How Does Strict Parallelism Affect Security? A Case Study on the Side-Channel Attacks against GPU-based Bitsliced AES Implementation

Yiwen GAO[1,2], Yongbin ZHOU[1,2(✉)], Wei CHENG[1]
[1]State Key Laboratory of Information Security,
Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{gaoyiwen,zhouyongbin,chengwei}@iie.ac.cn

Abstract*—Parallel cryptographic implementations are generally considered to be more advantageous than their non-parallel counterparts in mitigating side-channel attacks because of their higher noise-level. So far as we know, the side-channel security of GPU-based cryptographic implementations have been studied in recent years, and those implementations then turn out to be susceptible to some side-channel attacks. Unfortunately, the target parallel implementations in their work do not achieve strict parallelism because of the occurrence of cached memory accesses or the use of conditional branches, so how strict parallelism affects the side-channel security of cryptographic implementations is still an open problem. In this work, we make a case study of the side-channel security of a GPU-based bitsliced AES implementation in terms of bit-level parallelism and thread-level parallelism in order to show the way that works to reduce the side-channel security of strict parallel implementations. We present GPU-based bitsliced AES implementation as the study case because (1) it achieves strict parallelism so as to be resistant to cache-based attacks and timing attacks; and (2) it achieves both bit-level parallelism and thread-level parallelism (a.k.a. task-level parallelism), which enables us to research from multiple perspectives. More specifically, we first set up our testbed and collect electro-magnetic (EM) traces with some special techniques. Then, the measured traces are analyzed in two granularity. In bit-level parallelism, we give a non-profiled leakage detection test before mounting attacks with our proposed bit-level fusion techniques like multi-bits feature-level fusion attacks (MBFFA) and multi-bits decision-level fusion attacks (MBDFA). In thread-level parallelism, a profiled leakage detection test is employed to extract some special information from multi-threads leakages, and with the help of those information our proposed multi-threads hybrid fusion attack (MTHFA) method takes effect. Last, we propose a simple metric to quantify the side-channel security of parallel cryptographic implementations. Our research shows that the secret key of our target implementation can be recovered with less cost than expected, which suggests that the side-channel security of parallel cryptographic implementations should be reevaluated before application.

***Keywords*—Side-Channel Attacks (SCA), Side-Channel Fusion Attacks (SCFA), Electro-Magnetic Attacks (EMA), Strict Parallel Cryptographic Implementation, Warp Asynchronous Leakages**

I. Introduction

Parallel computing is a type of computation in which many calculations or the execution of processes are carried out concurrently. Generally speaking, large problems can often be broken down into smaller ones, which can then be solve at the same time. Due to the urgent need of high-performance computing power in many areas, parallel computing has become the dominant paradigm in computer architecture. Nowadays as the most widely used parallel computing platform, Graphics Processing Unit (GPU) has evolved from a specialized hardware for graphics rendering into a general-purpose computing device for various applications as biomedical analysis, signal processing, scientific computing and so on. GPU is designed as an SIMT (Single Instruction, Multiple Threads) device and well suited to cryptographic applications deployed in cloud computing environment. However, GPU-based cryptographic applications are vulnerable to many known attacks as proposed in [1][2][3]. Among those published vulnerabilities of GPUs, side-channel attacks are the most serious ones due to their non-invasiveness to target devices. In recent years, the studies on the side-channel attacks against cryptographic implementations have always been a research hotspot of cryptanalysis beyond algebraic analysis methods. As the most popular block cipher, AES is widely deployed on a wide variety of hardware platforms. The side-channel attacks against AES software implementations on CPUs/MCUs and hardware implementations on FPGAs have been deeply studied. Until very recently, some literatures mentioned that GPU-based cryptographic implementations are also susceptible to side-channel attacks through electro-magnetic emanation, power consumption or execution time leakages [4][5][3][6]. Unfortunately, these attacks are based on the special architectural features of CUDA-enabled GPUs like cache line access coalescing/serialization [4][3], shared memory bank conflict [6], and high occupancy of threads [5], which make the target implementations not strict parallel any more. Therefore, it is still an open problem about the side-channel security of cryptographic implementations with strict parallelism. In light of this, we study the side-channel attacks against a GPU-based bitsliced AES implementation in order to give a deep insight into the mechanisms mitigating the security of strict parallel cryptographic implementation.

In the work, we take GPU-based bitsliced AES implementation as the study case for two reasons.

First, GPU-based bitsliced AES implementation achieves strict parallelism across threads in a warp, which means threads in a warp execute the same instruction at any moment without exceptions like cache line access serialization and conditional branches that needs coordination among threads. In addition, the implementation is assumed to be resistant to cache attacks and timing attacks due to the constant execution time of strict parallelism.

Second, GPU-based bitsliced AES implementation achieves both bit-level parallelism and thread-level parallelism (a.k.a. task-level parallelism), which enables us to research from multiple perspectives. The two types of parallelism cover the major granularity of parallelisms, so it is of great significance to our research.

To the best of our knowledge, this is the first work to investigate the side-channel security of strict parallel cryptographic implementation in two granularity, i.e. bit-level parallelism and thread-level parallelism. Our contributions are summarized as follows:

• We study the side-channel security of bit-level parallelism in GPU-based bitsliced AES implementation. A non-profiled leakage detection method is employed to find special leakage patterns on multiple bits, and then multi-bits feature-level fusion attack (MBFFA) and multi-bits decision-level fusion attack (MBDFA) are proposed to analyze the side-channel security of the implementation.

• We study the side-channel security of thread-level parallelism in GPU-based bitsliced AES implementation. A profiled leakage detection method is employed to find special leakage patterns on multiple threads, and then multi-threads hybrid fusion attack (MTHFA) is proposed to analyze the side-channel security of the implementation.

• We also propose a simple metric to assess the side-channel security of a certain parallel cryptographic implementation for certain attacks, which is very useful in security assessments.

The rest of this paper is organized as follows. In section II, we give a brief introduction of CUDA-enabled GPU and GPU-based bitsliced AES implementation. In section III, we present some special techniques used in leakage acquisition and preprocessing. In section IV, we investigate the side-channel security of bit-level parallelism of the target implementation. In section V, we study the side-channel security of strict thread-level parallelism of target implementation. In section VI, we introduce a simple metric to assess the security of parallelism. In section VII, related work are listed. Finally, conclusions and future works are given in section VIII.

II. Preliminary

In this section, we give a brief introduction to the architecture of CUDA-enabled GPUs, the features of GPU-based bitsliced AES implementation as well as the definitions and notations involved in this paper.

A. CUDA-enabled GPU

Compute Unified Device Architecture (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$\times$ Streaming Multiprocessors (SM) and a global memory. Each SM has N$\times$ Scalar Processor (SP), a shared memory, several 32-bits registers, and a shared instruction unit. 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 gird 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.

B. GPU-based Bitsliced AES Implementation

The term bitsliced cipher was first proposed by Eli Biham [7] referring to the AES candidate Serpent. More precisely, it is a concept about cryptographic implementation instead of cryptographic algorithm or scheme itself.

The AES implementation of bitsliced version could process more than one plaintexts in a parallel fashion. The parallelism of plaintexts is determined by the length of machine word in bit. For 32-bit processors, 32 plaintexts can be encrypted simultaneously, which is also mentioned as *bit-level parallelism*. The first step of bitsliced AES implementation is to transpose multiple plaintexts by bit in order to adapt bitsliced execution fashion. As shown in Fig.1, 32 128-bit plaintexts are arranged by row, and each plaintext is written to or read from four 32-bit registers within GPU. The 32$\times$128 matrix is transposed before the first round encryption, and the inverse transposition is performed after the final round encryption. It is obvious that only one forward transposition and one inverse transposition are needed to finish one bitsliced AES encryption on a single GPU thread. For multiple threads execution on GPU, each thread executes the above process independently, which is also called *thread-level parallelism*. In a word, GPU-based bitsliced AES implementation achieves parallelism in two granularity, i.e. both bit-level parallelism and thread-level parallelism, so it is well suited for us to study the side-channel security of parallel cryptographic implementations.

C. Definitions and Notations

Definition 1. *For multi-threads cryptographic implementations, if an attacker is able to choose the same random plaintexts for some threads and collect the corresponding ciphertexts and side-channel leakages, it is called* ***Chosen-Thread Side-Channel Attacks (CTSCA)****. Attackers cannot choose specific plaintexts in CTSCA, which is different from Chosen-Plaintext Side-Channel Attacks (CPSCA).*

Definition 2. *When performing a CTSCA, if attackers choose the same random plaintexts for the consecutive M threads of totally M$\times$N threads, it is called* ***N-Group CTSCA****.*

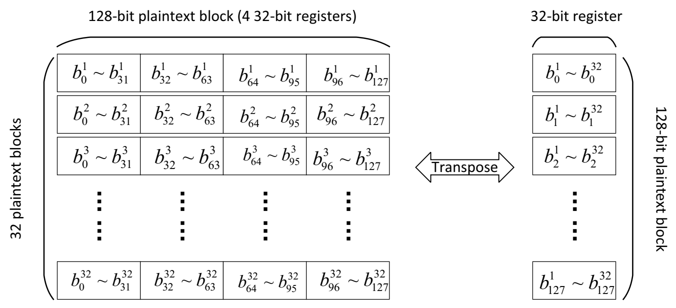
128-bit plaintext block (4 32-bit registers)
32-bit register
32 plaintext blocks
128-bit plaintext block
Transpose
$b_0^1 \sim b_{31}^1$
$b_{32}^1 \sim b_{63}^1$
$b_{64}^1 \sim b_{95}^1$
$b_{96}^1 \sim b_{127}^1$
$b_0^{32} \sim b_{31}^{32}$
$b_{32}^{32} \sim b_{63}^{32}$
$b_{64}^{32} \sim b_{95}^{32}$
$b_{96}^{32} \sim b_{127}^{32}$
$b_0^1 \sim b_0^{32}$
$b_1^1 \sim b_1^{32}$
$b_2^1 \sim b_2^{32}$
$b_{127}^1 \sim b_{127}^{32}$

Fig. 1. Transposition in bitsliced AES implementation. On the left side, it shows the normal byte ordering of 32 128-bit plaintext block layout in registers. On the right side, it is the bit-oriented transposed bitsliced ordering layout in registers. The transformation from the left to the right and the reverse are made only once at the beginning of encryption and at the end of encryption, respectively.

Definition 3. *For bitsliced crypto implementations, if we give the same plaintexts for the consecutive M slices of totally M×N slices, it is called **N-Slice (encryption) mode**.*

Notations.

$I_{n,m}^{(i,j)}$:	denotes the *m*-th bit from the least significant bit in the *n*-th byte of the first S-box outputs. (i, j) specifies the j-th slice of the i-th thread.
$P_{n,m}^{(i,j)}$	denotes the *m*-th bit from the least significant bit in the *n*-th byte of a plaintext. (i, j) specifies the j-th slice of the i-th thread.
'$\cdot$','$\star$':	denote quantifier *all* and *any*, respectively. For example, $P_{n,\cdot}^{(i,j)}$ denotes one byte plaintext, and $P_{\cdot,\cdot}^{(i,j)}$ denotes 16-byte plaintext, and $P_{n,\star}^{(i,j)}$ denotes any bit of the n-th byte plaintext.

III. Leakage Acquisition and Preprocessing

Electro-magnetic emanation around electronic devices can be captured without any difficulties, but it is not so easy to measure useful signals. Compared with power analysis, EM analysis enables us to take advantage of localization effects, which makes EM attacks more efficient than power attacks. we use two small magnetic probes *Rohde Schwarz RF B 3-3* and *Rohde Schwarz RS H 2.5-2* instead of larger ones in order to probe localized leakages from near-field emanation [8]. Theoretically, the region located less than $1/2\pi$ of wavelength away from the source is called *near-field*. All our probings in this work are conducted in this region.

Experimental Set-ups. We set up our testbed with the following configurations:

- We target a NVIDIA's GeForce GT 620 graphics card connected to the host with PCI-e bus. The device is of low performance, but it is enough to show the vulnerability of NVIDIA's GPU to EM attacks. Specifically, it has one streaming multiprocessor of 48 SPs, a L2 cache of 64KiB, and it is equipped with an off-chip device memory of 454MiB. The device is running at 1.27GiHz.
- We port a bitsliced AES implementation from a open source community [9] into our GPU. Since this is a table-free implementation, we do not need to consider the efficiency of table look-up with respect to memory usage among different memories. The device memory in our GPU is used to store the plaintexts to be encrypted as well as the ciphertexts to be produced.
- We employ an Agilent DSO9104A digital oscilloscope, which is capable of measuring signals with a sample rate up to 20GHz (20GSa/s). In our experiment, we set our sampling rate as 200MSa/s, which turns out to be enough for our attack.

Our testbed is set up with a client/server mode, which is widely used in internet applications. Cloud device that provides SECaaS work as a server, and inside attackers is authorized to encrypt any plaintexts P-s and obtain the corresponding ciphertexts C-s and measured EM traces T-s. With a sufficient number of *triple* $\langle P, C, T\rangle$, the attacker attempt to recover the preset secret key of our AES implementation.

Locate Signals. A printed circuit board (PCB) like GPU card is generally composed of hundreds of electronic parts and components such as chips, resistors, capacitors, inductors and so on. However, it is unnecessary to check all of them to locate target signals. Generally speaking, only the right above of GPU chip and capacitors on the back of GPU chip should be checked, because these positions or components tend to produce useful leakages, which is confirmed afterwards in our experiment. More specifically, we start up the CUDA program and run encryption in a loop. We adjust EM probe on the candidate components within their near-field zones until we find a position in which the oscilloscope captures a periodic signal. If some pattern within the signal repeats nine to ten times, leakage positions are found. The repeated signal in our experiment is shown in Fig.2. We call it *target signal*.

Collect Signals. Although the target signal is identified, it is still not easy to capture it without external triggers. In fact, it is impractical to provide an external trigger that controlled within program, so we design a delicate trigger with another magnetic probe. As shown in Fig.2, two signals measured at different probing positions look similar, and the amplitude in the upper one is basically less than that in the lower one. However, the two signals share a signal pattern of the same high voltage marked as *Trigger A* and *Trigger B*, so the more significant difference between *Trigger* and other signals in the upper channel makes *Trigger A* a better choice to work as a trigger to capture target signal.

Align Signals. Now we have measured almost aligned EM traces with our delicate trigger, but it is still not enough to perform a successful attack. More accurate alignment techniques are necessary. By zooming in the first round encryption of the lower signal in Fig.2, more details of the first round encryption are shown in Fig.3. First, we observe the special patterns on the

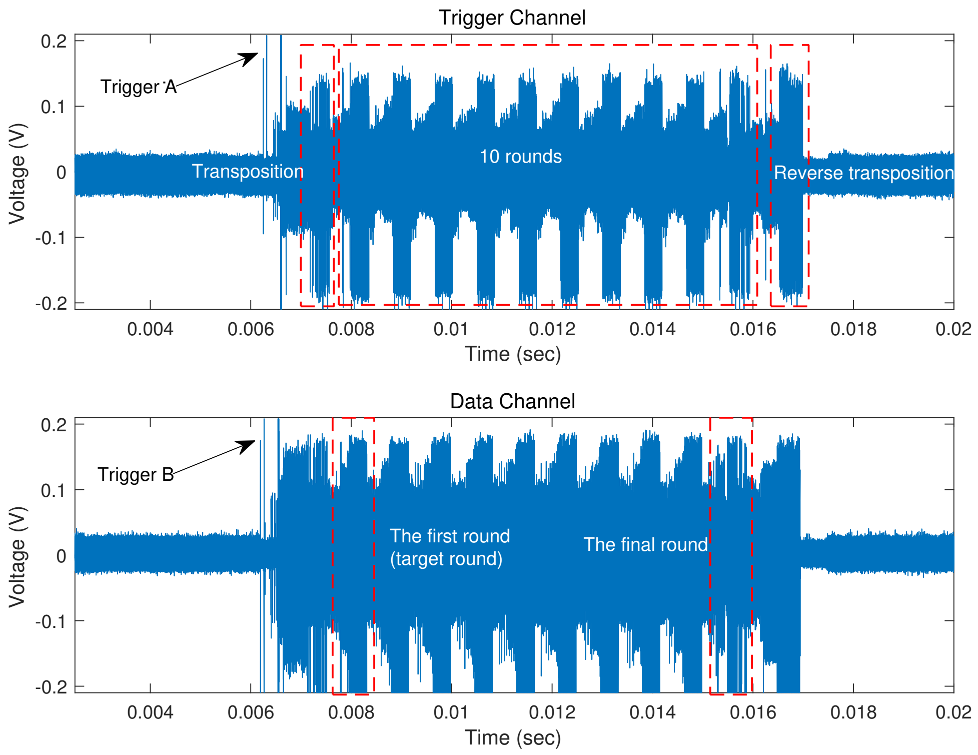

Fig. 2. Overview measurement of our GPU-based bitsliced AES implementation.

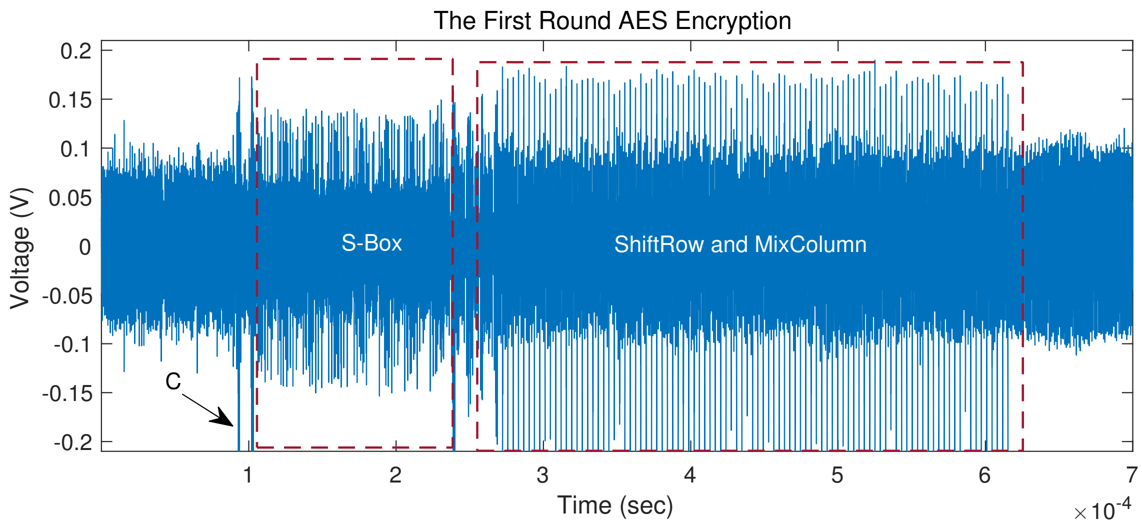

Fig. 3. Overview measurement of the first round AES encryption.

signal and find a *two-trough* (*C* in Fig.3) pattern that is shared by all traces. The pattern is likely an ideal reference to align all traces. Second, we match the pattern among several traces and find that the pattern in different traces are strongly correlated (Pearson Correlation Coefficient, PCC > 0.70). Third, for all traces we search the pattern by fixing one trace and sliding the others within a small range to find the position at which the pattern hold the maximum PCC with the pattern in the fixed trace. We exclude traces whose maximum PCC is less than 0.70. Then all traces with the maximum PCC no less than 0.70 could be aligned properly.

IV. The Security of Bit-Level Parallelism

The GPU-based bitsliced AES implementation achieves parallelism in two granularity, which are bit-level parallelism and thread-level parallelism. For bit-level parallelism, a single thread will suffice, so it is unnecessary to analyze its security with more than one group CTSCA. Hence, only one-group CTSCA is performed in our research.

A. *Non-Profiled Leakage Detection Test*

Since bits of the same position in multiple plaintexts are operated in the same register, bit-level leakages are very likely to happen. The Welch's t-test [10] is employed to detect the bit-level leakages induced by bitsliced implementation. Non-profiled leakage detection test needs specifying intermediates. For our implementation, we detect the independent bits of the output of S-box in the first round encryption. For simplicity, we feed 32 slices with the same plaintext, so all 128 registers for intermediates hold either 32 0's or 32 1's. Therefore, we just detect the difference of leakages in these two cases.

Test Method. First, N EM traces are partitioned into two groups G_0 and G_1 with respect to the intermediates $I = 0$ or $I = 2^{32} - 1$. Then, the following statistic is computed:

$$t(\tau) = \frac{|\mu_0(\tau) - \mu_1(\tau)|}{\sqrt{\frac{s_0^2(\tau)}{n_0} + \frac{s_1^2(\tau)}{n_1}}} \tag{1}$$

where $\mu_0(\tau)$, $\mu_1(\tau)$ are the means of G_0, G_1 at τ in time, and $s_0(\tau)$, $s_1(\tau)$ are the standard deviations of G_0, G_1 at τ in time, and n_0, n_1 are the cardinality of G_0, G_1. Although τ represents any value in the time domain, it takes discrete values due to finite sampling rate (e.g. 200MSa/s in our experiment) in practice. Last, it is time to determine whether two sets G_0 and G_1 are sampled from an identical population or not. Generally speaking, two sets are assumed to be sampled from two distinct populations, if the statistical quantity $t(\tau) > 4.5$ at some τ-s [11]. We follow this convention in our test experiments.

Test Results and Discussion. As noted above, there are totally 128 registers to hold the 128-bit *state* of bitsliced AES encryption. Now that the 128-bit *state* are leaked independently in the time domain, our experiment tests the leakage of each of the 128 bits by Welch's t-test method. As shown in Fig.4, 16 peaks or troughs are clearly visible, and they are far beyond the preset thresholds (-4.5 to 4.5). The 16 peaks/troughs of different colors represent the leakages of 16 S-box output in the first round encryption respectively. What is particular is the 12 peaks/troughs of extremely high/low t-statistic. According to the colors shown, the 12 peaks/troughs are created by four of totally 128 bits intermediate, precisely, the MSB of the 1st, 5th, 9th and 13th intermediate byte, respectively.

We also investigate the maximum of $t(\tau)$ in the time domain for each of 128 bits. For the sake of clarity, we make small changes in computing the maximum of $t(\tau)$ as follows:

$$t_{max} = \begin{cases} \max_\tau |t(\tau)|, & \text{if } \max_\tau |t(\tau)| > 4.5 \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The t_{max} of 128 intermediate bits are shown in Fig.5. It can be regarded as a 8×16 matrix $\Gamma(x, y)$, where $x \in \{1, 2, ..., 8\}$ and $y \in \{1, 2, ..., 16\}$. The eight rows show different patterns. For example, the 3th bit, 5th bit and 7th bit are detected to be leaky on every other bytes, and the MSB is tested to be leaky on every bytes. Those interesting patterns are of great help to recover the secret key, and they will be used to develop side-channel attacks against our target implementation.

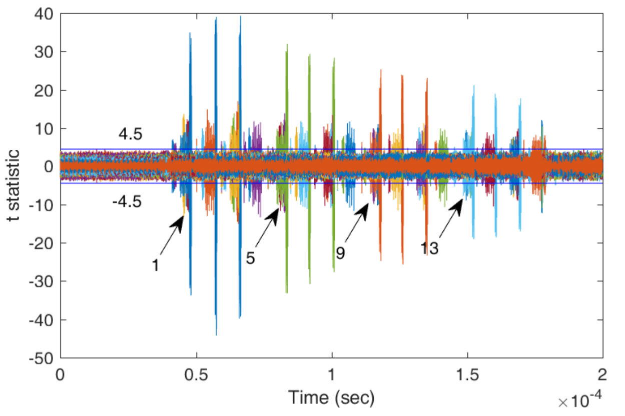

Fig. 4. Results of t-test for all 128 independent bits of the S-box output.

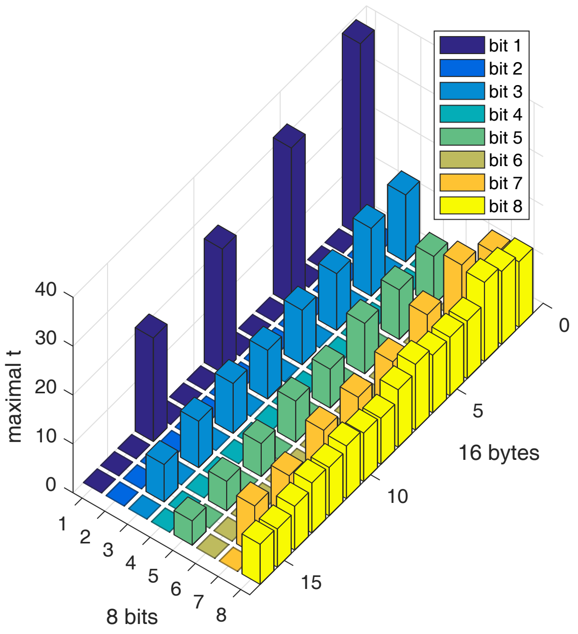

Fig. 5. Maximum of $t(\tau)$ in the time domain for all 128 individual bits of the S-box output.

B. *A Simple Single-Bit Correlation Analysis*

Correlation analysis [12] method is commonly used in side-channel attacks, which can be used to extract the correct secret key of our special AES implementation. For our bitsliced implementation, each intermediate (S-box output) is leaked in eight registers, so either of these eight values is qualified to recover the secret key with Correlation Electro-Magnetic Analysis (CEMA) method. Unfortunately, the leakages of the eight registers vary from byte to byte. We learn from above tests that only the most significant bit (MSB) of each intermediate byte is likely to be used to recover all 16-byte secret key. The leakage model is as follows:

$$L_{n,m} = a \cdot \sum_{j=1}^{32} I_{n,m}^{(1,j)} + B_{noise} \tag{3}$$

where $L_{n,m}$ denotes the predicted leakage of the m-th bit from the least significant bit in the n-th intermediate byte, and $\mathtt{HW}(\cdot)$ evaluates the Hamming weight of a variable. a is a scale factor, the value of which is insignificant for our attack. B_{noise} is a Gaussian variable, which might frustrate our attack if a high-level noise makes the signal-to-noise ratio drop significantly to some thresholds.

C. Multi-Bits Feature-level Fusion Attacks (MBFFA)

As mentioned above, only the MSB of $I_{n,\cdot}^{i,j}$ (namely $I_{n,8}^{i,j}$) can be used to recover 16-byte secret key, but it does not mean that the leakages on other bits are useless. Just on the contrary, appropriate multi-bits fusion techniques may improve the above attacks.

We propose a multi-bits feature-level fusion attack. Leakage features $\Gamma(x,y)$ are extracted from appropriate leakage detection methods such as Welch's t-test used above. Specifically, we build the following leakage model:

$$L_n = a \cdot \sum_{y=1}^{16} \left(Z(n,y) \cdot \sum_{j=1}^{32} I_{n,F_n}^{(1,j)} \right) + B_{noise} \tag{4}$$

where $n \in \{1,2,...,16\}$, and L_n is the predicted leakage of the n-th intermediate byte, and $Z(\cdot,\cdot)$ is defined as:

$$Z(x,y) = \begin{cases} 1, & \text{if } x = y, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

$F_n \in \{1,2,...,8\}$ for any n, and

$$F_n = \underset{j}{argmax} \left(\max_{j=1}^{8} \Gamma(j,n) \right), \tag{6}$$

where $\Gamma(j,n)$ is called *individual features*, and F_n is called *fused features* or *features*.

D. Multi-Bits Decision-level Fusion Attacks (MBDFA)

Just like MBFFA, multi-bits decision-level fusion attack also need to extract leakage features $\Gamma(x,y)$ from an appropriate leakage detection test before attacking. $\Gamma(x,y)$ tells on which bit to mount a attack before decision-level fusions. The leakage model is the as follows:

$$L_{n,i} = a \cdot \sum_{y=1}^{16} \left(Z(n,y) \cdot \sum_{j=1}^{32} I_{n,\Upsilon_n(i)}^{(1,j)} \right) + B_{noise} \tag{7}$$

where $\Upsilon_n = \{x | x \in \Gamma(\cdot,n) \text{ and } x \neq 0\}$ for $n \in \{1,2,...,16\}$, and $\Upsilon_n(i)$ denotes the i-th element of Υ_n. For the n-th secret key byte, multiple predicted leakages $L_{n,1}$, $L_{n,2}$, ... ,$L_{n,|\Upsilon_n|}$ are calculated. $L_{n,i}$ implicitly depends on the secret key byte k, so the secret key bytes are recovered with decision-level fusion as follows:

$$C(n,i) = \max_{k=0}^{255} \rho(L_{n,i}, E) \tag{8}$$

$$K_n = \underset{k}{argmax} \left(\max_{i=1}^{|\Upsilon_n|} C(n,i) \right) \tag{9}$$

where $\rho(\cdot,\cdot)$ evaluates the Pearson correlation coefficient of two vectors, and E is the measured EM leakage. Obviously, K_n is recovered by decision-level fusion technique. More details about the MBDFA is shown in Algorithm 1.

Algorithm 1 Multi-Bits Decision-level Fusion Attack

```
Input:
$P = [P_1, P_2, ..., P_{16}]$
$T = [T_1, T_2, ..., T_M]$ are EM traces of $M$ sampling points.
$\Upsilon = [\Upsilon_1, \Upsilon_2, ..., \Upsilon_{16}]$
Output:
$K = [k_1, k_2, ..., k_{16}]$: 16-byte secret key recovered.
1: for $l \leftarrow 1$ to 16 do
2:     for $k_{\mathtt{guess}} \leftarrow 0$ to 255 do
3:         $R \leftarrow \mathtt{SBox}\left(P_l \oplus k_{\mathtt{guess}}\right)$
4:         $S = \mathtt{HW}_{row}(R)$
5:         for $\beta \leftarrow 1$ to $|\Upsilon_l|$ do
6:             $b \leftarrow \Upsilon_l(\beta)$
7:             for $m \leftarrow 1$ to $M$ do
8:                 $\phi_{m,\beta} \leftarrow \rho(T_m, S_b)$
9:             $\Phi_{k_{\mathtt{guess}},\beta} \leftarrow \max\{\phi_{1,\beta}, \phi_{2,\beta}, ..., \phi_{M,\beta}\}$
10:        $W_\beta \leftarrow \max\{\Phi_{0,\beta}, \Phi_{1,\beta}, ..., \Phi_{255,\beta}\}$
11:    $W_{\mathtt{max}} \leftarrow \max\{W_1, W_2, ..., W_{|\Upsilon_l|}\}$
12:    $k_l \leftarrow \underset{k}{argmax}(W_{\mathtt{max}})$
13: return $K = [k_1, k_2, ..., k_{16}]$
```

E. Experimental Results and Discussion

Since the number of slices and groups for plaintexts have nothing to do with the bit parallelism of our target implementation, our experiments are performed in one-slice and one-group scenario, which means just one plaintext is needed for one EM trace measurement. As shown in Fig.3, the performance of single-bit CEMA varies from bit to bit. CEMA on the most significant bit outperforms that on other bits, which is consistent with the results of leakage detection test. We also investigate the performance of our proposed methods MBFFA and MBDFA. The experimental results show they are almost equivalent and more efficient than the single-bit CEMA on MSB. Just 400 EM traces is enough to mount a complete key recovery attack against our target AES implementation.

We have to note two facts. *First*, the MBFFA or MBDFA cannot work if any prior leakage detection test is not available, because both methods are based on the prior knowledge of leakages on 128 bits, say $I_{n,m}^{*,*}$, where $n = \{1,2,...,16\}$ and

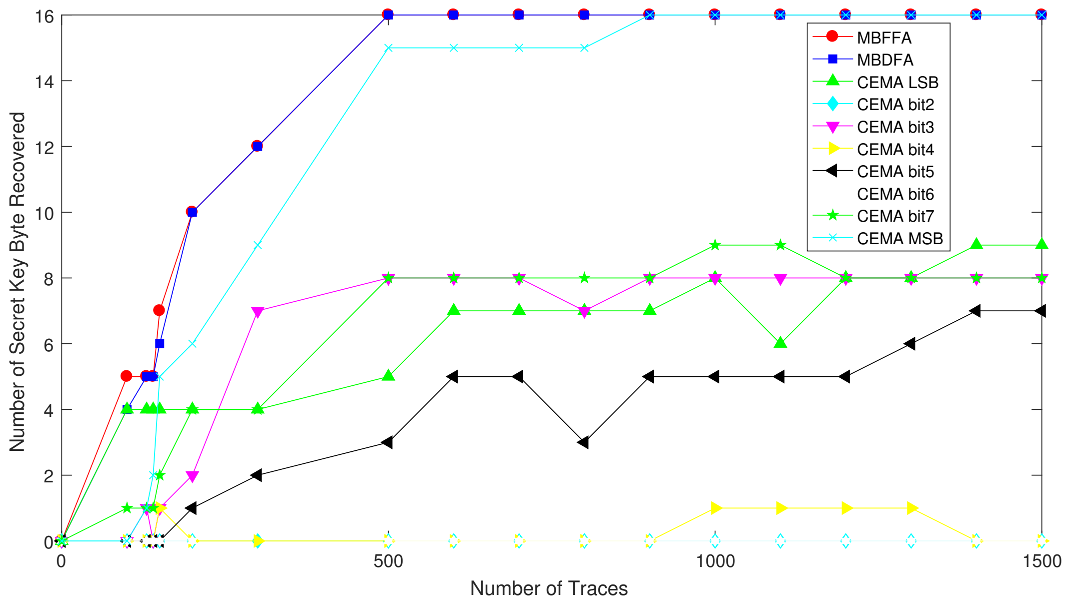

Fig. 6. Experimental results of MBFFA and MBDFA.

$m = \{1, 2, ..., 8\}$. In this sense, it should be classified as profiled attacks like template attacks. However, our methods may be more practical than template attacks, because profiling is done only once in our methods for devices of the same model before attacking while profiling and attacking in template attacks must be on identical devices. *Second*, the MBFFA and MBDFA are not much efficient, which is induced by different leakages on 8 bits. Thus, leakages on some bits dominate leakages on all bits, so it is reasonable that the non-dominant leakages will not help much to the dominate leakage when all the leakages are fused with some techniques. As a matter of fact, if leakages on all bits were the same or almost the same, the profiling stage mentioned above would not be necessary, either.

V. The Security of Thread-Level Parallelism

In this section, the side-channel security of thread-level parallelism of GPU-based bitsliced AES implementation towards electro-magnetic attacks will be deeply investigated. The thread-level parallelism on CUDA-enabled GPUs features the SIMT (Single Instruction, Multiple Threads) execution fashion from which programs with conditional branches would benefit. For cryptographic implementations like ours without any conditional branches, the program executes in a strict parallelism fashion, which means that threads in a GPU warp run at the same pace without any time deviation. As a result, much noises are introduced when multiple threads run on different inputs, which basically makes the side-channel attacks much difficult. However, how thread-level parallelism affects the security of strict parallel cryptographic implementation remains unknown, so further studies are needed.

A. Developing a Simple Attack

The parallel execution of many threads on GPU makes the EM emanation much complicated. Since the warp scheduling within GPU is primarily unpredictable, we just consider thread-level parallelism within one GPU warp. In this case, multiple threads execute our target encryption simultaneously. As mentioned above, multiple threads in a warp run in a strict parallelism fashion because there are not any cache accesses or conditional branches in our target program that make a divergence among threads. Therefore, the most direct way to model the simultaneous EM leakages of multiple threads in a warp is to add up all their respective leakages together, although it may not be the best one.

Intermediate Bit Selection. As our target implementation processes any byte of AES *state* bit by bit, we formulate the process of `AddRoundKey` followed by `SubByte` of any byte in bit operations:

$$[I_{n,1}^{(i,j)}, I_{n,2}^{(i,j)}, ..., I_{n,8}^{(i,j)}] \leftarrow F\left([P_{n,1}^{(i,j)}, P_{n,2}^{(i,j)}, ..., P_{n,8}^{(i,j)}] \oplus K_n\right) \tag{10}$$

where $\oplus$ is component-wise `XOR`, and $F(\cdot)$ is a series of `XOR` and `AND` operations to compute the multiplicative inversion of the 8-bit $P_{n,\cdot}^{(i,j)} \oplus K_n$ over $\mathbb{F}_{2^8}$. $I_{n,\cdot}^{(i,j)}$ is called *intermediate*, which is a key-dependent quantity. $I_{\star,m}^{(i,j)}$ is the m-th bit of any intermediate.

As mentioned in the previous section, $I_{n,1}^{(i,j)}, I_{n,2}^{(i,j)}, ..., I_{n,8}^{(i,j)}$ are scattered across eight independent registers. However, the general fusion methods are always defeated due to the different noise-level among bits or bytes, so only some special fusion methods with prior knowledge work. Since we just consider the thread-level parallelism, we would better use the MSBs of 16 intermediate bytes as the target bit in order to avoid multi-bits fusions.

Simple Synchronous Model (SSM). The most direct way to describe the simultaneous leakages in a warp is to sum them up as follows:

$$L_{n,m} = a \cdot \sum_{i=1}^{32} \mathtt{HW}\left(I_{n,m}^{(i,\cdot)}\right) + B_{noise}, \tag{11}$$

where $n \in \{1,2,...,16\}$, $m \in \{1,2,...,8\}$, $L_{n,m}$ evaluates the predicted leakages, and $I_{n,m}^{(i,\cdot)}$ is of 32-bit length stored in a single 32-bit register of processor. For the n-th key byte, we have eight leakage points at least. Those leakages happen at different point-in-time in the time domain.

Performance of SSM and Discussions. In order to evaluate the performance of the hypothetical model SSM we set up experiments of N-group CTSCA in one-slice mode. Since N can possibly be assigned any value exactly dividable to 32 (warp size in our setting), we choose three typical value of *N=2,8,32* in our experiments. As shown in Fig.7, all 16 secret key byte can be recovered with about 1,500 EM traces in 2-group CTSCA, and none of secret key bytes is recovered with up to 4,000 EM traces in 8-group or 32-group CTSCA. The experimental results also show that more groups make the attack harder due to more noises.

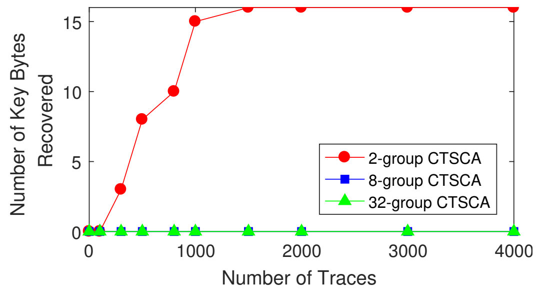

Fig. 7. Attack results of N-group CTSCA in one-slice mode with SSM.

It seems that the above model SSM works when dealing with simultaneous leakages of strict parallel implementation. However, it is just a heuristic model, so more accurate models will be further studied.

B. *Profiled Correlation-based Leakage Detection Test*

To further understand the nature of parallel leakages, we employ profiled leakage detection method to analyze the individual leakages of multiple threads. With profiled methods, we do not need to make any assumptions about the leakage model of the target implementation, which thereby lowers the requirement and simplifies the procedure. The profiled ρ-test method we use is originally due to Durvaux and Standaert [13]. The method takes advantage of the cross-validation techniques introduced in [14] and applies to the leakages of all threads in a warp. For the test of each thread, the leakages from any other threads are treated as random noises. Specifically, our test is carried out in three steps:

First, N EM traces with random plaintext inputs are sampled. For k-fold cross-validation, the set of acquired traces $\mathcal{L}$ is split into k (we set $k = 10$) non-overlapping subsets $\mathcal{L}^{(1)}, \mathcal{L}^{(2)}, ..., \mathcal{L}^{(k)}$ of (approximately) the same size. For $i = 1,2,...,k$, we define the profiling sets $\mathcal{L}_p^{(j)} = \bigcup_{i \neq j} \mathcal{L}^{(i)}$ and the test sets $\mathcal{L}_t^{(j)} = \mathcal{L} \backslash \mathcal{L}_p^{(j)}$. For each target plaintext byte variable X_m with $m \in \{1,2,...,32\}$ and for each cross-validation set j with $j \in \{1,2,...,k\}$, a model is estimated: $\hat{model}_\tau^{(j,m)}(X_m) \leftarrow \mathcal{L}_p^{(j,m)}$. For 8-bit plaintext bytes, this model corresponds to the sample means of the leakage sample τ corresponding to each value of the plaintext bytes.

Next, we compute the Pearson correlation coefficient between this model and the leakage sample in the test sets $\mathcal{L}_t^{(j,m)}$:

$$\hat{r_m}(\tau) = \frac{1}{k} \cdot \sum_{j=1}^{k} corr(\mathcal{L}_X^{(j,m)}(\tau), \hat{model}_\tau^{(j,m)}(X_m)) \tag{12}$$

where $m \in \{1,2,...,32\}$.

Last, the ρ-statistic of standard normal distribution is evaluated:

$$\hat{\rho_m}(\tau) = \frac{1}{2} \cdot ln\left(\frac{1+\hat{r_m}(\tau)}{1-\hat{r_m}(\tau)}\right) \cdot \sqrt{\frac{N}{k} - 3} \tag{13}$$

where N is the number of EM traces. Since $\hat{\rho_m}(\tau)$ satisfies standard normal distribution at any time τ, $|\hat{\rho_m}(\tau)| > 4.5$ can conclude the existence of leakage at τ with a probability of larger than 0.99.

In our experiment, the ρ-statistics of 32 individual tests at each time τ are evaluated and plotted within a single figure as Fig.8. It shows that there are approximately five leakage points marked as LP1, LP2, LP3, LP4 and LP5, respectively. At any of the five leakage points, multiple colors are accumulated, which seems that 32 threads leak information simultaneously. However, it is not like this when zooming in any of the five leakage points. As shown in Fig.9, the details of LP1, LP2, LP4 and LP5 tell that not all leakages from multiple threads are synchronous as we usually think. This discovery is so important because we always expect synchronous executions to generate synchronous leakages instead of asynchronous ones. It is obvious that the executions of Td1 and Td2 are almost overlapping, the same with Td17, Td18 and Td19. Since we cannot deny the existence of leakages when lacking indications in the figure, we still do not know whether they are leaky or not in other threads except Td1, Td2, Td17, Td18 and Td19. We do not know why the device leaks information in these special threads, and we think it may have something to do with the half-warp feature of CUDA-enabled GPU.

C. *A General Model and Hybrid Fusion Attack*

For parallel computing platforms, the above model SSM is effective only if the leakages from multiple computing units are absolutely synchronous in the time domain. It is widely believed that multiple threads execution for a warp on GPU always produces synchronous EM leakages because of the strict parallel execution fashion. However, our experimental

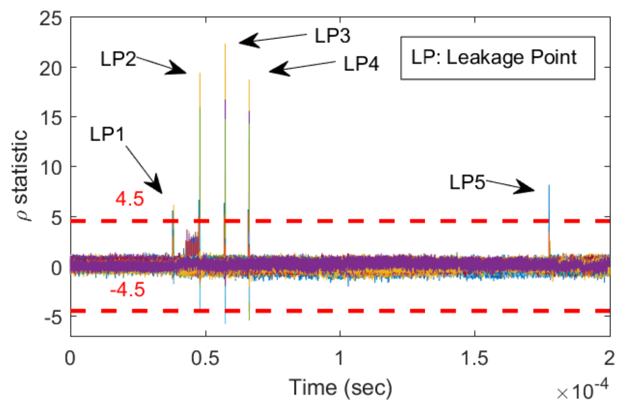

Fig. 8. Experimental results of ρ-test for 32 threads in 32-group mode.

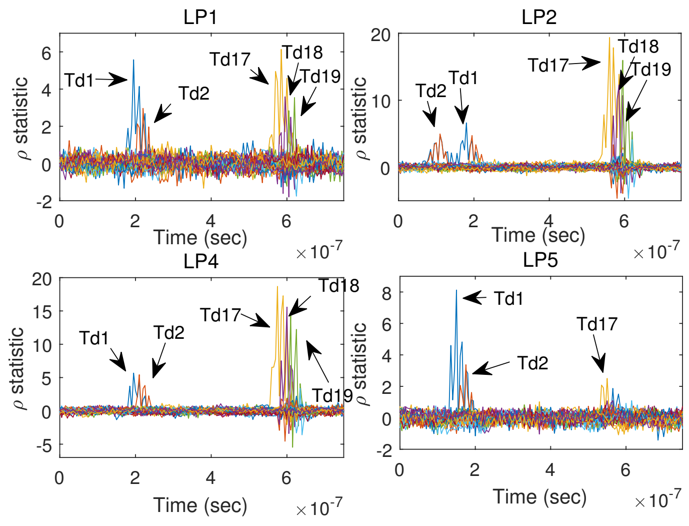

Fig. 9. Experimental results when zooming in LP1, LP2, LP4 and LP5.

result shows that all respective EM leakages for a GPU warp in our setting are not rightly overlapping. Therefore, it is reasonable that the above model SSM is not so effective, so we need to develop more effective methods.

Warp Asynchronous Leakage (WAL). As mentioned above, although all threads in a GPU warp execute in a strict parallelism fashion, they may not produce synchronous EM leakages. Suppose that $\mathcal{E}_i(t)$ denotes the quantity of EM emanation of the i-th thread and τ_i denotes the time offset of the i-th thread, and the quantity of multiple EM emanation is formulated as:

$$\mathcal{E}(t) = \sum_{i=1}^{32} \mathcal{E}_i(t-\tau_i) = \mathcal{E}_m(t-\tau_m) + \sum_{i\in D} \mathcal{E}_i(t-\tau_i) \quad (14)$$

where N_{td} denotes the total number of threads and $n \in [1,32] \cap \mathbb{Z}$ and $D = ([1,32] \cap \mathbb{Z}) \backslash \{m\}$. If $\mathcal{E}_i(t)$ at any moment t_0 is statistically independent of $\mathcal{E}_i(t)$ at any other moment except t_0 and time offset τ_i-s are pairwisely different, then it is assumed that multiple threads execution on GPU produces simultaneous EM leakages, and every single thread contributes to the superimposed EM leakage. In fact, a warp of threads start at the same time, but it do not mean that their respective leakages are exactly overlapping.

General Asynchronous Model (GAM). Based on the analysis of WAL above, the EM emanation of one thread is modeled as:

$$E_i(t) = h(t-\tau_i) \times H_i + b_i(t) \quad (15)$$

where $1 \le t \le T$ is the sampling point, $1 \le i \le 32$ is the index of thread, H_i is a leakage function of the intermediate, $h(t)$ is the the power of EM emanation of a thread with unit intermediate starting at time point ”0”. τ_i is the delay time of the i-th thread relative with time ”0”. Then the general asynchronous model can be formulated as:

$$E(t) = \sum_{i=1}^{32} E_i(t) = \sum_{i=1}^{32} h(t-\tau_i) \cdot H_i + B(t) \quad (16)$$

where $B(t) = b_1(t) + b_2(t) + ... + b_{32}(t)$. When we just focus on the j-th threads and treat the EM emanation from other threads as noises, the model can be rewritten as:

$$E(t) = E_j(t) + \sum_{i\in J\backslash\{j\}} E_i(t) = h(t-\tau_j) \cdot H_j + B'(t) \quad (17)$$

where $B'(t) = \left(B(t) + \sum_{i\in J\backslash\{j\}} E_i(t)\right)$ is treated as noises. Interestingly, although Equ.16 and Equ.17 are essentially equivalent, different approaches can be derived from the two equations. To be specific, Equ.16 tells that a data-level fusion technique may be effective, while Equ.17 means a decision-level/feature-level fusion attack (DFA/FFA) may be possible. The latter method requires less computation than the former one, so we just develop an attack based on Equ.17.

Multi-Threads Hybrid Fusion Attack (MTHFA). It has been proven experimentally that the EM leakages from different GPU threads in a warp is not exactly synchronous. The asynchronization among threads makes the leakages of the same instruction may occur at different time. We propose a MTHFA method, which is a combination of multi-threads feature-level fusion attack (MTFFA) and multi-threads decision-level fusion attack (MTDFA). To be specific, the MTFFA is applied among synchronous threads, and the groups of asynchronous threads is fused with MTDFA. The grouping $\Lambda = \{\Lambda_1, \Lambda_2, \Lambda_3, ...\}$ of threads is extracted from the above profiled leakage detection test, where $\Lambda_\iota \subset \{1, 2, ..., 32\}$ for any ι, and for any $\iota_1 \neq \iota_2$, $\Lambda_{\iota_1} \cap \Lambda_{\iota_2} = \emptyset$. In practice, synchronous threads cannot be detected due to errors in profiled leakage detection test, so two thread are considered to be synchronous if the distance of their POIs (Point of Interests) are within a very small threshold ϵ, that is $|t_1 - t_2| < \epsilon$. Specifically, we achieve MTFFA in the following model:

$$L_{n,\iota} = a \cdot \sum_{i\in\Lambda_\iota} \left(\sum_{j=1}^{32} I_{n,m}^{(i,j)} \right) + B_{noise} \quad (18)$$

Algorithm 2 Multi-Threads Hybrid Fusion Attack (MTHFA)

Input:
$P^1 = [P^1_1, P^1_2, ..., P^1_{16}]$, $P^2 = [P^2_1, P^2_2, ..., P^2_{16}]$, ... , $P^{32} = [P^{32}_1, P^{32}_2, ..., P^{32}_{16}]$ are plaintexts encrypted on 32 threads in a GPU warp,
$T = [T_1, T_2, ..., T_M]$ are EM traces of M sampling points.
$\Lambda = [\Lambda_1, \Lambda_2, ..., \Lambda_Z]$ are leakage patterns from leakage detection test, where $\Lambda_x \subset$\{1,2,...,32\} and $\bigcup_{x=1}^{Z} \Lambda_x = \{1, 2, ..., 32\}$
Output:
$K = [k_1, k_2, ..., k_{16}]$: 16-byte secret key recovered.

```
1: for $l \leftarrow 1$ to 16 do
2:     for $k_{\text{guess}} \leftarrow 0$ to 255 do
3:         $R^1 \leftarrow \texttt{SBox}\left(P^1_l \oplus k_{\text{guess}}\right)$
4:         $R^2 \leftarrow \texttt{SBox}\left(P^2_l \oplus k_{\text{guess}}\right)$
5:         ... ...
6:         $R^{32} \leftarrow \texttt{SBox}\left(P^{32}_l \oplus k_{\text{guess}}\right)$
7:         for $\alpha \leftarrow 1$ to $Z$ do
8:             $S^\alpha \leftarrow 0$
9:             for $\beta \in \Lambda_\alpha$ do
10:                $S^\alpha = S^\alpha + \texttt{HW}_{row}(\texttt{MSB}(R^\beta))$
11:        for $m \leftarrow 1$ to $M$ do
12:            $\phi_m \leftarrow 0$
13:            for $\alpha \leftarrow Z$ do
14:                $\phi_m \leftarrow \phi_m + \rho(T_m, S^\alpha)$
15:            $\phi_m \leftarrow \phi_m / Z$
16:        $W_{k_{\text{guess}}} \leftarrow \max\{\phi_1, \phi_2, ..., \phi_M\}$
17:    $W_{\texttt{max}} \leftarrow \max\{W_0, W_1, ..., W_{255}\}$
18:    $k_l \leftarrow \underset{k}{argmax}(W_{\texttt{max}})$
19: return $K = [k_1, k_2, ..., k_{16}]$
```

where $L_{n,\iota}$ is the predicted leakage of the n-th secret byte in threads Λ_ι. $m = 8$ in our study, which means only the MSB of intermediate byte is considered. After the feature-level fusion, multiple predicted leakages $L_{n,1}$, $L_{n,2}$, $L_{n,3}$, ... are calculated. Then, multiple CEMA are performed before making a decision-level fusion (MTDFA) as:

$$K_n = \underset{k}{argmax}\left(\max_{k=0}^{255} \frac{\sum_\iota \rho(L_{n,\iota}, E)}{|\Lambda|}\right) \tag{19}$$

where $K = \{K_1, K_2, ..., K_{16}\}$ is the secret key recovered with MTHFA. The "hybrid" in MTHFA means the method is a combination of two basic methods DFA and FFA.

D. Experimental Results and Discussion

Since our multi-threads fusion attack needs multiple threads encrypting different plaintexts, our experiments are performed in one-slice and multiple-group scenario. In our experiments, MTFFA, MTDFA and MTHFA are evaluated. We extract Λ by the profiled leakage detection test and find $\Lambda = \{\Lambda_1, \Lambda_2\}$, where $\Lambda_1 = \{1, 2\}$ and $\Lambda_2 = \{17, 18\}$. In this case, MTFFA can be applied between Td1 and Td2 or Td17 and Td18, and MTDFA can be applied between any one in Λ_1 and Λ_2. As shown in Fig.10, MTFFA, MTDFA and MTHFA outperform CEMA on single thread. Although both MTFFA and MTDFA are two-threads fusion attacks, MTFFA performs better than MTDFA. It tells that MTFFA can make better use of multi-threads leakages than MTDFA does. It is no doubt that MTHFA performs best. Just 2,000 EM traces suffice to recover 16-byte secret key of our target AES implementation.

We note that only five out of 32 threads are detected to be leaky from the profiled leakage detection test. In fact, all thread are leaky if using a non-profiled leakage detection method for a change, so further improvements are possible. In addition, Equ.16 implies a data-level fusion is also possible.

TABLE I
THE Θ OF OUR PROPOSED METHODS.

MBFFA	MBDFA	MTFFA	MTDFA	MTHFA
$\Theta = 1.7$	$\Theta = 1.7$	$\Theta = 1.5$	$\Theta = 2.0$	$\Theta = 2.0$

VI. The Assessment Criteria of the Security of Parallelism

Just as many studies show, parallelism does improve side-channel security of cryptographic implementations because of high-level noises. However, how to quantify the improvement is still an open question. In this section, we propose a very single metric Θ as a assessment criteria. The Θ is defined as:

$$\Theta = \frac{\gamma \cdot N_p}{\min\limits_{\iota} N_\iota} \tag{20}$$

where N_p denotes the number of traces for attacks with γ-way fusion and N_ι stands for the number of traces for attacks with the ι-th way leakage. For example, the MTFFA, MTDFA, MTHFA are 2-way, 2-way, 4-way fusions, respectively. Obviously, larger Θ means higher side-channel security of parallelism against certain attack. We assess the side-channel security of our target AES implementation against MBFFA, MBDFA, MTFFA, MTDFA and MTHFA, respectively. The assessment results are shown in Table 1. It tells that our target AES implementation achieves the highest security when only MTDFA or MTHFA is possible and the lowest security when only MTFFA is available. It seems that MTFFA is more efficient than MTHFA, which contradicts the above experimental results. In fact, it is reasonable because Θ takes γ into account. Θ describes the efficiency per way of a certain attack method, so it is a better criteria than conventional criteria without the number of ways for fusion, say Θ/γ.

VII. Related Work

A. GPU-based AES Implementation

An efficient option for AES software implementations on GPUs is the *T*-box approach due to Daemen and Rijmen [15]. The rationale behind the idea is to merge three (`SubBytes`, `ShiftRows`, `MixColumns`) of four AES transformations into four independent table lookups so as to make the best use of high efficient cached memory access on GPUs. Biagio *et al.* published a counter mode AES (AES-CTR) on an

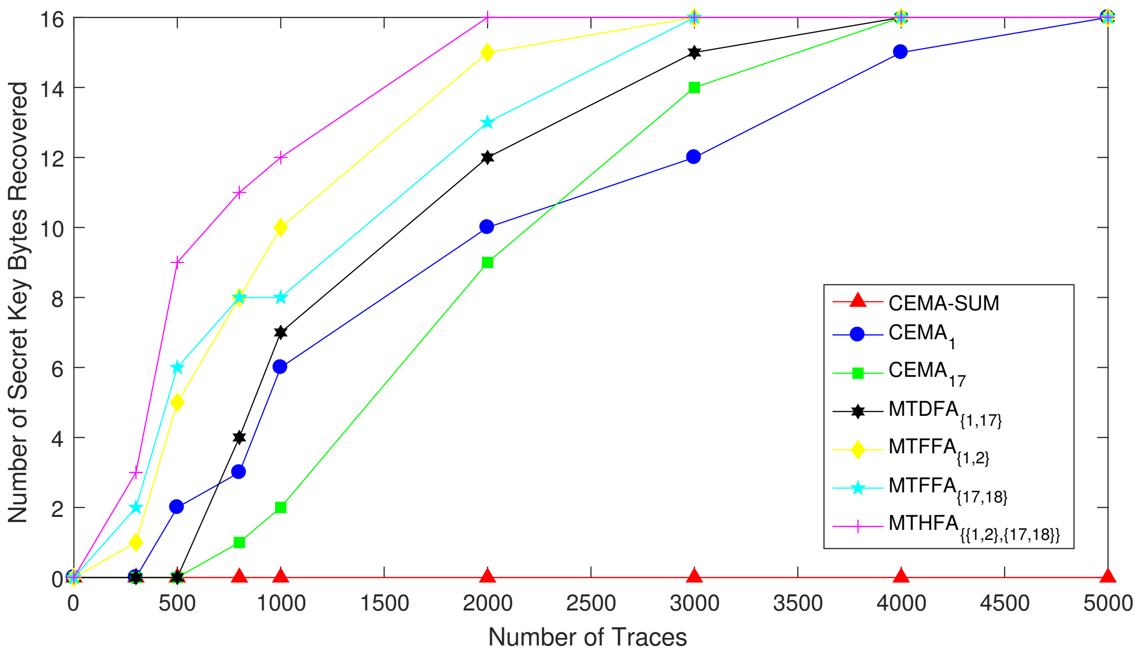

Fig. 10. Experimental results of MTFFA, MTDFA and MTHFA.

NVIDIA CUDA-enabled GPU [16]. They proposed a fine-grained solution exposing the internal parallelism of each AES round by dispatching four 32-bit words blocks to four SPs. Four T tables are loaded into shared memory in order to accelerate the T-boxes lookups. Because of address-dependent table accesses in table-based implementations, it is susceptible to cache attacks, especially, cache-timing attack. However, table-free implementations like bitsliced version could keep a constant execution time, so it becomes an implicit timing attack protection method with respect to masking protection. The first implementation of this type on GPUs is due to Lim, Petzold and Koç in [17]. Bitsliced implementations are not as competitive with word-level implementations on CPUs due to the limited number of registers and the cost of transpositions of the ciphertext.

B. SCA against GPU-based AES Implementation

Luo *et al.* proposed the first power analysis attack against GPU-based AES implementation in [5]. They inserted a resistor in series with power supply in order to measure the power consumption of GPU card. They targeted a T-box implementation of AES on GPU and built a simplified leakage model to avoid the synchronization of power traces in the time domain in multiple core scenarios. They employed Correlation Power Analysis (CPA) to recover 16-byte secret key of AES with 160,000 traces. Their attack is performed in a chosen-thread scenario, which requires the adversary be capable of encrypting the same plaintexts for all block threads. In fact, it is scarcely possible to conduct side-channel attacks successfully in known-plaintext and highly-occupied scenarios against GPU-based cryptographic implementations. After that, Jiang *et al.* proposed two cache-timing attacks against GPU-based T-table AES implementation based on the time differences induced by L1 cache line access serialization (CLAS) [3] and shared memory bank conflict (SMBC) [6]. They recovered 16-byte secret key of AES by Correlation Timing Analysis (CTA) and Differential Timing Analysis (DTA), respectively. Gao *at al.* proposed an electro-magnetic attacks against a GPU-based AES implementation based on the cache line access coalescing [4].

VIII. Conclusion and Discussion

In this work, for the first time we investigate the side-channel security of a GPU-based bitsliced AES implementation in order to give a deep insight into how strict parallelism affects security. We study the problem in terms of bit-level parallelism and thread-level parallelism and propose several methods, namely MBFFA, MBDFA and MTHFA, to analyze the side-channel security of the target implementation. We also propose a simple metric to assess the side-channel security of a certain parallel cryptographic implementation for certain attacks, which is very useful in security assessments.

It is obvious that this work is far from perfect. For bit-level parallelism, more researches are needed to find appropriate intermediates within bit-based S-box computation, by which can detect comparable leakages among registers containing the *state* of AES. In this way, the profiling stage of our attack may not be necessary. For thread-level parallelism, the reason for warp asynchronous leakage should be investigated and try to take more threads into account when mounting a fusion attack in order to improve efficiency.

For our future work, we will try to combine MTHFA and MBFFA/MBDFA together to achieve a further improvement of attack. In addition, the countermeasures against MBF-FA/MBDFA and MTHFA are also investigated.

References

[1] B. Di, J. Sun, and H. Chen, "A study of overflow vulnerabilities on gpus," in *Network and Parallel Computing - 13th IFIP WG 10.3 International Conference, NPC 2016, Xi'an, China, October 28-29, 2016, Proceedings*, 2016, pp. 103–115. [Online]. Available: https://doi.org/10.1007/978-3-319-47099-3_9

[2] H. N. Jouybari and N. B. Abu-Ghazaleh, "Covert channels on gpgpus," *Computer Architecture Letters*, vol. 16, no. 1, pp. 22–25, 2017. [Online]. Available: https://doi.org/10.1109/LCA.2016.2590549

[3] 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. [Online]. Available: https://doi.org/10.1109/HPCA.2016.7446081

[4] 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: http://doi.acm.org/10.1145/3195970.3196042

[5] 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. [Online]. Available: https://doi.org/10.1109/ICCD.2015.7357115

[6] Z. H. Jiang, Y. Fei, and D. R. Kaeli, "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. [Online]. Available: http://doi.acm.org/10.1145/3060403.3060462

[7] E. Biham, R. J. Anderson, and L. R. Knudsen, "AES proposal serpent," *AES CD-1: documentation*, 1998.

[8] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM side-channel(s)," in *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, 2002, pp. 29–45. [Online]. Available: https://doi.org/10.1007/3-540-36400-5_4

[9] C. Patrick, "Bitsliced AES implementation in C," https://github.com/conorpp/bitsliced-aes.

[10] J. G. Goodwill, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," in *NIST non-invasive attack testing workshop, 2011*, 2011.

[11] T. Schneider and A. Moradi, "Leakage assessment methodology - A clear roadmap for side-channel evaluations," in *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, 2015, pp. 495–513. [Online]. Available: https://doi.org/10.1007/978-3-662-48324-4_25

[12] 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

[13] F. Durvaux and F. Standaert, "From improved leakage detection to the detection of points of interests in leakage traces," in *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, 2016, pp. 240–262. [Online]. Available: https://doi.org/10.1007/978-3-662-49890-3_10

[14] F. Durvaux, F. Standaert, and N. Veyrat-Charvillon, "How to certify the leakage of a chip?" in *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, 2014, pp. 459–476. [Online]. Available: https://doi.org/10.1007/978-3-642-55220-5_26

[15] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*, ser. Information Security and Cryptography. Springer, 2002. [Online]. Available: https://doi.org/10.1007/978-3-662-04722-4

[16] A. D. Biagio, A. Barenghi, G. Agosta, and G. Pelosi, "Design of a parallel AES for graphics hardware using the CUDA framework," in *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy, May 23-29, 2009*, 2009, pp. 1–8. [Online]. Available: https://doi.org/10.1109/IPDPS.2009.5161242

[17] R. K. Lim, L. R. Petzold, and Ç. K. Koç, "Bitsliced high-performance AES-ECB on GPUs," in *The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*, 2016, pp. 125–133. [Online]. Available: https://doi.org/10.1007/978-3-662-49301-4_8