIMT systems, and propose leakage models suited to any SIMT systems and specifically to CUDA-enabled GPUs. Afterwards, we instantiate the models with a GPU AES implementation, which is also used for performance evaluations. In addition to the models, we provide optimizations on the attacks that are based on the models. To evaluate the models and optimizations, we run the GPU AES implementation on a CUDA-enabled GPU and, at the same time, collect its EM leakage. The experimental results show that the proposed models are more efficient and the optimizations are effective as well. Our study suggests that GPU-based cryptographic implementations may be much vulnerable to microarchitecture-based side-channel attacks. Therefore, GPU-specific countermeasures should be considered for GPU-based cryptographic implementations in practical applications.**

Index Terms—**Multi-Thread Mixed Leakage, GPU-specific Attacks, Cache Attacks, Elecro-Magnetic Analysis, Side-Channel Attacks.**

I. INTRODUCTION

SIDE-CHANNEL attacks have been a great concern in hardware security since it was first proposed more than 20 years ago [1]. The kind of attack draws so much attention from academia and industry communities because it turns out to be capable of breaking ciphers that are assumed to be mathematically secure. A lot of studies have been devoted to side-channel attacks against desktop CPUs, mobile CPUs (such as ARM, MSP and AVR) and FPGAs but rarely targeted GPUs, which have become one of important high-performance general-purpose computing platforms. Only recently, researchers studied side-channel attacks against GPU-based cryptographic implementations, thus having confirmed the vulnerabilities of GPUs to side-channel attacks. There have been studies on GPU-based cryptographic implementations, e.g. [2], [3], [4], [5], which feature their high performance, but may be susceptible to side-channel attacks.

Luo *et al.* proposed a power analysis attack against a GPU AES implementation [6], [7] (called Luo's attack hereinafter). Their approach is based on a naive leakage model, and thus introduces more noise. To reduce the noise, the GPU kernel has to run in a highly-occupied mode. The attack is simple and does not require aligned traces, which make their attack more practical. However, the attack is not efficient in practice due to the high-level noise. After that, Jiang *et al.* proposed timing attacks against GPU AES implementations [8], [9], [10], [11] (called Jiang's attacks hereinafter). The timing attacks, also called time-driven cache attack, make use of the time differences caused by microarchitectural features, namely *memory issue serialization* (MIS) and *memory bank conflict* (MBC), of caches equipped on target GPU hardware. Fomin also proposed a timing attack based on the MBC feature of GPU (called Fomin's attack hereinafter), but his attack targeted the second round encryption of AES, so a chosen-plaintext technique was employed to reduce practical complexity of the attack [12]. In addition, Gao *et al.* proposed an electro-magnetic attack against a bitsliced GPU AES implementation [13] (called Gao's attack hereinafter). Their approaches are essentially combinational attacks that implicitly use the MIS feature of GPU.

Although these attacks are effective, it is unfair to compare their performance, because these attacks were performed in different settings. All for that, each of them has its own features, which has given us inspiration. Hence, it is necessary to further discuss on their proposals, as well as the relation with our proposal.

a) Attacker's Capability: Generally speaking, the seriousness of an attack is closely related to the assumptions on attacker's capabilities like measuring some data and/or setting some parameters. In Luo's attack [6], [7], the attacker is able to choose maps from the plaintexts to the multiple threads of GPU and the number of threads to launch the GPU kernel. The attack is called *chosen-thread attack* (CTA, explained in II-D) in the paper. In Jiang's attacks [8], [9], [10], [11] and Gao's attack [13], the attackers also need be able to perform a CTA, because the attacker primarily used 32 threads (one GPU warp) to launch the GPU kernel, and then measured the timing or EM emissions of execution. In Fomin's timing attack [12], the attacker not only needs the capability of CTA, but also needs another capability, that is, choosing the plaintexts (terminologically, *chosen-plaintext attack*). Although Fomin's attack assumed a more powerful attacker than the others, all of them require attackers set some parameters for encryption. For attackers who are only able to obtain random ciphertexts by eavesdropping, they have not performed any successful attacks so far. In our study, the attacker is also assumed to be one who is able to perform CTAs.

b) GPU-specific Approach: Although all the attacks targeted GPUs, the approaches they took, to some extent are not necessarily GPU-specific. In other words, some of them also work on CPUs. For example, Jiang's attacks [8], [9], [10], [11] and Fomin's attack [12] are GPU-specific, while Luo's attack [6], [7] and Gao's attack [13] are not, because if there were not MIS or MBC feature, Jiang's attacks and Fomin's attack would not be effective. In our study, we propose GPU-specific approaches, which also depends on MIS feature of GPU, but make better use of the feature based on EM traces instead of timing traces used in previous work.

c) Multi-Thread Leakage: In multi-thread systems, each thread may generate its individual leakage, say power leakage, EM leakage. Luo's attack [6], [7] uses a naive multi-thread mixed leakage model, which introduces more noise. In the meantime, a variant (SSM in III-A) of the leakage model is one of our approaches, and it is primarily for comparison in the paper. In addition, Gao's attack [13] is exactly based on the variant model, as well as combinational analysis techniques. In our study, the variant is treated as a baseline model, and we propose more powerful models than the baseline model.

In light of this, our study aims to give a deeper insight into multi-thread leakage based on the MIS feature of GPU, and propose more efficient GPU-specific attacks under the common assumption, namely CTA (II-D). It should be noted that in previous studies [14], [15], they also presented multi-thread mixed leakage models, though they did not call them in this way. However, their studies are too specific and do not provide much details on their approaches. In this paper, we present more generic models and optimizations, together with detailed explanation of rationales behind the approaches. Besides, we stress on multi-threading, because EM leakage of multi-threaded execution is always more sophisticated, which makes it challenging to better use the leakage in order to perform efficient side-channel attacks.

A. Contributions

In this paper, we propose multi-thread mixed leakage models and optimizations, which are used to mount efficient attacks. The contributions are summarized as follows:

- We propose multi-thread mixed leakage models, namely *Simple Summation Model* (SSM) and *Partial Summation Model* (PSM), which apply to side-channel attacks against general cryptographic implementations on general SIMT systems. At the same time, we also suggest roadmaps for further improvements.
- We propose two special PSMs, namely *Cacheline Summation Model* (CSM), which is essentially instantiating a key component called *threads partitioner*, and *Cacheline Count Model* (CCM), which is actually *counting* the number of the partitioned parts. Both models are customized for typical table-based cryptographic implementations on specific SIMT systems, precisely CUDA-enabled GPUs.
- We employ combination-based approaches for optimization. The approaches are respectively named DCA^{+}, $\mathrm{DCA}^{\times}$ and $\mathrm{DCA}^{\max}$ (*Decision Combinational Attack*), which essentially define aggregate functions that combines distinguishers of, say CSMs and/or CCMs.

B. Organization

The rest of this paper is organized as follows. In Section II, we give a brief introduction to the architecture of CUDA-enabled GPUs (II-A), side-channel cryptanalysis (II-B), table-based cryptographic implementations (II-C), as well as definitions, notations (II-D) and abbreviations (II-E) involved in the paper. In Section III, we propose our leakage models and optimizations. Specifically, SSM and PSM are proposed in III-A, CSM is proposed in III-B, CCM is proposed in III-C, and the optimizations are proposed in III-D. In Section IV, we instantiate our methods with a GPU AES implementation. In Section V, we evaluate the models and optimizations on EM traces measured from the running GPU AES implementation, and we also evaluate the models on EM traces mixed with Gaussian noise to simulate the evaluation on EM traces from other GPU devices. In Section VI, we make a discussion on issues about our proposal. Finally, conclusions are given in Section VII.

II. Preliminary

A. CUDA-enabled GPU

CUDA is a general purpose parallel computing framework and programming model developed by NVIDIA for its GPUs. In a physical view, the CUDA-enabled GPU is composed of M× Streaming Multiprocessors (SM) and a global memory. Each SM has N× Scalar Processor (SP), a shared memory, several 32-bits registers, and a shared instruction unit (**Fig.**1). In an abstract view, CUDA defines the threading model, calling conventions and memory hierarchy for programmers.

Warps are the basic unit of execution in an SM. When you launch a grid of thread blocks, the thread blocks in the grid are distributed among SMs. Once a thread block is scheduled to an SM, threads in the thread block are further partitioned into warps. A warp consists of 32 consecutive threads and all threads in a warp are executed in *single instruction multiple thread* (SIMT) fashion; that is, all threads execute the same instruction, and each thread carries out that operation on its own private data.

Global memory resides in device memory and is accessible via 32-byte, 64-byte, or 128-byte memory transactions. These memory transactions must be naturally aligned; that is, the first address must be a multiple of 32 bytes, 64 bytes, or 128 bytes. When a warp performs a memory `load/store`, the number of transactions required to satisfy that request typically depends on the following two factors. There is one L1 cache per-SM and one L2 cache shared by all SMs. Both L1 and L2 caches are used to store data in local and global memory, including register spills. On Fermi (`compute_capability` = `2.x`) GPUs, CUDA allows you to configure whether `read`-s are cached in both L1 and L2, or only L2. All accesses to global memory go through the L2 cache. Many accesses also pass through the L1 cache, depending on the type of access. If both L1 and L2 caches are used, a memory access is serviced by a 128-byte memory transaction. If only the L2 cache is used, a memory access is serviced by a 32-byte memory transaction. On architectures that allow the L1 cache to be used for global memory caching, the L1 cache can be explicitly enabled or disabled at compiling time.

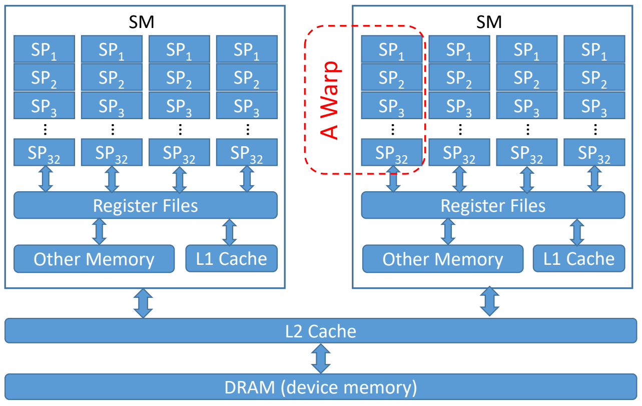

Fig. 1: Microarchitecture of CUDA-enabled GPU.

B. Side-Channel Cryptanalysis

A *side-channel attack* (SCA) is any attack based on information gained from the implementation of a computer system, rather than weaknesses in the implemented algorithm itself. If the target of an SCA is the implementation of a cryptographic algorithm, then the SCA is called *side-channel cryptanalysis*.

For a side channel (e.g. EM) in practice, if the channel is assumed linear, its leakage $l(x,k)$ is usually formalized as:

$$l(x,k) = \alpha \cdot f(Z(x,k)) + \beta + \eta, \tag{1}$$

where x is part of the known plaintext or ciphertext, k is a sub-key in a finite set $\mathcal{K}$, $Z(x,k)$ is a target intermediate value depending on x and k, α and β are some constants, η is the noise usually considered to be zero-mean, and $f(\cdot)$ is a data-dependent leakage function of the channel.

For multiple, say n, inputs with the same k, the leakage can be rewritten as:

$$\boldsymbol{l}(\boldsymbol{x},k) = \alpha \cdot \boldsymbol{f}(Z(\boldsymbol{x},k)) + \beta + \boldsymbol{\eta}, \tag{2}$$

where $\boldsymbol{x} = [x_1, x_2, ..., x_n]^T$, $\boldsymbol{\eta} = [\eta_1, \eta_2, ..., \eta_n]^T$, and $\boldsymbol{f}(Z(\boldsymbol{x},k)) := [f(Z(x_1,k)), f(Z(x_2,k)), ..., f(Z(x_n,k))]^T$.

C. Table-based Cryptographic Implementations

For some cryptographic algorithms, some building blocks of them can be implemented by looking up tables. The tables are usually loaded into the memory when executing on a processor and accessed by *table look-up* (TLU) operations. Formally, a *look-up table* (LUT) is defined as an injection:

$$T : \{0,1\}^{\phi} \to \{0,1\}^{\psi}, y = T[x], \tag{3}$$

where $0 < \phi \leq \psi$. So the size of the LUT is $\psi \cdot 2^{\phi}$ bits. Table-based cryptographic implementations are usually vulnerable to cache attacks, which were studied in papers like [16], [17].

D. Definitions & Notations

Definition 1 (Chosen-Thread Attack): For a side-channel attack against a (multi-thread) task-level parallel cryptographic implementation, if an attacker is able to choose some threads to encrypt the same but random plaintexts and, at the same time, collect the corresponding ciphertexts and side-channel leakage, then the attack is called *Chosen-Thread Attack* (CTA). Note that attackers of CTA cannot choose specific plaintexts for any of the threads, which is much different from *chosen-plaintext attack*.

In this paper, "$\cdot$" denotes *all* as a subscript, superscript or in [ ]. For example, if $\mathcal{M}$ is a 2×4 matrix, then $\mathcal{M}[\cdot, 2]$ denotes a column vector composed of the second column of $\mathcal{M}$. "$\cdot$" also denotes *something* certain but omitted when it appears in ( ) and "$\cdots$" denotes *something* uncertain thus being omitted. Note that "uncertain" does not mean "unknown", but "known" and omitted on purpose due to hardness to represent.

- $\mathbb{E}(X)$: computes the *mathematical expectation* of X.
- $\mathrm{H}(X)$: computes the *Hamming weight* of a bit string X.
- $\rho(\boldsymbol{X}, \boldsymbol{Y})$: computes the *Pearson correlation coefficient* (PCC) between every column of $\boldsymbol{X}$ and every column of $\boldsymbol{Y}$, where $\boldsymbol{X}$ and $\boldsymbol{Y}$ have the same number of rows, so $\rho(\boldsymbol{X}, \boldsymbol{Y})$ is of the same dimension with $\boldsymbol{X}^T\boldsymbol{Y}$.
- $\boldsymbol{L}^{(\delta,\sigma^2)}$: a set of traces measured in δ-group mode (IV-B), and then mixed with Gaussian noise of 0-mean and σ^2-variance. Specially, $\boldsymbol{L}^{(\delta)} := \boldsymbol{L}^{(\delta,0)}$, In addition, it is noted that the expressions "an attack in δ-group mode" and "an attack on $\boldsymbol{L}^{(\delta)}$" are always equivalent.

E. Abbreviations & Index

The abbreviations involved in the paper are as follows:

AES	*Advanced Encryption Standard*	
CCM*	*Cacheline Count Model*	III-C
CEMA	*Correlation Electro-Magnetic Analysis*	
CSM*	*Cacheline Summation Model*	III-B
CTA*	*Chosen-Thread Attack*	II-D
CTP*	*Cacheline-based Thread Partitioner*	III-B
CUDA	*Compute Unified Device Architecture*	
DCA*	*Decision Combinational Attack*	III-D
DEMA	*Differential Electro-Magnetic Analysis*	
DoM	*Difference-of-Means*	
EAM*	*Ends Alignment Model*	IV-D
GPU	*Graphics Processing Unit*	
GSR*	*Global Success Rate*	V-A
LCSM*	*Left Cacheline Summation Model*	IV-C
LUT	*Look-Up Table*	
MAM*	*Middle Alignment Model*	IV-D
MIS*	*Memory Issue Serialization*	I[8]
PSM*	*Partial Summation Model*	III-A
RCSM*	*Right Cacheline Summation Model*	IV-C
SIMT	*Single Instruction, Multiple Threads*	
SNR	*Signal-to-Noise Ratio*	
SSM*	*Simple Summation Model*	III-A
STG*	*Synchronous Thread Group*	III-A
STP*	*Synchronous Threads Partition*	III-A
TLU	*Table Look-Up*	

where "*" means the abbreviation is not commonly used, and will be explained below in the paper.

III. Our Methods

Our research targets the side-channel attacks against *single instruction multiple thread* (SIMT) systems, especially CUDA-enabled GPUs. As one of SIMT systems, GPUs feature their specific and special multicore architecture, which makes EM leakage from them more complicated in practice, so the most important issue in our research is to construct accurate multi-thread mixed leakage models. In light of this, we propose multi-thread mixed leakage models, namely SSM, PSM, CSM and CCM, and optimizations, namely DCA^{+}, DCA$^{\times}$ and DCA$^{\max}$, that are based on combination of distinguishers. Of the models, SSM and PSM are generic ones for general SIMT systems, while CSM and CCM are specific to table-based cryptographic implementations on CUDA-enabled GPUs. In addition, CSM and CCM are also involved in the optimizations with DCA^{+}, DCA$^{\times}$ and DCA$^{\max}$.

It should be noted that our studies are only for systems of SIMT execution fashion, which means the parallelized units execute an identical instruction sequence. The identical instruction sequence does not necessarily mean that each instruction in the sequence is executed synchronously in all threads, but dummy instructions are allowed between any two instructions of the instruction sequence in any threads. **Fig**.2 presents some examples, i.e. (A) and (B).

A. Simple Summation Model and Partial Summation Model

A lot of literature has studied the modelling of side-channel leakages in single-thread scenarios [18], [19] but rarely has been devoted to the studies in multi-thread scenarios. In the latter, multiple threads run simultaneously, so the leakages of them are mixed together, hence called *mixed leakage* in the paper. The simplest way to model mixed leakage of multiple threads is to add up their respective leakages. The intuitive model is named *simple summation model* (SSM) in this paper and formalized as:

$$F(\mathcal{I},k) := \sum_{j=1}^{N} f(Z(x^{\langle j\rangle},k)) \tag{4}$$

where N denotes the number of threads executing in parallel, $\mathcal{I}$ is a set of thread identities, $\mathcal{I}=[1,N]\cap\mathbb{Z}$, $f(\cdot)$, $Z(\cdot,\cdot)$, k and $x^{\cdot}$ are defined the same as in **Eq.**1, $x^{\langle j\rangle}$ denotes the x in the j-th thread, and F denotes the assumed mixed leakage. The equation also indicates the leakage of an operation on $z^{\langle j\rangle}$ synchronize at a moment, where $z^{\langle j\rangle}=Z(x^{\langle j\rangle},k)$ for each $j\in\mathcal{I}$.

Intuitively, the model is more accurate when the N threads are synchronous on the target operation; otherwise, the model seems somewhat unreasonable. In fact, if the N threads do not synchronize on the target operation, the asynchronization can be regarded as introducing more noise. Suppose the N threads, say th_1,th_2,...,th_N, are synchronous on the target operation in several subsets, named *synchronous thread group* (STG), say $\mathcal{I}_1$, $\mathcal{I}_2$,..., $\mathcal{I}_H$, at the moments t_1, t_2,...,t_H, respectively, where

Algorithm 1 SSM-based CEMA (SSM-CEMA)

```
Require: $\boldsymbol{L}_{W\times M}$, $[\boldsymbol{x}^{\langle 1\rangle},\boldsymbol{x}^{\langle 2\rangle},...,\boldsymbol{x}^{\langle N\rangle}]$, $\mathcal{K}$.
Ensure: $\hat{k}$.
1: $m \leftarrow 1$
2: for $k\in\mathcal{K}$ do
3:     $\boldsymbol{F}_{\cdot,m} \leftarrow \sum_{j=1}^{N} f(Z(\boldsymbol{x}^{\langle j\rangle},k))$
4:     $m \leftarrow m+1$
5: $\hat{k} \leftarrow argmax_k \max|\rho(\boldsymbol{L}_{EM},\boldsymbol{F})|$
6: return $\hat{k}$
```

the threads in each subset $\mathcal{I}_i$ $(i\in\{1,2,...,H\})$ are assumed synchronous for the target operation, so **Eq**.4 can be rewritten as:

$$\begin{aligned}F(\mathcal{I},k) &= \sum_{j\in\mathcal{I}} f(Z(x^{\langle j\rangle},k)) = \sum_{i=1}^{H}\sum_{j\in\mathcal{I}_i} f(Z(x^{\langle j\rangle},k))\\ &= \sum_{i=1}^{H} F(\mathcal{I}_i,k) = F(\mathcal{I}_1,k)+F(\mathcal{I}_2,k)+...+F(\mathcal{I}_H,k).\end{aligned}$$

where $\{\mathcal{I}_1,\mathcal{I}_2,...,\mathcal{I}_H\}$ is called a *synchronous threads partition* (STP) of the $\mathcal{I}$; that is, $\bigcap_{i=1}^{H}\mathcal{I}_i=\emptyset$ and $\bigcup_{i=1}^{H}\mathcal{I}_i=\mathcal{I}$. Suppose the target operation in the threads $\mathcal{I}_j$ happens at t_j. Then, the real leakage of the N threads at t_1 will be:

$$\begin{aligned}L_{t_1} &= \sum_{i\in\mathcal{I}} L_{t_1}(i)+\beta+\eta = \sum_{i\in\mathcal{I}_1} L_{t_1}(i)+\sum_{i\in\mathcal{I}-\mathcal{I}_1} L_{t_1}(i)+\beta+\eta\\ &= \alpha\cdot F(\mathcal{I}_1,k)+\sum_{i\in\mathcal{I}-\mathcal{I}_1} L_{t_1}(i)+\beta+\eta\\ &= \alpha\cdot F(\mathcal{I},k)-\alpha\cdot\sum_{i=2}^{H}F(\mathcal{I}_i,k)+\sum_{i\in\mathcal{I}-\mathcal{I}_1} L_{t_1}(i)+\beta+\eta\\ &= \alpha\cdot F(\mathcal{I},k)-\sum_{j=2}^{H}\sum_{i\in\mathcal{I}_j} L_{t_j}(i)+\sum_{i\in\mathcal{I}-\mathcal{I}_1} L_{t_1}(i)+\beta+\eta\\ &= \alpha\cdot F(\mathcal{I},k)-\gamma_1+\chi_1+\beta+\eta\end{aligned} \tag{5}$$

where $L_{t_1}(i)$ denotes the real leakage of the thread th_i at t_1, $\gamma_1 := \sum_{j=2}^{H}\sum_{i\in\mathcal{I}_j} L_{t_j}(i)$, $\chi_1 := \sum_{i\in\mathcal{I}-\mathcal{I}_1} L_{t_1}(i)$, and χ_1, γ_1 are regarded as noise. The equation suggests that the additional noise γ_1, χ_1 are introduced if the mixed leakage of asynchronous threads on the target operation are modeled by SSM. To deeply understand the transformation (**Eq**.5), we give an example as follows:

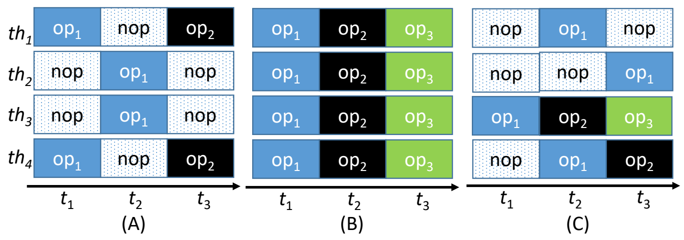

Fig. 2: Synchronization of threads at operations.

In **Fig**.2, it shows three typical cases, namely (A), (B) and (C), of three operations, namely op_1, op_2 and op_3, in four threads, namely th_1, th_2, th_3 and th_4, where op_1 is assumed to be the target operation. In (B), the threads are synchronous for op_1, so an SSM suffices to model the mixed leakage of op_1, and $\chi_1 = \gamma_1 = 0$ in this case. However, it is not like this for (A) or (C), because for op_1, it is not synchronous in all the threads. (A) is more special than (C); i.e., different operations never execute in the same time slot. That means if we model the mixed leakage of op_1 at t_1 (or t_2) with an SSM, i.e. $F(\{1,2,3,4\},k)$ in (A), $F(\{2,3\},k)$ (or $F(\{1,4\},k)$) is regarded as the additional noise ($\gamma_1 \neq 0$, $\chi_1 = 0$), while if we model with an SSM at t_2, i.e. $F(\{1,2,3,4\},k)$ in (C), the additional noise includes not only $F(\{2,3\},k)$ but also $L_{t_2}(3)$ (the leakage of op_2 in th_3 at t_2). In this case, $\gamma_1 = \alpha \cdot F(\{2,3\},k) \neq 0$ and $\chi_1 = L_{t_2}(3) \neq 0$. χ_1 is often not computable for attackers if only one target operation (op_1) is considered, so it cannot be removed, while γ_1 can be removed because the STP of the threads has been known from (C); that is, $\{\{1,4\},\{2\},\{3\}\}$. Fortunately, the case (C) does not happen in SIMT systems, so the noise χ_1 is not considered in our study.

Formally, we define a more generalized model called *partial summation model* (PSM):

$$F(\mathcal{I}',k) := \sum_{j\in\mathcal{I}'} f(Z(x^{\langle j\rangle},k)), \tag{6}$$

where $\mathcal{I}' \subseteq \mathcal{I}$ and the threads within the $\mathcal{I}'$ are synchronous at the target operation. By the definition, SSM is just a special case of PSM and they are equivalent when $\mathcal{I}' = \mathcal{I} = [1,N] \cap \mathbb{Z}$. SSM corresponds to the special STP, $\{\{1,2,3,...,N\}\}$, which is called the *trivial STP* of a PSM. Suppose $L = \alpha \cdot F(\mathcal{I},k) - \gamma + \chi + \beta + \eta$, then if an STP of the threads for an target operation is known, the γ can be removed. Hence, the first step of building a PSM is to find an STP of an target operation. Once an STP is ready, it comes to the second step; that is, selecting an STG from the STP. Although it seems that an STG of larger size will remove more noise from $F(\mathcal{I},k)$, it is not always true, because the selection of STG also strongly affects the trace alignment, thus reducing the effectiveness of distinguishers. Once an STG is chosen, a PSM is instantiated, and then a PSM-based attack can be mounted with appropriate distinguishers.

We present the SSM- and PSM-based attacks with *Pearson correlation coefficient* (PCC), which is one of the most commonly used distinguishers, in **Algo**.1 and **Algo**.2, respectively. In **Algo**.2, STP(...) denotes extracting an STP, and STG(...) means selecting an STG from the STP.

Algorithm 2 PSM-based CEMA (PSM-CEMA)

```
Require: EM traces: $\boldsymbol{L}_{W\times M}$, $[\boldsymbol{x}^{\langle 1\rangle}, \boldsymbol{x}^{\langle 2\rangle}, ..., \boldsymbol{x}^{\langle N\rangle}]$, $\mathcal{K}$.
Ensure: $\hat{k}$.
1: for $i \leftarrow 1$ to $W$ do
2:     $\mathcal{I} \leftarrow$ STG(STP(...)) ▷ partitioning and then selecting
3:     $m \leftarrow 1$
4:     for $k$ in $\mathcal{K}$ do
5:         $\boldsymbol{F}_{i,m} \leftarrow \sum_{j\in\mathcal{I}} f(Z(x_i^{\langle j\rangle},k))$
6:         $m \leftarrow m+1$
7: $\hat{k} \leftarrow argmax_k \max|\rho(\boldsymbol{L},\boldsymbol{F})|$
8: return $\hat{k}$
```

B. Cacheline Summation Model

The PSM proposed above is a generic model suited to any SIMT systems, and it might be more efficient than SSM if an non-trivial STP for the target operation is available. To find a non-trivial STP (if exists), constraints must be imposed on the target SIMT systems and cryptographic implementation. Specifically, we focus on:

1) the SIMT execution in a thread warp of CUDA-enabled GPUs, and
2) table-based GPU cryptographic implementations, whose tables are stored in the global memory of GPUs.

The constraints are reasonable in practice, because warps are the basic unit of execution in CUDA-enabled GPUs, and the global memory of a GPU is usually large enough to store tables of any reasonable cryptographic implementations.

As known from the architecture of CUDA-enabled GPUs, a warp is usually composed of 32 threads, say $th_1, th_2, ..., th_{32}$, and the 32 threads execute in an SIMT fashion. When the 32 threads access a table stored in the global memory, 32 requests will be coalesced to reduce the total number of requests. Finally, the 32 original requests are serviced by loading several cache lines (usually 64/128/256 bytes as an unit), which is the basic unit of data loading from L1 cache to registers in a CUDA-enabled GPU. Inspired by this, we propose a *cacheline-based threads partitioner* (CTP); that is, a TLU operation in threads whose requests are coalesced into the same cache line loading operation is supposed to be synchronous at the operation in the threads. Obviously, the CTP implies a non-trivial STP.

Algorithm 3 Cacheline-based Threads Partitioner (CTP)

```
Require: $[y^{\langle 1\rangle}, y^{\langle 2\rangle}, ..., y^{\langle N\rangle}]$, $L$ (bytes), $\psi$.
Ensure: $\{\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3, ...\}$.
1: $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_N\}$
2: $S \leftarrow L/(\psi/8)$
3: for $j \leftarrow 1$ to $N$ do
4:     $i \leftarrow \lfloor y^{\langle j\rangle}/S \rfloor + 1$
5:     $\mathcal{G}_i \Leftarrow j$                    ▷ put the $j$ into the set $\mathcal{G}_i$
6: $j \leftarrow 1$
7: for $i \leftarrow 1$ to $N$ do
8:     if $\mathcal{G}_i \neq \emptyset$ then
9:         $\mathcal{I}_j \leftarrow \mathcal{G}_i$
10:        $j \leftarrow j+1$
11: $H \leftarrow j-1$
12: return $\{\mathcal{I}_1, \mathcal{I}_2, ..., \mathcal{I}_H\}$
```

Suppose that the table-based cryptographic implementation refer to an LUT, say $T[\cdot]$, of ϕ-bit input and ψ-bit output, and the size of cache line in target GPU be L bytes. Then the $T[\cdot]$ takes $\frac{\psi\cdot 2^{\phi}}{8\cdot L}$ cache lines. As the time spent on one cache line loading operation is constant, the leakages of the operation on the same cache line are synchronous. In this way, the non-trivial threads partition, say $\{\mathcal{I}_i, \mathcal{I}_2, ..., \mathcal{I}_H\}$, by CTP is as follows:

$$\begin{aligned}\mathcal{I}_1 &= \{j \mid \lfloor y^{\langle j\rangle}/(L/(\psi/8))\rfloor = 0, j \in [1,32]\}\\ \mathcal{I}_2 &= \{j \mid \lfloor y^{\langle j\rangle}/(L/(\psi/8))\rfloor = 1, j \in [1,32]\}\\ &\dots\\ \mathcal{I}_H &= \{j \mid \lfloor y^{\langle j\rangle}/(L/(\psi/8))\rfloor = H-1, j \in [1,32]\}\end{aligned} \tag{7}$$

where $y^{\langle j\rangle} = Z(x^{\langle j\rangle}, k)$ and $H \in [1, \min\{\frac{\psi\cdot 2^{\phi}}{8\cdot L}, 32\}]$. **Algo**.3 shows details of the CTP.

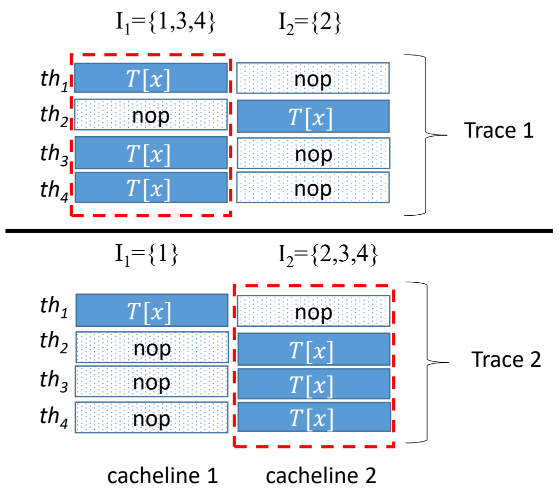

Fig. 3: Trace misalignment on TLU, caused by inappropriate STG selections.

The CTP naturally implies specific PSMs. The H PSMs are named *cacheline summation model* (CSM), which literally means summing up the ones within the same cache line, and formalized as H independent models:

$$F(\mathcal{I}_i, k) := \sum_{j\in\mathcal{I}_i} f(Z(x^{\langle j\rangle}, k)), \quad \text{for each } i \in [1, H] \tag{8}$$

It should be noted that it is not trivial to choose a specific $\mathcal{I}'$ from $\{\mathcal{I}_1, \mathcal{I}_2, , ..., \mathcal{I}_H\}$. It is reasonable to select the one of the largest size, say $\mathcal{I}_{argmax_i \max_{i\in[1,H]}|\mathcal{I}_i|}$, in order to make the best use of the leakage from multiple threads, but sometimes the selection also causes troubles, such as misalignment of traces (**Fig**.3). In **Fig**.3, $\mathcal{I}_1$ is chosen in *Trace 1*, while $\mathcal{I}_2$ is chosen in *Trace 2*. Their TLU operations are misaligned just as the dotted rectangular boxes show. Therefore, the selection of the $\mathcal{I}'$ from $\{\mathcal{I}_1, \mathcal{I}_2, ..., \mathcal{I}_H\}$ varies from case to case. It will be further discussed in IV.

C. Cacheline Count Model

As explained before, the threads are grouped with respect to their synchronization on TLU operations. Suppose the N threads be partitioned into h groups, say $\mathcal{I}_1, \mathcal{I}_2, ..., \mathcal{I}_h$, and $y^{\langle j\rangle} = f(z^{\langle j\rangle})$, $z^{\langle j\rangle} = Z(x^{\langle j\rangle}, k)$ for each $j \in [1, N]$. Then we compute the mathematical expectation of real leakage of PSM when selecting an STG, say $\mathcal{I}_1$:

$$\begin{aligned}\mathbb{E}[L] &= \mathbb{E}\left[\alpha\cdot\sum_{j\in I} f(Z(X^{\langle j\rangle}, k) + \beta + \Omega\right]\\ &= \alpha\cdot\mathbb{E}\left[\sum_{j\in I} f(Z(X^{\langle j\rangle}, k)\right] + \mathbb{E}[\beta] + \mathbb{E}[\Omega]\end{aligned} \tag{9}$$

where α and k are treated as constants, and $X^{\langle j\rangle}$, Ω, I are random variables corresponding to $x^{\langle j\rangle}$, η and $\mathcal{I}_1$, respectively. Since $X^{\langle j\rangle}$-s ($j \in I$) are pairwise independent, and β is also a constant, which means $\mathbb{E}[\beta] = 0$, then we have:

$$\begin{aligned}\mathbb{E}[L] &= \alpha\cdot\sum_{j\in I}\mathbb{E}\left[f(Z(X^{\langle j\rangle}, k)\right] + \mathbb{E}[\Omega]\\ &= \alpha\cdot\sum_{j\in I}\left[\sum_{i=1}^{|\mathcal{X}|}\left(f(Z(\varpi_i, k))\cdot\frac{1}{|\mathcal{X}|}\right)\right] + \mathbb{E}[\Omega]\\ &= \alpha\cdot\mathbb{E}[|I|]\cdot\frac{1}{|\mathcal{X}|}\cdot\sum_{\varpi\in\mathcal{X}} f(Z(\varpi, k)) + \mathbb{E}[\Omega].\end{aligned} \tag{10}$$

where $\mathcal{X} := \{\varpi_1, \varpi_2, \varpi_3, ...\}$ denotes the domain of $X^{\langle j\rangle}$ for any $j \in I$. For any I, the mathematical expectation of $|I|$ is:

$$\mathbb{E}[|I|] = \sum_{n=1}^{N-h+1}(n\cdot\Pr[|I| = n]) = \frac{N}{h} \tag{11}$$

so the mathematical expectation of its real leakage is:

$$\mathbb{E}[L] = \alpha\cdot\frac{N}{h}\cdot\frac{1}{|\mathcal{X}|}\cdot\sum_{\varpi\in\mathcal{X}} f(Z(\varpi, k)) + \mathbb{E}[\Omega] \tag{12}$$

where N denotes the number of threads in a warp. In a specific setting, α, N, $\mathcal{X}$ and $\mathbb{E}[\Omega]$ are constants, so $\mathbb{E}[L]$ only depends on the number of cache lines, namely h. Let us reconsider the CTP proposed in III-B. It is essentially $h = |\text{CTP}(\cdot)|$. Our multi-thread mixed leakage model is named *cacheline count model* (CCM), which literally means counting the number of STGs, and formalized as:

$$F(k) = \frac{N}{h} = \frac{N}{|\text{CTP}([y^{\langle 1\rangle}, y^{\langle 2\rangle}, ..., y^{\langle N\rangle}])|}. \tag{13}$$

where $y^{\langle j\rangle} = Z(x^{\langle j\rangle}, k)$. The real leakage is also rewritten as:

$$\mathbb{E}(L) = \alpha'\cdot F(k) + \beta', \tag{14}$$

where α' and β' are some constants and easily computed with **Eq**.12. The exact values are actually not important at all.

It should be noted that $F(k)$ just models the mathematical expectation of the real leakage L instead of L itself, so it is reasonable when averaging the measured leakage L with a large number of traces. In other words, with limited traces the model may be ineffective at all, so the distinguisher with PCC (also CEMA) does not work any more. We employ the distinguisher with *difference of means* (DoM). Theoretically, $h \in [1, H]$, $H := \min\{\frac{\psi\cdot 2^{\phi}}{8\cdot L}, 32\}$, where ϕ, ψ and L are defined the same as in **Eq**.7, so there are totally $\binom{H}{2} = \frac{H!}{2!\cdot(H-2)!} = (H-1)\cdot H/2$ choices on which DoMs of corresponding traces are computed. In the sense, CCM is also formalized as:

$$F(k) = \begin{cases} F_1, & \textbf{\textit{if}} \quad h = H_1\\ F_2, & \textbf{\textit{if}} \quad h = H_2\end{cases} \tag{15}$$

where $H_1, H_2 \in [1, H]$ and $H_1 \neq H_2$, F_1 and F_2 are random variables satisfying:

$$\mathbb{E}(F_1) = \frac{N}{H_1} \quad and \quad \mathbb{E}(F_2) = \frac{N}{H_2} \tag{16}$$

so we have the difference of them is non-zero:

$$|\mathbb{E}(F_1) - \mathbb{E}(F_2)| = N\cdot\left|\frac{1}{H_1} - \frac{1}{H_2}\right| \neq 0 \tag{17}$$

Then, the real leakage, say L_1 and L_2, should satisfy:

$$\begin{aligned}|\mathbb{E}(L_1)-\mathbb{E}(L_2)| &= |\mathbb{E}(\alpha'\cdot F_1+\beta')-\mathbb{E}(\alpha'\cdot F_2+\beta')| \\ &= |\alpha'|\cdot|\mathbb{E}(F_1)-\mathbb{E}(F_2)| \\ &= |\alpha'|\cdot N\cdot\left|\frac{H_1-H_2}{H_1\cdot H_2}\right|\neq 0.\end{aligned}$$

Obviously, $|\mathbb{E}(L_1)-\mathbb{E}(L_2)|$ is maximized when $H_1=1$ and $H_2=H$, or vice versa. However, the choice of H_1 and H_2 also affects the alignment of traces for target operations in practice, so $H_1=1$ or $H_2=H$ is not necessarily the best choice. Besides, with limited traces, for some h-s in $[1,H]$, they may never happen with random plaintexts encryption due to different probabilities of $h=1,2,3,...,H$. For example, $\Pr[h=1]=\frac{1}{2^{31}}\approx 0$ when $H=2$ and $N=32$, so there is theoretically only one trace in which $h=1$, with up to 2^{31} measured traces. This also tells that there must be wasted traces unless $H=2$, and the number of averaged traces, which also affects performance, varies with different selection of H_1 and H_2. In a word, the selection of H_1 and H_2 is not a trivial issue when H is larger than 2.

D. Combination-based Optimizations

In III-B and III-C, special PSMs, namely CSM and CCM, are investigated. Although they are much efficient, which will be shown afterwards, there is room for further improvement. In this subsection, we study optimizations based on combination techniques, which have been investigated in many papers, e.g. [20], [21], [22]. With multiple models, the simplest way to combine them is to build a combined distinguisher based on their respective distinguishers. In some papers, it is called *decision combinational analysis* (DCA) [13]. The procedure of a DCA is shown in **Fig**.4. For each $i\in[1,S]$, Δ_i is generated with M_i via a distinguisher, and their distinguishers can be the same or not. In our studies, the models can be CSM or CCM; that is, $M_1,M_2,...,M_S\in\{\text{CSM},\text{CCM}\}$, and their distinguishers are based on PCC and DoM, respectively.

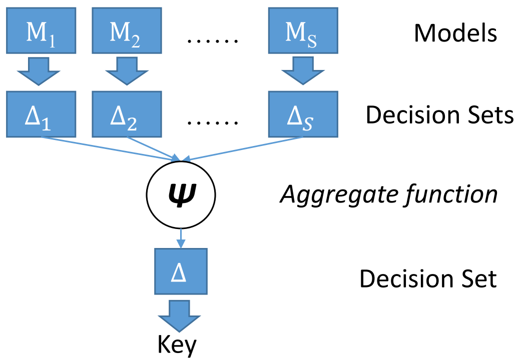

Fig. 4: Overall procedures of a decision-level combinational analysis.

A DCA is essentially implementing an *aggregate function* [21], say Ψ, that returns a single vector/matrix based on multiple vectors/matrices. The Ψ is usually assigned simple functions like Sum$(\cdot)$, Prod$(\cdot)$, Max$(\cdot)$. With the functions, the Δ-s are computed as follows:

$$\begin{aligned}\Delta^+ &:= \text{Sum}(\Delta_1,\Delta_2,...,\Delta_H)=\sum_{i=1}^{H}\Delta_i \\ \Delta^\times &:= \text{Prod}(\Delta_1,\Delta_2,...,\Delta_H)=\prod_{i=1}^{H}\Delta_i \\ \Delta^{\max} &:= \text{Max}(\Delta_1,\Delta_2,...,\Delta_H)=\max_{i=1}^{H}\Delta_i\end{aligned} \tag{18}$$

where Δ^+, $\Delta^\times$ and Δ^{max} denote the Δ computed with corresponding aggregate functions. A DCA with Sum$(\cdot)$, Prod$(\cdot)$ or Max$(\cdot)$ is named DCA$^+$, DCA$^\times$ or DCA$^{\max}$, respectively. In addition, it is noted that the Δ_i-s should be normalized before aggregated with Sum$(\cdot)$, Prod$(\cdot)$ or Max$(\cdot)$, if they are of different metrics. Different distinguishers may generate data sets of different metrics.

In our studies, CSM-CEMA generates a matrix of Pearson coefficient $\mathcal{C}:=[\rho_{ij}]$, and CCM-DEMA generates a matrix of DoM $\mathcal{D}:=[d_{ij}]$, where $i\in[1,|\mathcal{K}|]$, $j\in[1,M]$, and M denotes the number of sampling points in a trace. Suppose we have S models M_1, M_2,..., M_S in total. If they are of the same model (CSM or CCM), then the aggregate function of the DCA is like **Eq**.18; otherwise, a normalization is required before using the aggregate function:

$$\Delta=\Psi(\text{N}(\Psi(\mathcal{C}_1,\mathcal{C}_2,...)),\text{N}(\Psi(\mathcal{D}_1,\mathcal{D}_2,...))) \tag{19}$$

where N$(\cdot)$ denotes a normalization function on a matrix. The normalization function can be row-data-based, column-data-based or all-data-based. Row-data-based (resp. Column-data-based) matrix normalization means each row (resp. column) vector is normalized independently, and all-data-based matrix normalization means all data in the matrix are involved in the normalization.

The name of a DCA follows a format like M$_1$-M$_2$-M$_3$-...-DCA^F, where M$_i$ is a CSM or a CCM, and $F\in\{+,\times,\max\}$. For example, if two CSMs, say CSM$_1$, CSM$_2$, and one CCM are used, and their distinguishers are aggregated with Sum$(\cdot)$, then the DCA is named CSM$_1$-CSM$_2$-CCM-DCA$^+$.

IV. Instantiated with a GPU AES Implementation

In III, we propose multi-thread mixed leakage models that apply to general/specific cases, as well as optimizations. As we do not assume any specific cryptographic implementation, the proposals are somewhat hard to understand. In this section, we instantiate the models and the optimizations with a specific GPU AES implementation to uncover more details when they are used in practical attacks.

A. Target GPU-based AES Implementation

There are lots of AES implementations for general purpose processors or optimized for specific processors. They mainly fall into three categories: S-Box LUT fashion, T-Table (or T-Box) fashion [23], and bitsliced [24] implementations, of which S-Box LUT and T-Table fashion are table-based implementations. As the CSM and CCM proposed are specific to table-based implementations, we port an S-Box LUT based AES implementation from an open source crypto library [25] into our target GPU (detailed in V-A). We choose an S-Box fashion AES instead of a T-Table one because S-Box fashion is more basic and enough to show the vulnerabilities of GPUs under EM attacks.

An AES-128-ECB is implemented on the target GPU, where each thread of the GPU runs a single AES-128 encryption. It also means all threads execute with identical 128-bit secret key. Furthermore, due to a more complicated memory hierarchy in GPUs than that in CPUs, programmers have more control over the memory to optimize their programs. It is well known that the execution footprint of CUDA programs varies among different usage of the memory, so it is necessary for us to provide more details about our implementation. Specifically, we follow the most general procedure of a CUDA program. *At first*, we copy all data including 16-byte plaintext to be encrypted, 176-byte subkeys scheduled on the host, and 256-byte S-box LUT from host memory to global memory. *Then*, we launch the kernel with 32 block threads. We feed 32 block threads with the same plaintexts or partially different plaintexts. They will result in different EM leakages. *Finally*, we copy the encrypted data (ciphertexts) from global memory back to host memory. It is noted that we do not use any shared memory, constant memory or texture memory in our implementation, because they need more technical modifications on the original program.

B. Attacking Modes for Multi-thread Scenarios

For attacking against the target implementation, we consider an attacker with capability of CTA (**Def**.1). We assume such an attacker because the attacker is able to perform attacks in multiple modes, which has great effects on synchronization among threads and alignment among traces, thus challenging the effectiveness of the proposed models. We do need the challenge to make our proposals more convincing.

As defined in **Def**.1, an attacker of CTA is able to choose some threads to encrypt the same plaintexts, so for simplicity we define several modes with respect to the number of threads the attacker chooses. The modes are named *grouped threads mode* in the paper. In grouped threads modes, the 32 threads in a thread warp are divided into groups of equal size, and the threads in the same group are fed with the same plaintext, so there are six modes in total, namely **$\boldsymbol{\delta}$-group mode**, where $\delta \in \{1, 2, 4, 8, 16, 32\}$ denotes the number of groups, and hence the size of each group is $\frac{32}{\delta}$. It should be noted that consecutive threads are always grouped together in the modes.

The six modes fall into three categories named Category I, Category II, and Category III (**TABLE** I), respectively. We define the categories because they are of different features in, say threads synchronization and traces alignment. It is noted that each of the six grouped threads modes actually defines a degree of parallelism in a sense. For example, $\delta = 1$ simulates single thread and $\delta = 4$ simulates four threads executing in parallel.

As known from **Algo**.3, when $N = 32$ (thread warp size) and $L = 128$ (cache line size), for our target implementation, the target TLU in the 32 threads is executed in one or two stages. Specifically, the target TLU in the 32 threads is executed in one stage, called *one-staged* TLU when $|\text{CTP}(\cdot)| = 1$, and in two stages, called *two-staged* TLU when $|\text{CTP}(\cdot)| = 2$. Let us compute probabilities of the occurrences:

Cat.	[1]δ	[2]*Sync?*	[3]*Aligned?*	*Models*	p_1
I	1	**Yes**	**Yes**	SSM (=CSM=CCM)	1
II	2	**No**	**No**	CSM, CCM	$1/2$
	4				$1/2^3$
	8				$1/2^7$
III	16	**No**	**Yes**	CSM	$1/2^{15} \approx 0$
	32				$1/2^{31} \approx 0$

[1] δ-group mode.
[2] Synchronization on TLU among threads.
[3] Alignment on TLU among traces.

TABLE I: A summary of threads synchronization and traces alignment.

$$p_1 := \Pr[|\text{CTP}(\cdot)| = 1] = \frac{1}{2^{\delta-1}}$$

$$p_2 := \Pr[|\text{CTP}(\cdot)| = 2] = 1 - \frac{1}{2^{\delta-1}}$$

Obviously, $p_1 = 1$ when $\delta = 1$, so for mode in Category I all threads are always synchronous on every TLU in the final round of AES. SSM is suited to mode in the category, and it is essentially equivalent to CSM for mode in the category. The same way, $p_1 = \frac{1}{2^{15}}, \frac{1}{2^{31}} \approx 0$ when $\delta = 16, 32$, so for modes in Category III, the threads are almost (so regarded as always) partitioned into two STGs, say $\mathcal{I}_1$, $\mathcal{I}_2$, within which the threads are synchronous on TLU. CSM is suited to modes in the category because $|\text{CTP}(\cdot)| > 1$ in the modes, while CCM is not suited because $|\text{CTP}(\cdot)|$ is (almost) constant ($= 2$) in multiple traces. For the modes in Category I and III, the target TLU is always one-staged (*Cat.* I) or two-staged (*Cat.* III), so the final round of all traces are of the same length (equivalent to the time spent on loading one cache line for one-staged TLU or two cache lines for two-staged TLU) in the time domain. That is to say, traces are always aligned in modes of both categories. However, for modes in Category II, one-staged TLU and two-staged TLU happen in comparable probabilities; that is, $p_1 = \frac{1}{2}, \frac{1}{8}, \frac{1}{128}$ and $p_2 = \frac{1}{2}, \frac{7}{8}, \frac{127}{128}$ when $\delta = 2, 4, 8$, respectively. In the modes, both CSM and CCM are effective, but traces of the final round are not aligned any more due to the difference of time spent on one-staged TLU and two-staged TLU. It seems that the misalignment of traces is unfavourable to attacks, but not necessarily like this in practice, which will be elaborated afterwards.

C. Instantiating the Models

In our studies, we consider the final round of AES encryption. As is known, for a ciphertext byte, say c, it is generated with the following transformation:

$$c \leftarrow \text{SBox}[r] \oplus k,$$

where $\text{SBox}[\cdot]$ denotes an S-Box LUT, r denotes an input byte of the final round, and k denotes a key byte of the final round key. That means, if the *S-Box TLU* is regarded as target operation, for a single thread $Z(\cdot)$ is set as:

$$Z(c, k) := \text{SBox}^{-1}[c \oplus k]. \tag{20}$$

If the single-thread leakage function, namely $f(\cdot)$, is assumed to be *Hamming weight evaluation*, then the leakage model of a single thread is instantiated to:

$$f(Z(c,k)) := \mathrm{H}(\mathrm{SBox}^{-1}[c \oplus k]), \tag{21}$$

where $\mathrm{H}(\cdot)$ denotes the Hamming weight of a bit string. Then, for multi-thread encryption, its SSM (**Eq**.4) is instantiated to:

$$F(\{1,2,...,32\},k) := \sum_{j=1}^{32} \mathrm{H}(\mathrm{SBox}^{-1}[c^{\langle j\rangle} \oplus k]) \tag{22}$$

Accordingly, the real leakage should be:

$$L := \alpha \cdot \sum_{j=1}^{32} \mathrm{H}(\mathrm{SBox}^{-1}[c^{\langle j\rangle} \oplus k]) + \beta + \eta, \tag{23}$$

for some α, β and η, if leakage of 32 threads in a warp are assumed to be synchronous at specified operations. Based on the SSM (**Eq**.21) and the linearity assumption of the channel (**Eq**.23), it is easy to perform an SSM-CEMA (**Algo**.1) to recover the 128-bit secret key of AES.

The SSM-CEMA should be efficient for mode in Category I, because in this mode each operation is synchronously executed in the 32 threads, and thus their leakages are also assumed to be synchronous in the time domain. In addition, SSM is also equivalent to CSM for the mode. However, it will not be like this if the 32 threads are not given the same plaintexts as it does in 1-group mode. In this case, a PSM should be employed, and we try CSM first.

As the 32 threads execute a TLU in a 256-byte S-Box LUT ($\phi = \psi = 8$) stored in the global memory of the GPU and the size of L1 cache line of the target GPU is 128 bytes ($L = 128$), the TLU should be either one-staged or two-staged. If we assume the TLU to be a target operation, then the 32 threads on the TLU are fully synchronous or partially synchronous.

Suppose $r^{\langle 1\rangle}, r^{\langle 2\rangle}, ..., r^{\langle 32\rangle}$ be the r in the 32 threads th_1, th_2 ... th_{32} respectively, then the threads with their IDs (also subscripts) in the following two sets are synchronous:

$$\begin{aligned} G_1 &= \{j | r_j \equiv 1 \pmod{128}\} \\ G_2 &= \{j | r_j \equiv 0 \pmod{128}\} \end{aligned} \tag{24}$$

where $r^{\langle j\rangle} \in [0, 255]$ for each $j \in [1, 32]$.

If $G_1 = \emptyset$ or $G_2 = \emptyset$, then $\mathrm{CTP}(\cdot) = \{\{1, 2, 3, ..., 32\}\}$. That means the 32 threads are synchronous on the TLU, so the PSM is actually an SSM; otherwise, the leakage can be modeled by either of the following two CSMs, namely:

$$F(G_1,k) := \sum_{j\in G_1} \mathrm{H}(\mathrm{SBox}^{-1}[c^{\langle j\rangle} \oplus k]); \tag{25}$$

$$F(G_2,k) := \sum_{j\in G_2} \mathrm{H}(\mathrm{SBox}^{-1}[c^{\langle j\rangle} \oplus k]). \tag{26}$$

That mean that we can use either G_1 or G_2 in modeling mixed leakage at the target TLU. As the TLU happens in G_1 earlier than in G_2, the CSMs with G_1 and G_2 are named *left CSM* (LCSM) (**Eq**.25) and *right CSM* (RCSM) (**Eq**.26), respectively. The models actually use *chronological* STG selectors, selecting an STG (i.e. G_1 or G_2) by their chronological order.

In a broad sense, CSM is suited to any modes including modes in *Category* I, II and III. However, CCM is suited to modes in which both one-staged and two-staged TLU happen in comparative probabilities. For the purpose, we consider modes in *Category* II. The number of STGs (i.e. $|\mathrm{CTP}(\cdot)|$) is $h \in \{1, 2\}$, so the only choice is $H_1 = 1$ and $H_2 = 2$ (or vice versa). Then, the CCM for the target TLU in our target implementation is:

$$F(k) = \begin{cases} \frac{N}{H_1} = 32, & \textbf{\textit{if}} \quad G_1 = \emptyset \ \textit{or} \ G_2 = \emptyset \\ \frac{N}{H_2} = 16, & \textbf{\textit{otherwise}} \end{cases} \tag{27}$$

Since we use a DoM distinguisher, the CCM used is actually:

$$|\mathbb{E}(F_1) - \mathbb{E}(F_2)| = 16 \tag{28}$$

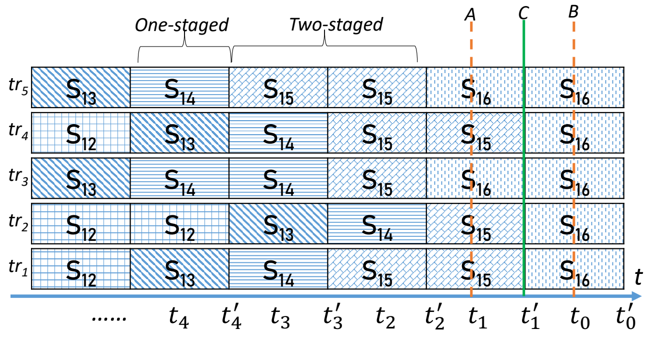

Fig. 5: The layout of trace alignment on S-Box TLU operation.

D. Issues on Misalignment of Traces

With the models (**Eq**.22,25,26,28), attacks can be performed by employing PCC distinguisher or DoM distinguisher. As is known, both distinguishers require aligned traces or, more precisely, traces aligned on target operation. It is indeed true for modes in Category I and III, but not for modes in Category II. Luckily, for modes in Category II, the misalignment of traces follows some rules instead of completely random, so the distinguishers still work.

Our explanation is based on two alignment models: *middle alignment model* (MAM) and *ends alignment model* (EAM). **Fig**,5 shows the layout of cache line access paradigm in five traces. The traces are perfectly *middle-aligned* (MAM) on S_{16} (the $16th$ S-Box TLU) at t_1 when using RCSM, and tr_3, tr_5 are middle-aligned on S_{16} at t_3 when using LCSM, while the traces are perfectly *ends-aligned* (EAM) on S_{16} at t_2 for both LCSM and RCSM. The above explanation is suited to CCM as well. As explained above, of the three positions A, B and C, A is for LCSM+MAM, B is for RCSM+MAM and C is for both LCSM+EAM and RCSM+MAM, while both A and B are for CCM+MAM, and C is for CCM+EAM.

It should be noted that t_3 is a moment at which S_{15} is also middle-aligned for RCSM on tr_1, tr_2 and tr_4. That means at some moments $t \in \{t_1, t_2, t_3, ...\}$ (actually most moments), multiple S-Box TLU operations ($\{S_{16}, S_{15}, S_{14}, ..., S_1\}$) are middle/ends-aligned for LCSM/RCSM/CCM on some traces. That also means, for certain model (CSM or CCM), a certain S-Box TLU operation (S_1, S_2,... or S_{16}) is middle/ends-aligned at ***multiple*** moments on several traces, so the moments at which the most traces are aligned on the operation should be the best choice for corresponding distinguisher to recover secret key byte. Therefore, we compute the probability that the $(16-j)$-th S-Box TLU s_j with LCSM/RCSM/CCM happens at t_{i+j} (MAM) and t'_{i+j} (EAM):

$$\mathcal{P}^{\text{MAM}}_{\text{RCSM}}(s_j, t_{i+j}) := \Pr[\mathfrak{S} = s_j, \mathfrak{T} = t_{i+j}] = \binom{j}{i} p_1^{j-i} p_2^i$$

$$\mathcal{P}^{\text{MAM}}_{\text{LCSM}}(s_j, t_{i+j}) := \Pr[\mathfrak{S} = s_j, \mathfrak{T} = t_{i+j}] = \binom{j+1}{i} p_1^{1+j-i} p_2^i$$

$$\begin{aligned}\mathcal{P}^{\text{EAM}}_{\text{RCSM}}(s_j, t'_{i+j}) &:= \Pr[\mathfrak{S} = s_j, \mathfrak{T} = t'_{i+j}] \\ &= \binom{j}{i-1} p_1^{1+j-i} p_2^{i-1} + \binom{j}{i} p_1^{j-i} p_2^i\end{aligned}$$

$$\begin{aligned}\mathcal{P}^{\text{EAM}}_{\text{LCSM}}(s_j, t'_{i+j}) &:= \Pr[\mathfrak{S} = s_j, \mathfrak{T} = t'_{i+j}] \\ &= \binom{j+1}{i-1} p_1^{2+j-i} p_2^{i-1} + \binom{j+1}{i} p_1^{1+j-i} p_2^i\end{aligned}$$

where $p_2 = 1 - p_1$, $i \in \{0, 1, ..., 31\}$ for t_i, $i \in \{0, 1, ..., 32\}$ for t'_i, $j \in \{0, 1, 2, ..., 15\}$, $\binom{n}{k} := 0$ for $k \notin [0, n] \cap \mathbb{Z}$ and s_j maps to S_{16-j} (**Fig**.5). The same way,

$$\mathcal{P}^{\text{MAM}}_{\text{CCM}}(s_j, t_{i+j}) := \max\{\mathcal{P}^{\text{MAM}}_{\text{RCSM}}(s_j, t_{i+j}), \mathcal{P}^{\text{MAM}}_{\text{LCSM}}(s_j, t_{i+j})\}$$

$$\mathcal{P}^{\text{EAM}}_{\text{CCM}}(s_j, t_{i+j}) := \max\{\mathcal{P}^{\text{EAM}}_{\text{RCSM}}(s_j, t_{i+j}), \mathcal{P}^{\text{EAM}}_{\text{LCSM}}(s_j, t_{i+j})\}$$

By probability theory, for RCSM-CEMA, LCSM-CEMA and CCM-DEMA with W traces, there are at most $W_1(j) := W \cdot \max_i(\mathcal{P}^{\text{MAM}}_{\text{RCSM}}(s_j, t_{i+j}))$, $W_2(j) := W \cdot \max_i(\mathcal{P}^{\text{MAM}}_{\text{LCSM}}(s_j, t_{i+j}))$ and $W_3(j) := W \cdot \max_i(\mathcal{P}^{\text{MAM}}_{\text{CCM}}(s_j, t_{i+j}))$ traces utilized in respective attacks if counted under MAM. In the same way, the corresponding results, namely $W'_1(j)$, $W'_2(j)$ and $W'_3(j)$, counted under EAM can be drawn.

For modes in Category II, it can be verified by enumeration that: $\mathcal{P}^{\text{MAM}}_{\text{CCM}}(s_j, t_{i+j}) = \mathcal{P}^{\text{MAM}}_{\text{RCSM}}(s_j, t_{i+j}) \geq \mathcal{P}^{\text{MAM}}_{\text{LCSM}}(s_j, t_{i+j})$ and $\mathcal{P}^{\text{EAM}}_{\text{CCM}}(s_j, t'_{i+j}) = \mathcal{P}^{\text{EAM}}_{\text{RCSM}}(s_j, t'_{i+j}) \geq \mathcal{P}^{\text{EAM}}_{\text{LCSM}}(s_j, t'_{i+j})$ for any i, j in their domains, so we have $W_3(j) = W_1(j) \geq W_2(j)$ and $W'_3(j) = W'_1(j) \geq W'_2(j)$. Combining with the fact that RCSM and RCSM employ the same distinguisher, we expect RCSM-CEMA performs no worse than LCSM-CEMA does, which is confirmed afterwards.

Theoretically, as explained above, MAM and EAM are related but different, while MAM and EAM are almost equivalent in practice, because (1) the time spent on a cache line loading operation is *very short*, which means the rectangles in **Fig**.5 would be much narrower, and (2) the leakage of the target operation happens in *a period of time* instead of just a moment as represented with lines like A, B, C in **Fig**.5. Hence, more traces in practical attacks than theoretically computed above are actually aligned on target operations, in a broader sense of alignment.

V. Evaluations

A. Setups

In our experiments, we target a NVIDIA GeForce GT 620 GPU connected to the host with a PCI-E bus. The GPU device has one SM of 48 SPs, a L2 cache of 64 KiB, and it is equipped with an off-chip memory of 454 MiB. More hardwares and softwares used in our experiments are listed in **TABLE** II.

As for evaluation criterion, we employ *(global) success rate* and *the number of recovered key bytes*. They are formalized as $\frac{m}{n}$ and $\frac{b}{n}$, respectively, where m denotes the number of tests that (fully) recover the secret key, b denotes the number of key bytes recovered in the m tests, and n denotes the total number of tests performed in an evaluation. We set $n = 10$ for V-B2, V-C1 and V-C2, and $n = 100$ for V-B1, V-B3, V-B4 and V-B5 in view of their required precision in the experiments.

Hardware/Software	*Model/Version*
GPU	NVIDIA GeForce GT 620[26]
Oscilloscope	KeySight DSO9104A
Host	ThinkCentre M8400t-N000
EM Probe	Rohde Schwarz RF B 3-2
AES source code	mbedTLS v2.5.1[25]
CUDA	CUDA 7.1

TABLE II: Setups of our experiments.

B. Experimental Results

1) For Category I

In the mode, 32 threads in a thread warp are given the same plaintext, so the execution time is constant, which means the measured traces are aligned on every operation including the S-Box TLU as target operation. As the TLU is one-staged for any trace, SSM is employed, and then SSM-based CEMA (SSM-CEMA) is performed. The experiment for the category also aims to find an optimal sampling rate used for oscilloscope to measure traces for the follow-up experiments. Considering the space requirement for storing traces, we measure traces at four different sampling rates up to 2 GSa/s. The experimental results are shown in **Fig**.6. We find SSM-CEMA with traces sampled at 2 GSa/s performs just a little bit better than it does at 500 MSa/s, so 500 MSa/s is assumed to be a good choice with the consideration of both efficiency and the size of trace data, and thus applied in the follow-up experiments. With the traces sampled at 500 MSa/s, approximately 15,000 traces are required to recover the 128-bit secret key. Intuitively, the mode may be the easiest one in which a side-channel attack is performed to recover the secret key, because in this mode (1) a single plaintext is encrypted in all 32 threads, so it should produce low noise level; and (2) only one-staged S-Box TLU happens, so the traces are aligned on any operation. However, it is actually not like this, and SSM-CEMA in one-group mode is not the best attack as expected, which will be proved by introducing more powerful attacks in the experiments below.

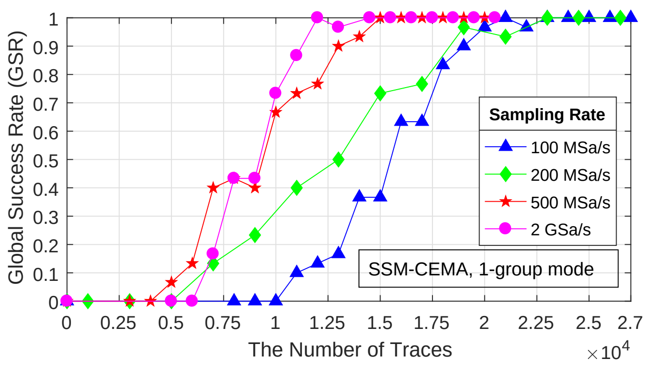

Fig. 6: The performance of SSM-CEMA for mode in Category I.

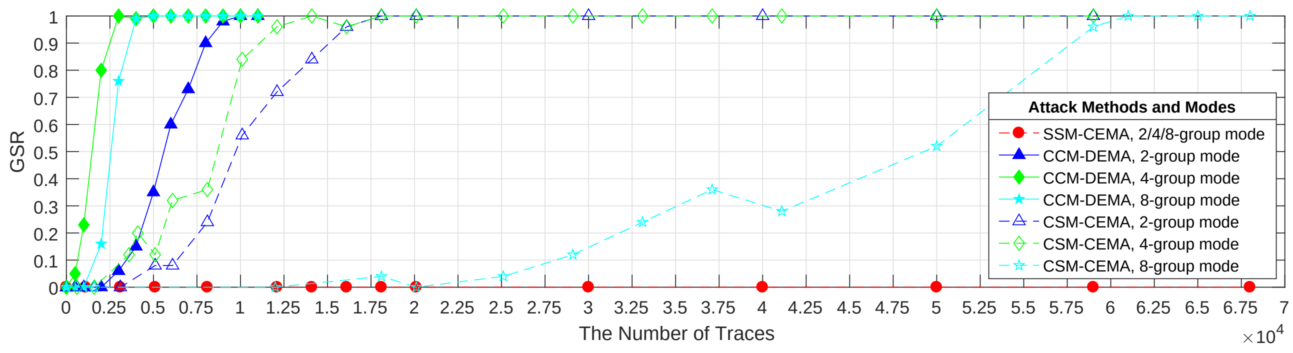

Fig. 7: The comparison amongst SSM-CEMA, CSM-CEMA (LCSM) and CCM-DEMA of their performance for modes in Category II.

2) *For Category III*

In the modes, S-Box TLU is two-staged for almost all traces, so CSM is employed and CSM-based CEMA (CSM-CEMA) is performed. In the meanwhile, an SSM-CEMA is used for comparison. Note that CCM does not work for modes in Category III and Category I, because CCM requires that the number of STGs be not constant across traces. The constant also implies that traces are aligned in the modes. As 16 or 32 threads encrypt different plaintexts, a high noise level (η in **Eq**.1) is introduced, which increases the number of traces used in attacks and, at the same time, narrows the performance gap between SSM-CEMA and CSM-CEMA (LCSM) in the modes. As showed in **Fig**.8, both SSM-CEMA and CSM-CEMA (LCSM) consume more than 150,000 and 300,000 traces in 16-group mode and 32-group mode, respectively. Nevertheless, CSM-CEMA (LCSM) still outperforms SSM-CEMA in both modes.

Note that the experimental results are drawn in 10 instead of 100 tests, namely $n = 100$ (V-A) adopted in the other experiments, because the attacks in the modes are much time-consuming when performed on desktop computers. To remedy this, we evaluate *the number of recovered key bytes* rather than *global success rate*, because the former is less sensitive to the number of tests than the latter. In addition to this, it is also noted that we use LCSM instead of RCSM in the CSM-CEMA, because RCSM is more powerful than LCSM. Using the worst case makes performance comparison among different models more convincing. We also fulfill the principle in V-B3 and V-B4.

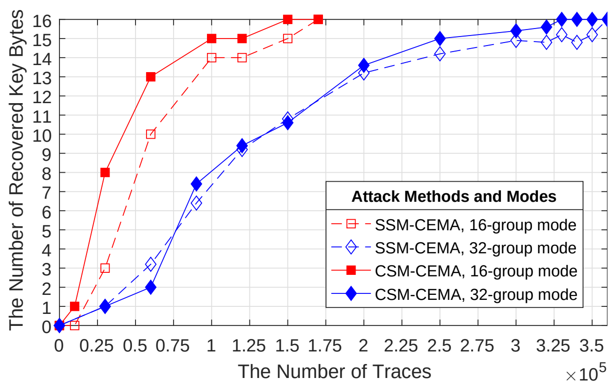

Fig. 8: The comparison between SSM-CEMA and CSM-CEMA (LCSM) of their performance for modes in Category III.

#Key Enum.	*SSM-CEMA*	*CSM-CEMA*	*CCM-DEMA*
———	≈ 12,100 traces	≈ 18,100 traces	≈ 5,000 traces
1 byte	≈ 6,100 traces	≈ 2,100 traces	≈ 600 traces
	≈ 0.05 ms on key search		
2 bytes	≈ 4,600 traces	≈ 1,100 traces	≈ 320 traces
	≈ 100.67 ms on key search		
3 bytes	≈ 4,100 traces	≈ 820 traces	≈ 280 traces
	≈ 120.26 sec on key search		
4 bytes	≈ 3,600 traces	≈ 700 traces	≈ 240 traces
	≈ 27.79 hr on key search		

TABLE III: A summary of the performance in our experiments, of SSM-CEMA, CSM-CEMA (LCSM) and CCM-DEMA with and without brute-force search.

δ-group Mode	*SSM-CEMA*	*CSM-CEMA*	*CCM-DEMA*
$\delta = 1$	**≈12,100**		-
$\delta = 2$	70,000+	≈18,100	≈10,000
$\delta = 4$	70,000+	**≈18,100**[*]	**≈5,000**[*]
$\delta = 8$	70,000+	≈62,000	≈5,000
$\delta = 16$	≈170,000	≈150,000	-
$\delta = 32$	≈360,000	≈330,000	-

[*] Optimal by trend.

TABLE IV: The comparison among SSM-CEMA, CSM-CEMA (LCSM) and CCM-DEMA of performance in the six modes.

3) *For Category II*

This is the most complicated one among the categories, because in these modes, the execution of the 32 threads makes the measured traces misaligned, which is known to be one of great challenges for side-channel attacks. The advantage is both CSM and CCM are applicable, so we evaluate SSM-CEMA, CSM-CEMA (LCSM) and CCM-DEMA in these modes. The evaluation results are showed in **Fig**.7. Obviously, SSM-CEMA remains ineffective when using up to 70,000 traces, while CSM-CEMA (LCSM) and CCM-DEMA are able to fully recover the secret key with less traces. CCM-DEMA outperforms CSM-CEMA (LCSM) in all the modes, and both perform best in 4-group mode. More precisely, about 3,000 traces are required for CCM-DEMA, and 15,000 traces for CSM-CEMA (LCSM). This shows that CCM-DEMA performs much better than CSM-CEMA (LCSM) does, which is consistent with the theoretical prediction in IV-D.

4) *Across Categories*

In the above, the comparisons are made among the models only in the same modes, so it is still unknown that which model performs better without the consideration of modes. In other words, it is asking a question like which model(s)

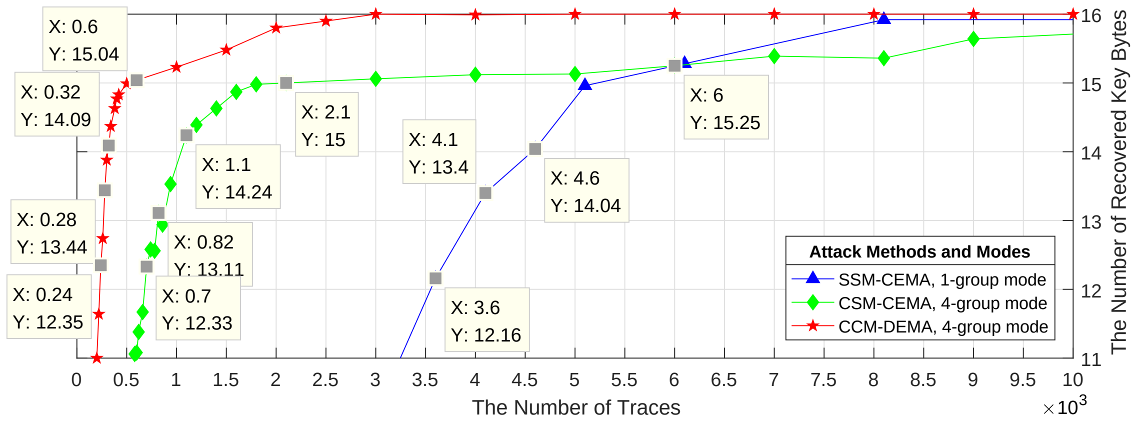

Fig. 9: The comparison amongst SSM-CEMA, CSM-CEMA (LCSM) and CCM-DEMA with 1, 2, 3 or 4 byte(s) key enumeration, of their best performance in the six modes.

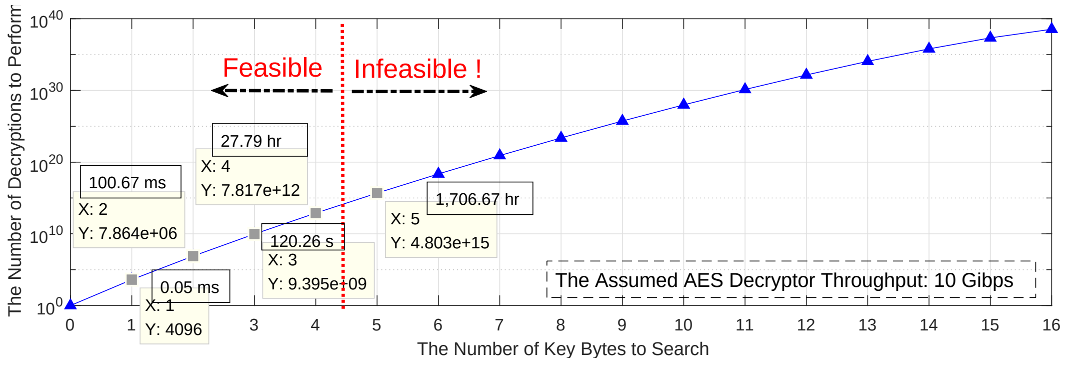

Fig. 10: The number of decryptions needed for and the time spent on brute-force search.

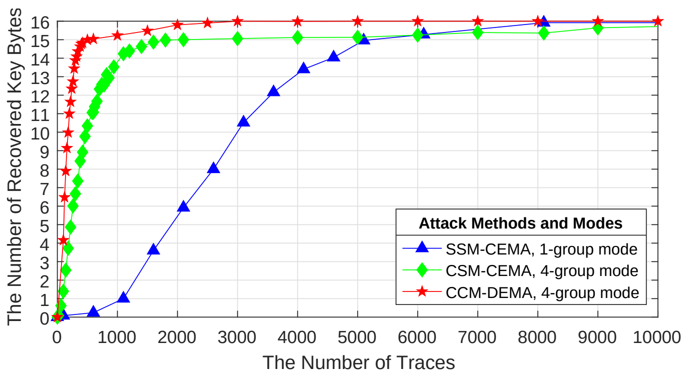

Fig. 11: The comparison amongst SSM-CEMA, CSM-CEMA (LCSM) and CCM-DEMA of their best performance in the six modes.

is(are) more efficient if attackers are able to choose any one of the six modes as they like? The question is actually quite practical, because it imposes less constraints on attackers. To answer the question, more evaluations should be given across categories. Based on the previous evaluations, the performance of SSM-CEMA, CSM-CEMA (LCSM) and CCM-DEMA in the six modes are summarized in **TABLE** IV. It is obvious that SSM-CEMA in 1-group mode, CSM-CEMA (LCSM) and CCM-DEMA in 4-group mode perform best for the respective method in all the modes, so CCM-DEMA in 4-group mode should be employed by an attacker. Although that is true, it is necessary to give a deeper insight into the performance of CSM-CEMA (LCSM) in 4-group mode and SSM-CEMA in 1-group mode. We evaluate in another criteria; that is, *the number of recovered key bytes* mentioned before, because the performance of recovering less than 16 key bytes is also of great significance; that is to say, attackers can take a strategy of recovering some key bytes with distinguishers and the rest with *key enumeration algorithms* (KEAs). *Brute-force search* is the most basic KEA. The cost for searching partial key bytes of AES with brute-force search is shown in **Fig**.10. The assumed AES decryption throughput (10 Gibps) comes from a typical performance of AESNI-enabled processors. As showed, it is practical to brute-force search 1, 2, 3 (or 4) key bytes, which corresponds to recover any 15, 14, 13 (or 12) of 16 key bytes. As showed in **Fig**.9,11, especially **Fig**.9, summarized in **TABLE** III, we conclude that CSM-CEMA (LCSM) does perform better than SSM-CEMA in practice.

5) *With Optimizations*

The optimizations are based on three decision-level combinations, namely DCA$^+$, DCA$^\times$ and DCA$^{\max}$, that combine distinguishers of CSMs and/or CCMs. As mentioned before, two CSMs (LCSM and RCSM) and one CCM are available for the target AES implementation on the G-PU, so we evaluate the six combinational attacks named LCSM-RCSM-DCA$^+$, LCSM-RCSM-DCA$^\times$, LCSM-RCSM-DCA$^{\max}$, LCSM-RCSM-CCM-DCA$^+$, LCSM-RCSM-CCM-

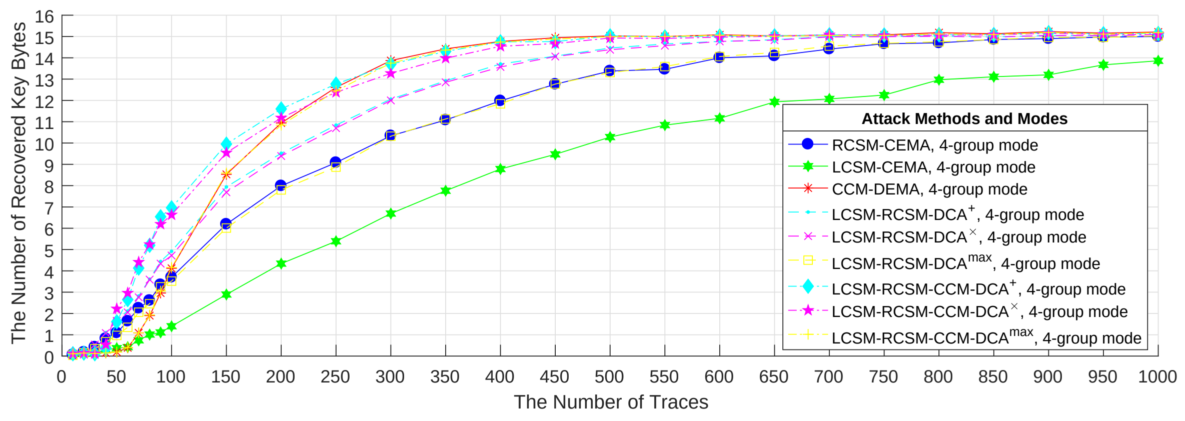

Fig. 12: The comparison between non-combinational (CSM-CEMAs, CCM-DEMA) and combination-based attacks of their performance in 4-group mode.

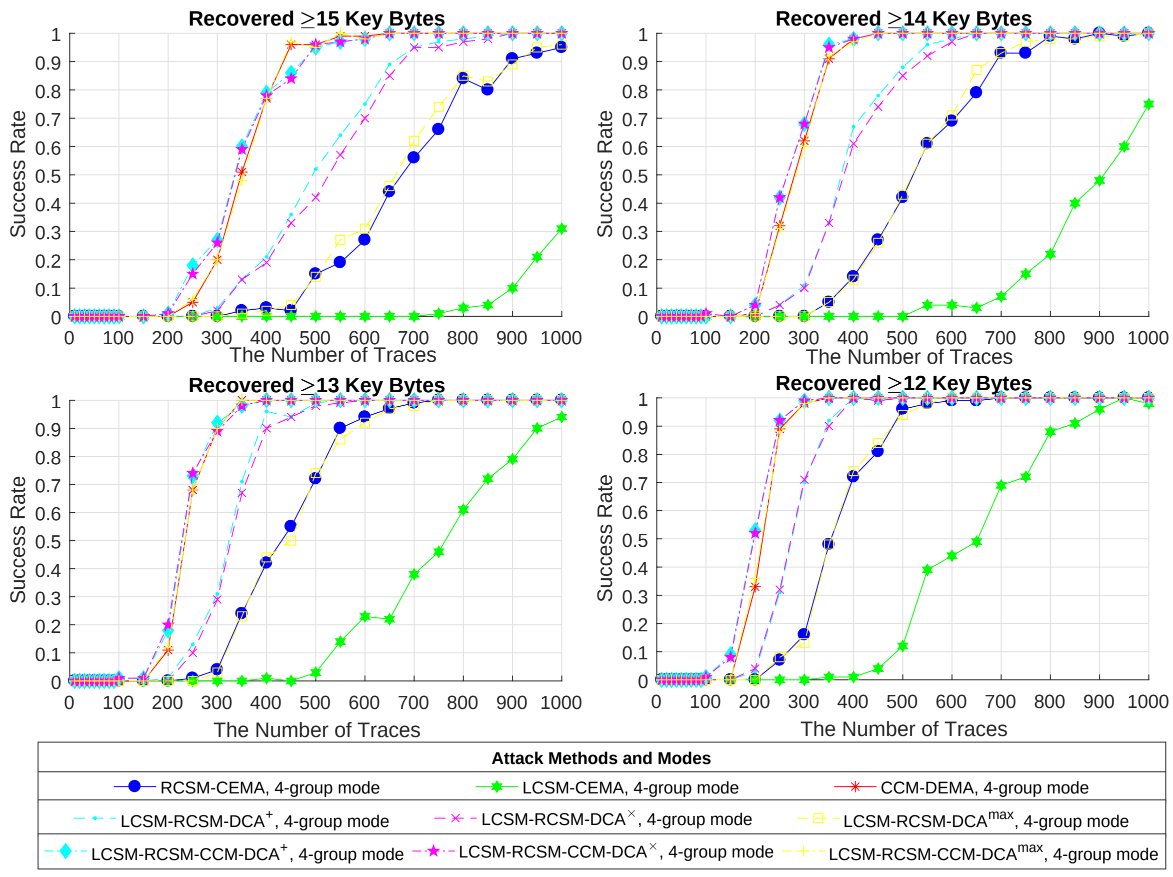

Fig. 13: The comparison between non-combinational (CSM-CEMAs, CCM-DEMA) and combination-based attacks with 1, 2, 3 or 4 byte(s) key enumeration, of their performance in 4-group mode.

DCA$^{\times}$ and LCSM-RCSM-CCM-DCA$^{\max}$. Their names follow the naming convention in III-D. Since both CSM-CEMA (LCSM) and CCM-DEMA hit the best performance in 4-group mode, the evaluations are also made in the mode. The same way, we consider the previous attack strategy, i.e. distinguisher (DCA^{+}, DCA$^{\times}$ or DCA$^{\max}$) plus KEA (brute-force search). As shown in **Fig**.12, LCSM-RCSM-DCA$^{\max}$ is not effective, and it performs almost the same as RCSM-CEMA does, which suggests LCSM-RCSM-DCA$^{\max}$ makes little/no use of the decision information from the distinguisher of LCSM. It also shows that the performance of LCSM-RCSM-DCA^{+} and LCSM-RCSM-DCA$^{\times}$ are almost the same and both are better than that of LCSM-CEMA, RCSM-CEMA and LCSM-RCSM-DCA$^{\max}$ with the assistance of brute-force search. Of the other combination-based attacks, LCSM-RCSM-CCM-DCA$^{\max}$ is almost ineffective as LCSM-RCSM-DCA$^{\max}$ does, LCSM-RCSM-CCM-DCA^{+} and LCSM-RCSM-CCM-DCA$^{\times}$ perform similarly and better than either of the non-combinational attacks, but become closer to CCM-DEMA with more traces. The same trends are also shown in **Fig**.13 for evaluations in *success rate*, so we will not repeat them.

C. Simulations

Since all the evaluations above are made in a specific GPU device, we cannot conclude the methods are effective/efficient to other GPU devices as well. In light of this, we introduce simulation experiments on σ^2-traces under the assumption that the EM traces from different GPU devices differ only in terms of Gaussian noise, thus bearing different SNR, where σ^2-traces denote the traces mixed with Gaussian noise of 0-mean and σ^2-variance. More specifically, the simulation experiments are based on $\boldsymbol{L}^{(\cdot,\sigma^2)}$ (II-D) generated by **Algo**.4.

Suppose SNR of original (measured) traces $\boldsymbol{L}^{(\cdot)}$ is represented as $\xi = \sigma_S^2/\sigma_N^2$, where σ_S^2 and σ_N^2 denote the variance of the signal part and noise part, respectively. Since we do not know the exact value of ξ, σ_S^2 or σ_N^2, we try on $\boldsymbol{L}^{(1,\sigma^2)}$-s of different σ^2 values until we find a σ^2 sequence so that the trend of performance of SSM-CEMA on the $\boldsymbol{L}^{(1,\sigma^2)}$-s are clearly distinguishable. As a result, we find the σ^2-s, $\{0.0001, 0.00025, 0.0005, 0.001, 0.01, 0.1\}$ meets the needs. Finally, we generate $\boldsymbol{L}^{(1,0.0001)}$, $\boldsymbol{L}^{(1,0.00025)}$, $\boldsymbol{L}^{(1,0.0005)}$, $\boldsymbol{L}^{(1,0.001)}$, $\boldsymbol{L}^{(1,0.01)}$, $\boldsymbol{L}^{(1,0.1)}$, $\boldsymbol{L}^{(4,0.001)}$, $\boldsymbol{L}^{(4,0.01)}$ and $\boldsymbol{L}^{(4,0.1)}$ for the experiments below.

1) For Category I on σ^2-traces

The experiment aims to find the upper bound of σ^2 with which SSM-CEMA in 1-group mode works, and then evaluate SSM-CEMA on $\boldsymbol{L}^{(1,\sigma^2)}$. As **Fig**.14 shows, the upper bound is no more than 0.001, because SSM-CEMA is almost ineffective on $\boldsymbol{L}^{(1,0.001)}$. Also, it shows as expected the performance of SSM-CEMA in 1-group mode decreases as the σ^2 increases.

Algorithm 4 σ^2-traces Generation

```
Require: Original traces: $\boldsymbol{L}^{(\cdot)}$ of $W \times M$, $\sigma^2$.
Ensure: $\sigma^2$-traces: $\boldsymbol{L}^{(\cdot,\sigma^2)}$.
1: for $i \leftarrow 1$ to $W$ do
2:     for $j \leftarrow 1$ to $M$ do
3:         $R \leftarrow N(0,\sigma^2)$                ▷ generate Gaussian random
4:         $\boldsymbol{L}_{i,j}^{(\cdot,\sigma^2)} \leftarrow \boldsymbol{L}_{i,j}^{(\cdot)} + R$
5: return $\boldsymbol{L}^{(\cdot,\sigma^2)}$
```

2) Across Categories on σ^2-traces

As known from the above experiment (V-C1), SSM-CEMA does not work on $\boldsymbol{L}^{(1,\sigma^2)}$ where $\sigma^2 \geq 0.001$. Then how about the performance of CSM-CEMA and CCM-DEMA on $\boldsymbol{L}^{(4,\sigma^2)}$ when $\sigma^2 \geq 0.001$?

We make evaluations on $\boldsymbol{L}^{(4,0.001)}$, $\boldsymbol{L}^{(4,0.01)}$ and $\boldsymbol{L}^{(4,0.1)}$, and the performance are shown in **Fig**.15, **Fig**.16 and **Fig**.17, respectively. We find that both methods still work well, even if SSM-CEMA does not work at all when $\sigma^2 = 0.001, 0.01, 0.1$. The experimental results are beyond our expectations, and they suggest that CSM and CCM are not only more efficient but also more robust than SSM under CTA.

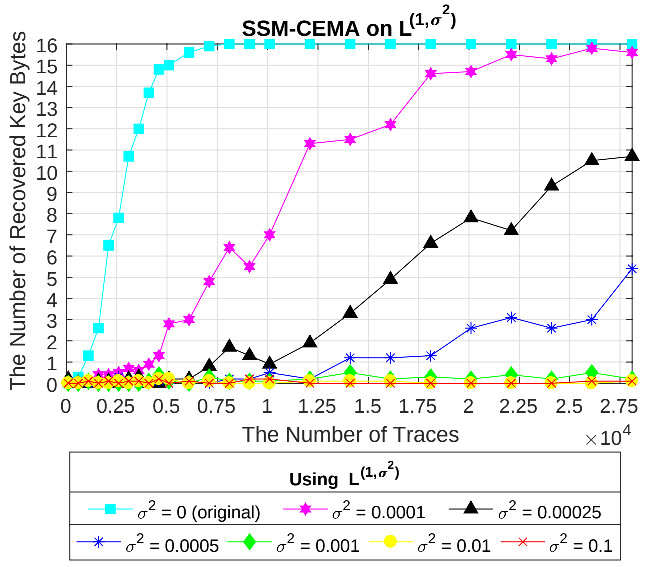

Fig. 14: The performance of SSM-CEMA on $\boldsymbol{L}^{(1,\sigma^2)}$.

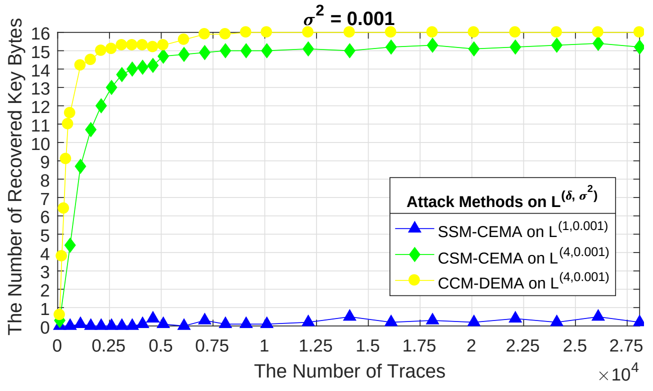

Fig. 15: The comparison among SSM-CEMA on $\boldsymbol{L}^{(1,0.001)}$, CSM-CEMA (RCSM) on $\boldsymbol{L}^{(4,0.001)}$ and CCM-DEMA on $\boldsymbol{L}^{(4,0.001)}$of their performance.

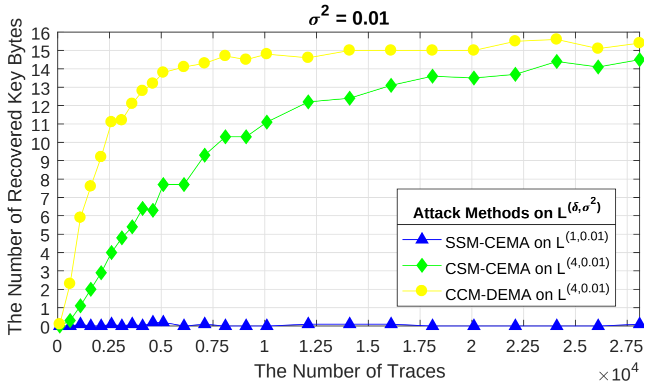

Fig. 16: The comparison among SSM-CEMA on $\boldsymbol{L}^{(1,0.01)}$, CSM-CEMA (RCSM) on $\boldsymbol{L}^{(4,0.01)}$ and CCM-DEMA on $\boldsymbol{L}^{(4,0.01)}$of their performance.

VI. Discussions

A. Synchronization

We can see that CSM and CCM are the key proposals of this paper, and both of them are based on the threads partitioner CTP, so we would better verify in some way if the CTP is reasonable or not. Actually, we tried leakage detection method on each single thread, in order to give an visual presentation of the leakages from multiple threads in the same time domain, but we found it difficult than expected, because for multi-thread scenarios the leakage detection to any one thread also means treating the leakages from the other

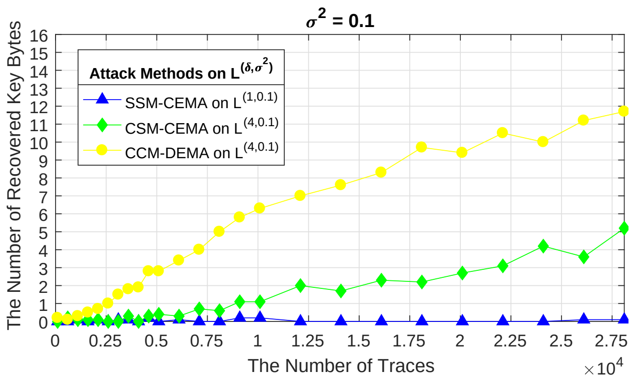

Fig. 17: The comparison among SSM-CEMA on $\boldsymbol{L}^{(1,0.1)}$, CSM-CEMA (RCSM) on $\boldsymbol{L}^{(4,0.1)}$ and CCM-DEMA on $\boldsymbol{L}^{(4,0.1)}$of their performance.

threads as noise. The high noise makes the leakage detection rather tricky. In spite of this, there are evidences indicating the correctness of CTP. On the one hand, previous timing attacks, such as [8], [10], only depend on the CTP, so the effectiveness of the timing attacks indicates ① *the threads in different subsets partitioned by the CTP are not synchronous at TLU operation*. That CSM-CEMA outperforms SSM-CEMA in 32-group mode (see **Fig**.8) also indicates the statement, because SSM always introduces additional noise when the multiple threads are not synchronous at target operation (see **Eq**.5). On the other hand, for the same reason the hardness of leakage detection on single thread indicates ② *the threads in the same subset partitioned by the CTP are synchronous at TLU operation*; otherwise, the leakage detection cannot be harder than CSM-CEMA in 32-group mode, where the hardness is measured by the number of traces. As a result, ① plus ② indicates CTP is reasonable.

B. Countermeasure

Since CSM and CCM are the key proposals of this paper, we give a brief discussion below on countermeasures to them. Actually, the papers [27], [28] have investigated countermeasures to MIS-based side-channel attacks. The countermeasures are able to resist CSM-CEMA and CCM-DEMA but require hardware alternation, so they are not portable in practice. In addition, as one of generic countermeasures, masking is also effective to CSM-CEMA and CCM-DEMA, but the naive way of applying a masking countermeasure to GPU-based cryptographic implementations will consume thousands of times as many random numbers as the masking countermeasure to a CPU-based one, thus greatly reducing the performance. Therefore, it is still an open question and needs further studies.

VII. Conclusions

In the paper, we investigate side-channel cryptanalysis with multi-thread mixed leakage. We propose several leakage models, of which SSM and PSM are for general cryptographic implementations on general SIMT systems, while CSM and CCM are for table-based implementations on specific SIMT systems, precisely CUDA-enabled GPUs. In addition to the models, combination-based techniques are employed for optimization on CSM- and/or CCM-based attacks. Theoretically, CSM, CCM and their optimizations apply to attacks in any δ-group mode against any table-based cryptographic implementations running on any CUDA-enabled GPU, as long as the size of the table is larger than the size of the L1 cache line, but it needs further verifications in practice, because as explained, if more cache lines are required for storing target LUT, for some modes the pattern of trace alignment becomes more sophisticated, which may eventually frustrate the attacks.

To summarize up, our study proposes methodologies and achieves efficient practical side-channel attacks against a table-based GPU AES implementation. The research suggests that GPU-based cryptographic implementations may be much vulnerable to microarchitecture-based side-channel attacks. Therefore, GPU-specific countermeasures should be considered for GPU-based cryptographic implementations in practical applications.

References

[1] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, 1996, pp. 104–113. [Online]. Available: https://doi.org/10.1007/3-540-68697-5_9

[2] R. Szerwinski and T. Güneysu, "Exploiting the power of gpus for asymmetric cryptography," in *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, 2008, pp. 79–99.

[3] X. Kang, B. George, and K. Lueh, "Efficient implementation of rsa using gpu/cpu architecture," Feb. 16 2016, uS Patent 9,262,166.

[4] Q. Li, C. Zhong, K. Zhao, X. Mei, and X. Chu, "Implementation and analysis of AES encryption on GPU," in *14th IEEE International Conference on High Performance Computing and Communication & 9th IEEE International Conference on Embedded Software and Systems, HPCC-ICESS 2012, Liverpool, United Kingdom, June 25-27, 2012*, 2012, pp. 843–848.

[5] O. Hajihassani, S. K. Monfared, S. H. Khasteh, and S. Gorgin, "Fast AES implementation: A high-throughput bitsliced approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 10, pp. 2211–2222, 2019. [Online]. Available: https://doi.org/10.1109/TPDS.2019.2911278

[6] C. Luo, Y. Fei, P. Luo, S. Mukherjee, and D. R. Kaeli, "Side-channel power analysis of a GPU AES implementation," in *33rd IEEE International Conference on Computer Design, ICCD 2015, New York City, NY, USA, October 18-21, 2015*, 2015, pp. 281–288.

[7] C. Luo, Y. Fei, L. Zhang, A. A. Ding, P. Luo, S. Mukherjee, and D. R. Kaeli, "Power analysis attack of an AES GPU implementation," *J. Hardware and Systems Security*, vol. 2, no. 1, pp. 69–82, 2018. [Online]. Available: https://doi.org/10.1007/s41635-018-0032-7

[8] Z. H. Jiang, Y. Fei, and D. R. Kaeli, "A complete key recovery timing attack on a GPU," in *2016 IEEE International Symposium on High Performance Computer Architecture, HPCA 2016, Barcelona, Spain, March 12-16, 2016*, 2016, pp. 394–405.

[9] ——, "A novel side-channel timing attack on GPUs," in *Proceedings of the on Great Lakes Symposium on VLSI 2017, Banff, AB, Canada, May 10-12, 2017*, 2017, pp. 167–172.

[10] E. Karimi, Z. H. Jiang, Y. Fei, and D. R. Kaeli, "A timing side-channel attack on a mobile GPU," in *36th IEEE International Conference on Computer Design, ICCD 2018, Orlando, FL, USA, October 7-10, 2018*, 2018, pp. 67–74. [Online]. Available: https://doi.org/10.1109/ICCD.2018.00020

[11] Z. H. Jiang, Y. Fei, and D. R. Kaeli, "Exploiting bank conflict-based side-channel timing leakage of gpus," *TACO*, vol. 16, no. 4, pp. 42:1–42:24, 2020. [Online]. Available: https://doi.org/10.1145/3361870

[12] D.B.Fomin, "A timing attack on CUDA implementations of an AES-type block cipher," *Mathematical Aspects of Cryptography, 2016, v.7, No 2, pp.121-130 (Russian)*, 2016.

[13] Y. Gao, Y. Zhou, and W. Cheng, "Efficient electro-magnetic analysis of a GPU bitsliced AES implementation," *Cybersecurity*, vol. 3, no. 1, p. 3, 2020. [Online]. Available: https://doi.org/10.1186/s42400-020-0045-8

[14] Y. Gao, W. Cheng, H. Zhang, and Y. Zhou, "Cache-collision attacks on gpu-based AES implementation with electro-magnetic leakages," in *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 12th IEEE International Conference On Big Data Science And Engineering, TrustCom/BigDataSE 2018, New York, NY, USA, August 1-3, 2018*, 2018, pp. 300–306. [Online]. Available: https://doi.org/10.1109/TrustCom/BigDataSE.2018.00053

[15] Y. Gao, H. Zhang, W. Cheng, Y. Zhou, and Y. Cao, "Electro-magnetic analysis of gpu-based AES implementation," in *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, 2018, pp. 121:1–121:6. [Online]. Available: https://doi.org/10.1145/3195970.3196042

[16] F. Liu and R. B. Lee, "Security testing of a secure cache design," in *HASP 2013, The Second Workshop on Hardware and Architectural Support for Security and Privacy, Tel-Aviv, Israel, June 23-24, 2013*, 2013, p. 3. [Online]. Available: https://doi.org/10.1145/2487726.2487729

[17] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, 2015, pp. 605–622. [Online]. Available: https://doi.org/10.1109/SP.2015.43

[18] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, 2004, pp. 16–29. [Online]. Available: https://doi.org/10.1007/978-3-540-28632-5_2

[19] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, 1999, pp. 388–397. [Online]. Available: https://doi.org/10.1007/3-540-48405-1_25

[20] W. Schindler, "A combined timing and power attack," in *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*, 2002, pp. 263–279. [Online]. Available: https://doi.org/10.1007/3-540-45664-3_19

[21] Y. Souissi, S. Bhasin, S. Guilley, M. Nassar, and J. Danger, "Towards different flavors of combined side channel attacks," in *Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings*, 2012, pp. 245–259. [Online]. Available: https://doi.org/10.1007/978-3-642-27954-6_16

[22] W. Yang, Y. Zhou, Y. Cao, H. Zhang, Q. Zhang, and H. Wang, "Multi-channel fusion attacks," *IEEE Trans. Information Forensics and Security*, vol. 12, no. 8, pp. 1757–1771, 2017. [Online]. Available: https://doi.org/10.1109/TIFS.2017.2672521

[23] P. S. Shastry, N. Somani, A. Gadre, B. Vispute, and M. S. Sutaone, "Rolled architecture based implementation of aes using t-box," in *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2012, pp. 626–630.

[24] N. Nishikawa, H. Amano, and K. Iwai, "Implementation of bitsliced AES encryption on cuda-enabled GPU," in *Network and System Security - 11th International Conference, NSS 2017, Helsinki, Finland, August 21-23, 2017, Proceedings*, 2017, pp. 273–287.

[25] PolarSSL/mbedTLS, "An open source SSL library licensed by ARM limited," https://tls.mbed.org.

[26] Wikipedia, "GeForce 500 series," https://en.wikipedia.org/wiki/GeForce_500_series.

[27] G. Kadam, D. Zhang, and A. Jog, "Rcoal: Mitigating GPU timing attack via subwarp-based randomized coalescing techniques," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*. IEEE Computer Society, 2018, pp. 156–167. [Online]. Available: https://doi.org/10.1109/HPCA.2018.00023

[28] ——, "Bcoal: Bucketing-based memory coalescing for efficient and secure gpus," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22-26, 2020*. IEEE, 2020, pp. 570–581. [Online]. Available: https://doi.org/10.1109/HPCA47549.2020.00053