Gao *et al. Cybersecurity* (2020) 3:3
https://doi.org/s42400-020-0045-8

Cybersecurity

RESEARCH Open Access

Efficient electro-magnetic analysis of a GPU bitsliced AES implementation

Yiwen Gao[1,2], Yongbin Zhou[1,2*] and Wei Cheng[1]

Abstract

The advent of CUDA-enabled GPU makes it possible to provide cloud applications with high-performance data security services. Unfortunately, recent studies have shown that GPU-based applications are also susceptible to side-channel attacks. These published work studied the side-channel vulnerabilities of GPU-based AES implementations by taking the advantage of the cache sharing among multiple threads or high parallelism of GPUs. Therefore, for GPU-based bitsliced cryptographic implementations, which are immune to the cache-based attacks referred to above, only a power analysis method based on the high-parallelism of GPUs may be effective. However, the leakage model used in the power analysis is not efficient at all in practice. In light of this, we investigate electro-magnetic (EM) side-channel vulnerabilities of a GPU-based bitsliced AES implementation from the perspective of bit-level parallelism and thread-level parallelism in order to make the best of the localization effect of EM leakage with parallelism. Specifically, we propose efficient multi-bit and multi-thread combinational analysis techniques based on the intrinsic properties of bitsliced ciphers and the effect of multi-thread parallelism of GPUs, respectively. The experimental result shows that the proposed combinational analysis methods perform better than non-combinational and intuitive ones. Our research suggests that multi-thread leakages can be used to improve attacks if the multi-thread leakages are not synchronous in the time domain.

Keywords: GPU-based cryptographic implementations, Side-channel analysis (SCA), Electro-magnetic attacks (EMA), Micro-architectural vulnerabilities, Combinational analysis

Introduction

Nowadays as the most widely used parallel computing platform, Graphics Processing Unit (GPU) has evolved from a special hardware for graphics rendering into a general-purpose computing device for various applications as biomedical analysis, signal processing, scientific computing and so on. GPU executes program in a Single-Instruction, Multiple-Thread (SIMT) fashion, so it is well suited for cryptographic applications deployed in cloud computing environment to provide the Security-as-a-Service (SECaaS). Unfortunately, GPU-based applications are vulnerable to many known attacks as proposed in Di et al. (2016); Naghibijouybari et al. (2018); Jiang et al. (2016). Among those published vulnerabilities of GPUs, side-channel vulnerabilities are the most serious ones due to their non-invasiveness to target devices. In recent years, the study 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 has been widely deployed on a variety of hardware platforms. The side-channel attacks against CPU-based AES software implementations and FPGA-based hardware implementations have been deeply investigated. Until very recently, some literatures mentioned that GPU-based cryptographic implementations are also susceptible to side-channel attacks through electro-magnetic (EM) emanation (Gao et al. 2018; Gao et al. 2018), power consumption (Luo et al. 2015) or execution time leakages (Jiang et al. 2016; 2017). Thanks to the bitsliced cipher proposed by Biham Biham (1997) as well as the efficient GPU-based bitsliced AES implementations proposed by Lim et al. Lim et al. (2016) and Nishikawa et al. Nishikawa et al. (2017), we are capable of deploying GPU-based AES

*Correspondence: zhouyongbin@iie.ac.cn
[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

cryptosystems that are resistant to cache-timing attacks (Jiang et al. 2016; 2017) and cache-based EM attacks (Gao et al. 2018; Gao et al. 2018). However, it does not mean that GPU-based AES implementation is not vulnerable to other side-channel attacks, for example the power analysis attack proposed in (Luo et al. 2015), though it is not efficient at all in practice. In light of this, we study more efficient side-channel attacks against a GPU-based bitsliced AES implementation in order to give a deeper insight into the side-channel vulnerabilities of GPU-based cryptographic implementations in the perspective of parallelism.

Related work

Luo et al. proposed the first power analysis attack against a GPU-based AES implementation in (Luo et al. 2015). They inserted a resistor in series with power supply in order to measure the power consumption of GPU card. They targeted a T-table-based GPU AES implementation and built a simplified leakage model to avoid the synchronization of power traces in the time domain. They employed correlation power analysis (CPA) to recover 16-byte secret key of AES with 160,000 power traces. Their attack is performed in a chosen-thread mode, which requires the adversary be capable of encrypting the same plaintexts for all block threads. In fact, it is almost impossible to conduct side-channel attacks successfully in known-plaintext or highly-occupied scenarios against GPU-based cryptographic implementations. After that Jiang et al. proposed two cache-based timing attacks against T-table-based GPU AES implementation based on the time differences induced by L1 cache line access serialization (Jiang et al. 2016) and shared memory bank conflict (Jiang et al. 2017). They recovered the 16-byte secret key of a GPU-based AES implementation by correlation timing analysis and differential timing analysis, respectively. Very recently, Gao et al. proposed electro-magnetic analysis attacks against a GPU-based AES implementation based on the cache line access coalescing (Gao et al. 2018; Gao et al. 2018), which are proved to be much efficient.

Contributions

To the best of our knowledge, this is the first work that investigates EM side-channel vulnerabilities of GPU-based bitsliced cryptographic implementation in the perspective of both bit-level parallelism and thread-level parallelism. The contributions are as follows:

First, we study the vulnerabilities of bit-level parallelism in a GPU-based bitsliced AES implementation. With the help of a multi-bit EM leakage features extracted from non-profiled t-test, we construct two special multi-bit combinational analysis methods, namely multi-bit feature combinational analysis and multi-bit decision combinational analysis, which take the full advantage of the bit-level parallelism of bitsliced ciphers and are experimentally proved to be more efficient than traditional single-bit CEMA.

Second, we also study the vulnerabilities of thread-level parallelism in the same AES implementation referred to above. In the study, a profiled correlation-based leakage detection test is employed to extract a multi-thread EM leakage feature, which is later used to construct special multi-thread combinational analysis methods. The proposed methods make the best of the thread-level parallelism of GPUs and our experimental result shows that the proposed combinational methods outperform traditional non-combinational ones.

Organization

The rest of this paper is organized as follows. In "Preliminary" section, we give a brief introduction of CUDA-enabled GPUs, a GPU-based bitsliced AES implementation, and definitions and notations involved in this paper. In "EM measurement" section, we present special techniques for leakage acquisition and preprocessing. In "The vulnerabilities of bit-level parallelism" and "The vulnerabilities of thread-level parallelism" sections, we investigate the side-channel vulnerabilities of bit-level parallelism and thread-level parallelism of the GPU-based bitsliced AES implementation, respectively. Finally, conclusions are given in "Conclusions" section.

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.

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× Streaming Multiprocessors (SM) and a global memory. Each SM has N× Scalar Processor (SP), a shared memory, several 32-bit 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 SIMT fashion; that is, all threads execute the same instruction, and each thread carries out that operation on its own private data.

GPU-based bitsliced AES implementation

The terminology *bitsliced cipher* was first proposed by Biham Biham et al. (1998) referring to the AES candidate *Serpent*. Precisely speaking, bitsliced cipher is a concept about cryptographic implementation instead of cryptographic algorithm or scheme itself.

The AES implementation of bitsliced version could process more than one 128-bit plaintext in a parallel fashion. The parallelism is determined by the word-length of a processor. For 32-bit processors, 32 128-bit plaintexts can be encrypted in parallel, which is also mentioned as *bit-level parallelism*. The first step of a bitsliced AES implementation is to transpose multiple plaintexts by bit in order to adapt bitsliced execution fashion. As showed 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 32×128 matrix is transposed before the first round encryption, and the inverse transposition is performed after the final round encryption. Obviously, only one forward transposition and one inverse transposition are needed to finish one bitsliced AES encryption on a single GPU thread. For multiple thread executions on GPU, each thread executes the above process independently, which is also referred to as *thread-level parallelism*. In a word, GPU-based bitsliced AES implementation achieves parallelism in two dimensions, namely bit-level parallelism and thread-level parallelism.

Definitions and notations

Definition 1 *For a bitsliced AES implementation on a 32-bit processor, 32 16-byte standard AES state is mapped as Fig. 1 to an* (8×16)*-sized matrix:*

$$\begin{pmatrix} W_0^1 & W_0^2 & W_0^3 & \cdots & W_0^{16} \\ W_1^1 & W_1^2 & W_1^3 & \cdots & W_1^{16} \\ W_2^1 & W_2^2 & W_2^3 & \cdots & W_2^{16} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ W_7^1 & W_7^2 & W_7^3 & \cdots & W_7^{16} \end{pmatrix} \tag{1}$$

where W_i^j *is a 32-bit value. The matrix is called bitsliced AES state, which is opposite to standard AES state. In addition, each bit of* W_i^j *is also called one slice.*

Definition 2 *For a side-channel attack to a multi-thread cryptographic implementation, if an attacker is able to choose the same random plaintexts for some threads and collect the corresponding ciphertexts and side-channel leakages, then the attack is called Chosen-Thread Side-Channel Attack (CTSCA). Please note that attackers cannot choose specific plaintext for any thread in CTSCA mode, which is different from chosen-plaintext side-channel attack.*

Definition 3 *For a CTSCA to an* $(\boldsymbol{m}\times\boldsymbol{n})$*-thread parallel cryptographic implementation, if an attacker is able to assign the same random plaintext to every consecutive* $\boldsymbol{m}$ *threads, the CTSCA is called* $\boldsymbol{n}$*-group multi-thread CTSCA or T*$\boldsymbol{n}$*G CTSCA for short, and the chosen-thread encryption is called* $\boldsymbol{n}$*-group multi-thread encryption or T*$\boldsymbol{n}$*G encryption for short.*

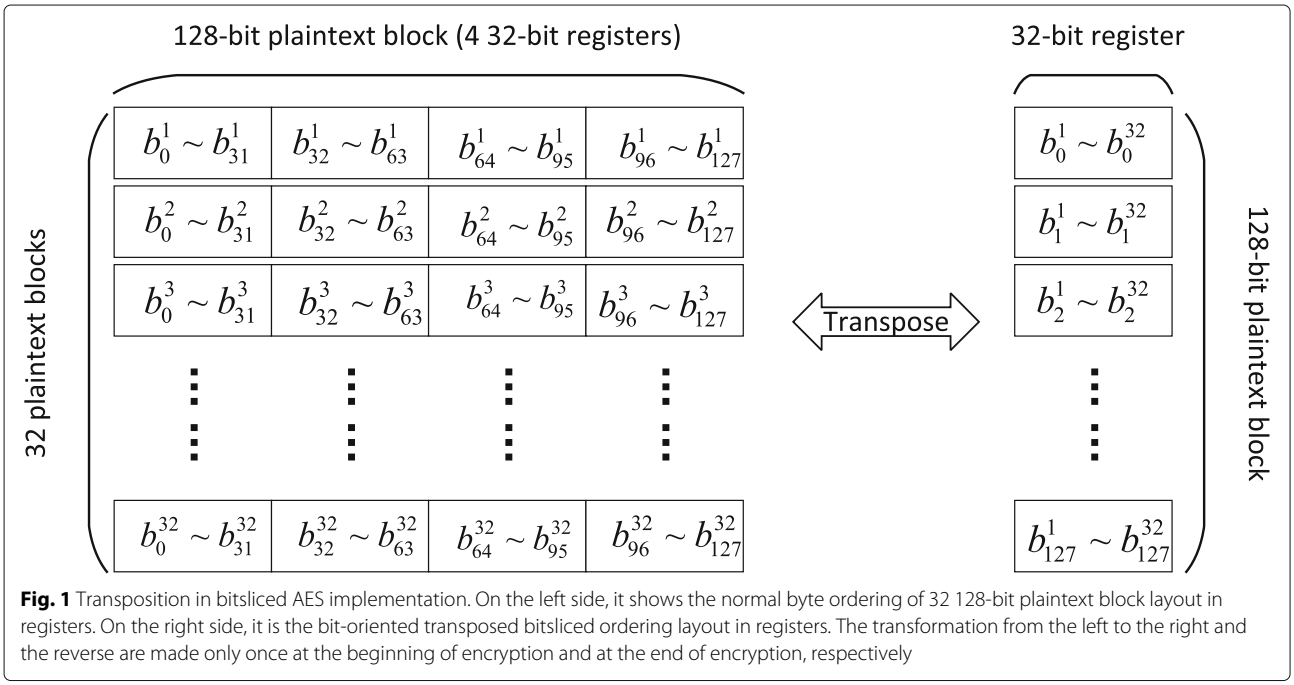

Fig. 1 Transposition in bitsliced AES implementation. On the left side, it shows the normal byte ordering of 32 128-bit plaintext block layout in registers. On the right side, it is the bit-oriented transposed bitsliced ordering layout in registers. The transformation from the left to the right and the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

Definition 4 *For a side-channel attack to a bitsliced cipher implemented on an* $\boldsymbol{m}\times\boldsymbol{n}$*-bit word-length processor, if an attacker is able to give the same plaintext for every consecutive* $\boldsymbol{m}$ *slices, the attack is called* $\boldsymbol{n}$*-group multi-slice CSSCA or S*$\boldsymbol{n}$*G CSSCA for short, and the chosen-slice encryption is called* $\boldsymbol{n}$*-group multi-slice encryption or S*$\boldsymbol{n}$*G encryption for short.*

Notations.

In this paper, $\cdot$ and $\star$ denote quantifier *all* and *any*, respectively. For example, if $\mathcal{M}$ is a 5×10 matrix, then $\mathcal{M}[2,\cdot]$ denotes the 10 elements of the second row of the $\mathcal{M}$, which is also a 10-dimensional vector, and $\mathcal{M}[\star,3]$ denotes any one of the 5 elements in the third column of the $\mathcal{M}$, which is also regarded as a set composed of 5 elements.

$\mathtt{Bit}_j(\mathcal{R})$: the function outputs the j-th bit of each component of $\mathcal{R}$ in its original format. For example, if $\mathcal{R} = [0x10010101, 0x01011101]$, then $\mathtt{Bit}_1(\mathcal{R}) = [0x1, 0x1]$ and $\mathtt{Bit}_4(\mathcal{R}) = [0x0, 0x1]$.

EM measurement

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 the advantage of localization effects, which makes EM attacks more efficient than power analysis 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 (Agrawal et al. 2002). 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.

Set-ups The testbed in this work is set up with the following configurations:

- We target an NVIDIA's GeForce GT 620 graphics card connected to the host with PCI-e bus. Though the device is of low performance, it is enough to show the vulnerability of NVIDIA's GPU to EM attacks. Specifically, the device has one streaming multiprocessor of 48 SPs, an 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 of an open source community (Patrick) into the GPU. Since it is a table-free implementation, we do not need to consider the efficiency of table look-up with respect to different types of memory. 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 sampling rate up to 20GHz (20GSa/s). We set our sampling rate as 200MSa/s, which turns out to be enough for our experiments.

Our testbed is set up in a client/server mode which is widely used for internet applications. Specifically, in cloud computing environment, cloud devices that provide SECaaS work as servers, and inside attackers (Duncan et al. 2015) are authorized to encrypt any plaintexts $\mathcal{P}$-s then obtain the corresponding ciphertexts $\mathcal{C}$-s and measured EM traces $\mathcal{T}$-s. With a sufficient number of triples $\langle\mathcal{P},\mathcal{C},\mathcal{T}\rangle$, the attackers attempt to recover the preset secret key of our GPU bitsliced AES implementation.

Locate Signals A printed circuit board such as GPU card is usually composed of hundreds of electronic components like chips, resistors, capacitors, inductors and so on. However, it is not necessary to check all of them to locate target signals. Generally speaking, only the right above of GPU chip and the capacitors on the back of GPU chip should be checked, because these positions or components tend to produce useful leakages, which is also confirmed afterwards in our experiments. More specifically, we start up the CUDA program and run the encryption procedure 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 patterns within the signal repeat nine to ten times, leakage positions are found. A repeated signal in our experiments is showed 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 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* signal 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 trace 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 showed in Fig. 3. First

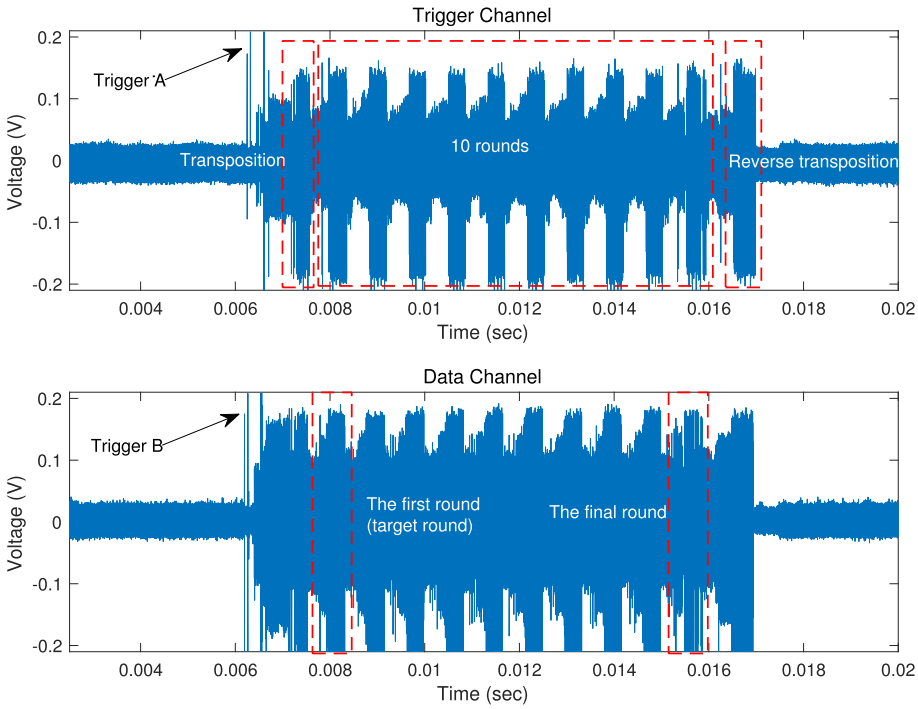

Fig. 2 Overall EM measurement of our GPU-based bitsliced AES implementation

of all, we observe the special patterns on the signal and find a *two-trough* (*C* in Fig. 3) pattern that is shared by all traces. The pattern is very 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, $\mathrm{PCC} > 0.70$). 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

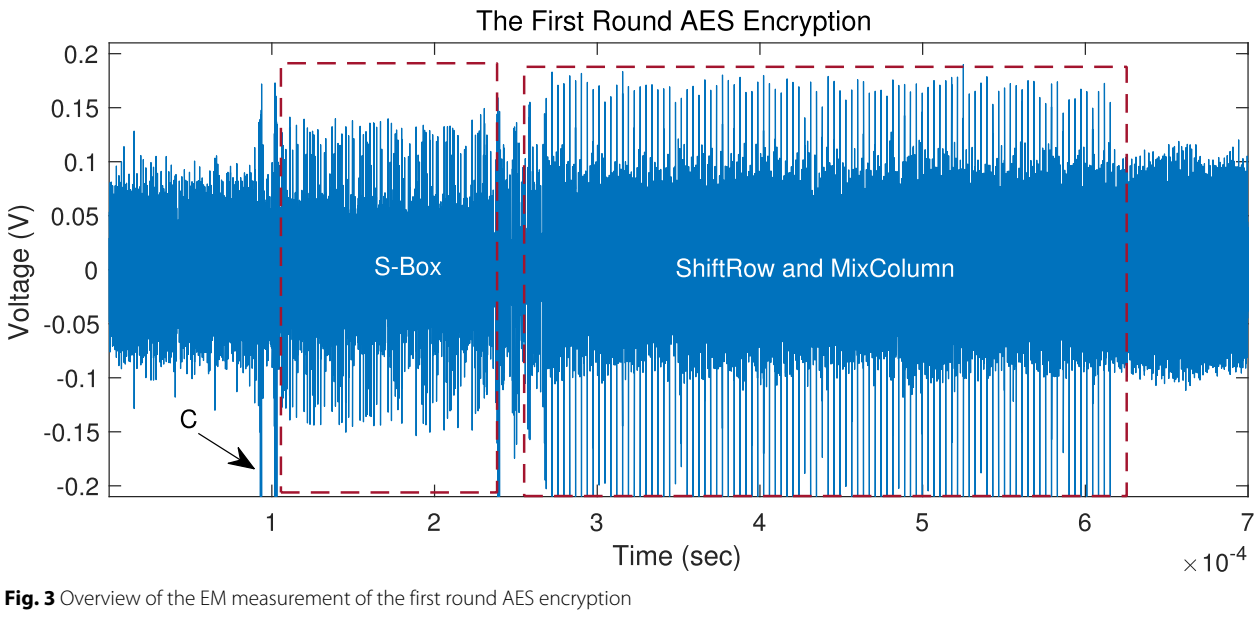

Fig. 3 Overview of the EM measurement of the first round AES encryption

the traces whose maximum PCC is less than 0.70. Then the traces with the maximum PCC no less than 0.70 will be aligned properly.

The vulnerabilities of bit-level parallelism

Bitsliced implementation features bit-level parallelism, which means that the bits located at the same position of multiple plaintexts are processed simultaneously at any moment. As known, the 16-byte *state* of standard AES implementation is processed byte by byte, so its EM leakages at any moment is a function of a byte. However, for bitsliced AES implementation the bits at the same position of the standard *states* from multiple slices are gathered into a single register of a processor, so the EM leakages at any moment is a function of several bits from the standard states of multiple slices. Therefore, for the bitsliced version each secret key byte is likely leaked at eight different moments at least, which makes it possible for specific multi-bit combinational analysis methods to be effective. In this section, we investigate the side-channel vulnerabilities of a GPU-based bitsliced AES implementation from the perspectives of bit-level parallelism provided by the intrinsic properties of bitsliced ciphers with combinational analysis techniques.

We analyze the output of SubByte in the first AES round of a bitsliced AES implementation (Patrick). The *C* code snippet of the SubByte of the implementation is shown in Fig. 4, where U and S are the input and output of the function, respectively, and T*x* and L*xx* are temporary variables. The function processes the SubByte of one byte for multiple slices. The length of a word_t is 32-bit for the target GPU, so 32 slices are processed simultaneously in any single GPU thread.

We know from S[0], S[1], S[2], S[3], S[4], S[5], S[6] and S[7] that for each byte in standard AES state the eight bits are processed independently, and the same bit of multiple slices are processed simultaneously. The fact is of great importance for the research in this section.

Non-profiled leakage detection test

Since the 128 bits in standard AES state are processed at different instants of time, the leakages from the 128 bits are not overlapped in the time domain. Therefore, it is possible for a non-profiled leakage detection test to each of the 128 bits to evaluate the amount of EM leakages from each individual bit. For other specific bitsliced AES implementations, the test results may differ much from ours, but the method itself works as well.

Specifically, Welch's *t*-test (Goodwill et al. 2011) is employed to detect the EM leakages from each of the 128 bits. Non-profiled leakage detection test requires specifying intermediates to be tested. In our test, we suppose that the 128 bits after the SubByte operation be the target intermediates.

Because of the multi-slice plaintexts used in bitsliced AES, we take into account of the CSSCA and CTSCA

```
211  void bs_sbox(word_t U[8])
212  {
213      word_t S[8];
214      word_t
215          T1,T2,T3,T4,T5,T6,T7,T8,
216          T9,T10,T11,T12,T13,T14,T15,T16,
217          T17,T18,T19,T20,T21,T22,T23,T24,
218          T25, T26, T27;
219
237
238      T1 = U[7] ^ U[4];
239      T2 = U[7] ^ U[2];
240      T3 = U[7] ^ U[1];
241      T4 = U[4] ^ U[2];
242      T5 = U[3] ^ U[1];
243      T6 = T1 ^ T5;
244      T7 = U[6] ^ U[5];
245      T8 = U[0] ^ T6;
246      T9 = U[0] ^ T7;
350      L22 = L3 ^ L12;
351      L23 = L18 ^ L2;
352      L24 = L15 ^ L9;
353      L25 = L6 ^ L10;
354      L26 = L7 ^ L9;
355      L27 = L8 ^ L10;
356      L28 = L11 ^ L14;
357      L29 = L11 ^ L17;
358      S[7] = L6 ^ L24;
359      S[6] = ~(L16 ^ L26);
360      S[5] = ~(L19 ^ L28);
361      S[4] = L6 ^ L21;
362      S[3] = L20 ^ L22;
363      S[2] = L25 ^ L29;
364      S[1] = ~(L13 ^ L27);
365      S[0] = ~(L6 ^ L23);
366
367      memmove(U,S,sizeof(S));
368  }
```

Fig. 4 The *C* code snippet of the SubBytes of a bitsliced AES

mode when performing a test. As showed in Fig. 1, the *state* of 32-slice bitsliced AES encryption can be formalized as a 128 × 32 binary matrix:

$$\mathcal{I} := \begin{pmatrix} b_0^1 & b_0^2 & b_0^3 & \cdots & b_0^{32} \\ b_1^1 & b_1^2 & b_1^3 & \cdots & b_1^{32} \\ b_2^1 & b_2^2 & b_2^3 & \cdots & b_2^{32} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{127}^1 & b_{127}^2 & b_{127}^3 & \cdots & b_{127}^{32} \end{pmatrix} \tag{2}$$

where b_i^j corresponds to the $(i+1)$-th bit in standard AES state of the j-th slice. The 32 bits in each row of $\mathcal{I}$ are stored into an independent register, so 128 registers are required to hold the (128 × 32)-bit *bitsliced AES state*. To reduce noises, our test is conducted in $S1G$ encryption mode, in which 32 slices are fed with the same plaintexts, so that each of the 128 registers hold either 32 zeros or 32 ones instead of 2^{32} possible values. The rationale is that to distinguish one of two values from the other is much easier than to distinguish one of 2^{32} values from the others. To further reduce noises, 32 threads in a warp also run the same plaintexts, which is also referred to as $T1G$ encryption mode.

Test Method The procedure of t-test consists of three steps. For our target implementation, t-test will be performed on each of the 128 intermediate bytes, say $b_0, b_1, b_2, ..., b_{127}$, where $b_i := b_i^1 b_i^2 b_i^3 \ldots b_i^{32}$. Now take the first intermediate byte b_0 for example.

First, N EM trace samples of M sampling points in the time domain are partitioned into two groups, namely G_0 and G_1, with respect to their corresponding intermediate byte $b_0 = 0$ or $b_0 = 2^{32} - 1$.

Then, t-statistic at each sampling point is computed:

$$t(\tau) = \frac{\mu_0(\tau) - \mu_1(\tau)}{\sqrt{\frac{s_0^2(\tau)}{n_0} + \frac{s_1^2(\tau)}{n_1}}} \tag{3}$$

where $\mu_0(\tau)$, $\mu_1(\tau)$ are the means of G_0 and G_1 at τ in time (or the τ-th sample point, so $\tau \in [1, M] \cap \mathbb{Z}$), and $s_0(\tau)$ and $s_1(\tau)$ are the standard deviations of G_0 and G_1 at τ in time, and n_0 and n_1 are the cardinality of G_0 and G_1.

Last, it is time to determine whether the two sets G_0 and G_1 are sampled from the same 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 (Schneider and Moradi 2015). We also follow this convention in our test.

The same tests are carried out for the leakages from the other 127 intermediate bytes, say b_1, b_2, ... , b_{127}. As a result, 128 × M t-statistics are obtained:

$$T = \begin{pmatrix} t_{0,1} & t_{0,2} & t_{0,3} & \cdots & t_{0,M} \\ t_{1,1} & t_{1,2} & t_{1,3} & \cdots & t_{1,M} \\ t_{2,1} & t_{2,2} & t_{2,3} & \cdots & t_{2,M} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{127,1} & t_{127,2} & t_{127,3} & \cdots & t_{127,M} \end{pmatrix} \tag{4}$$

where $t_{\star,j}$ corresponds to $t(j)$.

Test Results and Discussions As mentioned above, there are 128 registers to hold the (128 × 32)-bit *state* of bitsliced AES encryption. We perform the t-test on each of the 128 32-bit intermediate stored in the 128 registers. The test results are showed in Fig. 5. We observe that 16 peaks are clearly visible, and they are far beyond the preset thresholds $[-4.5, 4.5]$ for deciding if the value in a register is leaked or not. The 16 peaks in different colors represent the leakages of 16 `SubByte` outputs in the first round encryption of the bitsliced implementation.

As a matter of fact, the values of t-statistic in the time domain is not that important, because we intend to determine whether rather than when any of 128 intermediate bytes is leaked or not. Hence, the maximum of the values in each row of T is sufficient, so we define

$$\begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ \vdots \\ t_{127} \end{pmatrix} := \begin{pmatrix} \max_{j=0}^{M}\{|t_{0,j}|\} \\ \max_{j=0}^{M}\{|t_{1,j}|\} \\ \max_{j=0}^{M}\{|t_{2,j}|\} \\ \vdots \\ \max_{j=0}^{M}\{|t_{127,j}|\} \end{pmatrix} \tag{5}$$

Each consecutive 8 values, for example t_0, t_1, ...t_7, in $(t_0, t_1, t_2, ..., t_{127})^{\mathrm{T}}$ (Eq. 5) corresponds to one byte of standard AES *state*. As is known from standard AES algorithm, the 8 values all depend on the same key byte, so we say that each key byte is leaked through at least 8 intermediate bytes that are executed the same operation. This is also the essence of the distinctive leakage feature provided by the bit-level parallelism. In this study, the distinctive leakage feature is formally defined as a (16 × 8) matrix:

$$\Gamma := \begin{pmatrix} t_0 & t_1 & t_2 & \cdots & t_7 \\ t_8 & t_9 & t_{10} & \cdots & t_{15} \\ t_{16} & t_{17} & t_{18} & \cdots & t_{23} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{120} & t_{121} & t_{123} & \cdots & t_{127} \end{pmatrix} \tag{6}$$

The matrix describes the leakages on each of the 128 intermediate bytes. The experimental results for Γ are also showed in Fig. 6. Γ corresponds to the histogram, but the t-s that are beyond $[-4.5, 4.5]$ in Γ are reduced to zeros in

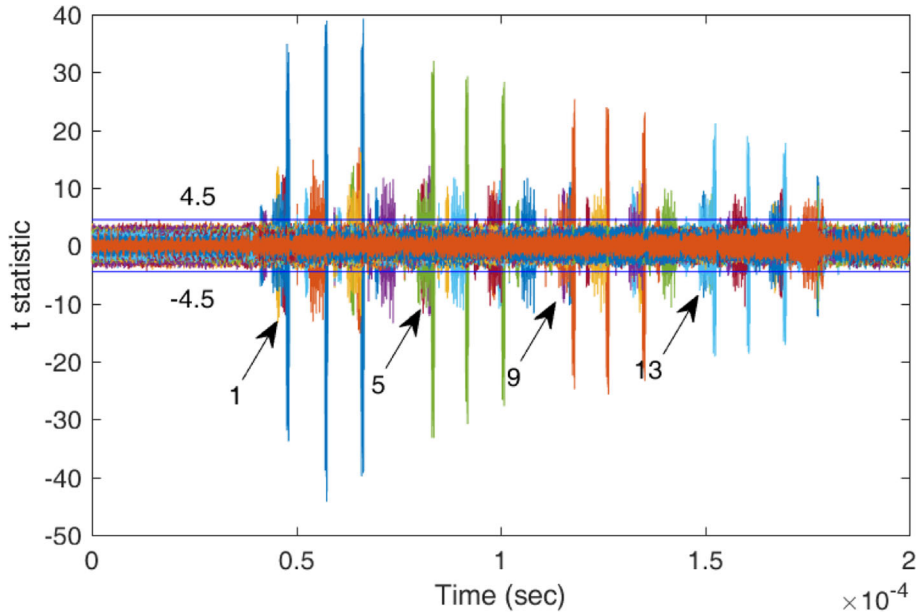

Fig. 5 Experimental results of Welch's *t*-test for all 128 bits of 16 intermediate bytes in *T1G-S1G* encryption mode

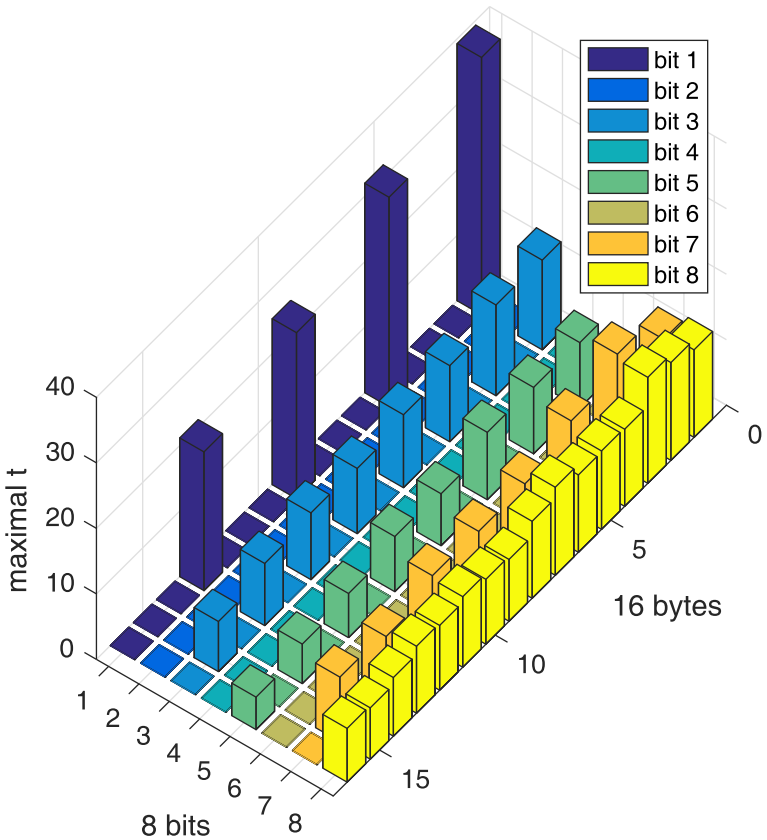

Fig. 6 Maximum of $t(\tau)$ in the time domain for all 128 individual bytes of the `SubByte` output in *T1G-S1G* encryption mode

the histogram. We also define another matrix based on Γ:

$$\mathcal{F} := \begin{pmatrix} \zeta_0 & \zeta_1 & \zeta_2 & \cdots & \zeta_7 \\ \zeta_8 & \zeta_9 & \zeta_{10} & \cdots & \zeta_{15} \\ \zeta_{16} & \zeta_{17} & \zeta_{18} & \cdots & \zeta_{23} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \zeta_{120} & \zeta_{121} & \zeta_{123} & \cdots & \zeta_{127} \end{pmatrix} \tag{7}$$

where for any $i \in \{0, 1, 2, ..., 127\}$, the variable $\zeta_i = 1$ if $t_i \notin [-4.5, 4.5]$; otherwise the variable $\zeta_i = 0$. In addition, we define a Multi-Bit Leakage Feature (MBLF):

$$\mathcal{J} := [\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3, ..., \mathcal{J}_{16}] \tag{8}$$

where $\mathcal{J}_n \subset \{1, 2, ..., 8\}$ denotes a set of indices of scalars that equals to 1 in $\mathcal{F}[n, \cdot]$.

A simple single-bit correlation analysis

We learn from the above that all 16 secret key bytes are leaked from the most significant bit (MSB) of the respective intermediate bytes, because all elements in the 8th column of F are equal to 1. Therefore, it is possible to recover all 16 secret key bytes if an MSB-based correlation EM analysis (Brier et al. 2004) (MSB-CEMA) is employed. The leakage model of the MSB-CEMA is

$$\mathcal{L}_n = a \cdot \sum_{j=1}^{32} \mathcal{I}[8 \cdot n, j] + \mathcal{N}_{noise} \tag{9}$$

where $\mathcal{L}_n$ denotes the predicted EM leakage of n-th intermediate byte, $\mathcal{I}[\cdot, \cdot]$ denotes the intermediate matrix in Eq. 2, and a is a scale factor, the value of which is insignificant. In addition, $\mathcal{N}_{noise}$ is a Gaussian noise.

The single-bit CEMA related above makes use of the leakages from the MSB of each intermediate byte. In fact, each secret key byte is leaked from more than one bit of relevant intermediate byte. For example, the first secret key byte is leaked from the 1st, 2nd, 6th and 8th bit and the second secret key byte is leaked from the 1st and 4th bit. Therefore, it is possible to take full advantage of the multi-bit leakages of a certain key byte to improve performance.

Multi-bit combinational analysis

The first combinational method is proposed to be multi-bit feature combinational analysis (MB-FCA). Just as its name suggests, MB-FCA makes the best of Γ showed in Eq. 6 to determine which of the eight bits is used to compute the predicted leakage. Specifically, the bit is selected as follows:

$$b_n = \underset{j}{argmax} \left(\max_{j=1}^{8} |\Gamma[n, j]| \right) \tag{10}$$

The rationale is that the intermediate bit with the maximum amount of EM leakage is the best candidate to predict the measured EM leakages.

Let $\mathcal{B} := [b_1, b_2, b_3, ..., b_{16}]$, and it is called combined multi-bit leakage feature (cMBLF). The $\mathcal{B}$ is the essence of the MB-FCA and used to construct the following leakage model:

$$\mathcal{L}_n = a \cdot \sum_{j=1}^{32} \mathcal{I}[(n-1) \times 8 + b_n, j] + \mathcal{N}_{noise} \tag{11}$$

With the leakage model derived from cMBLF, a simple CEMA will suffice to recover the 16-byte secret key.

The second combinational method is proposed to be multi-bit decision combinational analysis (MB-DCA). Compared with MB-FCA, MB-DCA does not care about the selection of intermediate bit before modeling leakages but tries on every leaked bits then makes a decision on the results of their respective analysis.

The first step of MB-DCA is to perform analysis on every leaked intermediate bits based on the MBLF, so the leakage model is:

$$\mathcal{L}_n^{\mathcal{J}_n\{i\}} = a \cdot \sum_{j=1}^{32} \mathcal{I}[(n-1) \times 8 + \mathcal{J}_n\{i\}, j] + \mathcal{N}_{noise} \tag{12}$$

where $\mathcal{J}_n\{i\}$ denotes the i-th element in $\mathcal{J}_n$, and $\mathcal{L}_n^{\mathcal{J}_n\{i\}}$ denotes the predicted leakage of the $\mathcal{J}_n\{i\}$-th bit of the n-th intermediate.

To recover the n-th secret key byte, $|\mathcal{J}_n|$ CEMAs should be performed before making a decision:

$$k_n = \underset{\tilde{k}}{argmax} \left(\max_{i \in \{1,2,...,|\mathcal{J}_n|\}, \tilde{k} \in [0,255] \cap \mathbb{Z}} |\rho_n^{\mathcal{J}_n\{i\}}(\tilde{k})| \right) \tag{13}$$

where $\rho_n^{\mathcal{J}_n\{i\}}(\tilde{k})$ is a matrix of Pearson Correlation Coefficient obtained from CEMAs based on the model $\mathcal{L}_n^{\mathcal{J}_n\{i\}}$. Obviously, it is feasible to recover the 16 secret key bytes of AES. More details about the MB-DCA is showed in Algorithm 1.

Experimental results and discussion

Since the number of multi-slice groups and multi-thread groups for the GPU-based bitsliced AES encryption of $(32 \times 32 \times 16)$-byte plaintext within a GPU warp have nothing to do with MB-FCA and MB-DCA, our experiments are performed in a simplified mode, say *T1G-S1G* encryption mode. Therefore, only one 128-bit plaintext is required in order to obtain one EM trace. As mentioned above, an MBLF and cMBLF must be extracted from $\mathcal{F}$ and Γ, respectively, before the MB-FCA or MB-DCA is applied. Finally, we obtain an MBLF $\mathcal{J}$ and a cMBLF $\mathcal{B}$ in our setting:

$$\begin{aligned} \mathcal{J} = [&\{1, 3, 7, 8\}, \{5, 8\}, \{3, 7, 8\}, \{4, 7, 8\}, \{1, 3, 7, 8\}, \{5, 8\}, \\ &\{3, 7, 8\}, \{4, 7, 8\}, \{1, 3, 7, 8\}, \{5, 8\}, \{3, 7, 8\}, \{4, 7, 8\}, \\ &\{1, 3, 7, 8\}, \{5, 8\}, \{3, 7, 8\}, \{4, 7, 8\}] \end{aligned}$$

Algorithm 1 Multi-Bit Decision Combinational Analysis (MB-DCA)

Input:

$[p^{(j)}_{i,n}]_{i\in\{1,2,...,N\},j\in\{1,2,...,32\},n\in[0,15]\cap\mathbb{Z}}$: $32\times N$ plaintexts.

$[\varsigma_{i,j}]_{i\in\{1,2,...,N\},j\in\{1,2,...,M\}}$: N EM traces with M sampling points on every traces.

$\mathcal{J}=[\mathcal{J}_1,\mathcal{J}_2,...,\mathcal{J}_{16}]$: MBLF from t-test.

Output:

$[k_0,k_1,...,k_{15}]$: 16-byte secret key recovered.

for $n\leftarrow 0$ to 15 **do**
  for $\tilde{k}\leftarrow 0$ to 255 **do**
    $r^{(\cdot)}_{\cdot,n}\leftarrow \mathtt{SBox}(p^{(\cdot)}_{\cdot,n}\oplus\tilde{k})$
    for $i\leftarrow 1$ to N **do**
      for $j\leftarrow 1$ to 8 **do**
        $s_{i,j}\leftarrow\sum_{b=1}^{32}\mathtt{Bit}_j(r^{(1)}_i)$
    for $\beta\leftarrow 1$ to $|\mathcal{J}_{n+1}|$ **do**
      $b\leftarrow\mathcal{J}_{n+1}\{\beta\}$
      for $m\leftarrow 1$ to M **do**
        $\rho_{m,\beta}\leftarrow|\mathtt{Corr}(\varsigma_{\cdot,m},s_{\cdot,b})|$
      $\phi_\beta\leftarrow\max\{\rho_{1,\beta},\rho_{2,\beta},...,\rho_{M,\beta}\}$
    $w_{\tilde{k}}\leftarrow\max\{\phi_1,\phi_2,...,\phi_{|\mathcal{J}_{n+1}|}\}$
  $k_n\leftarrow argmax_{\tilde{k}}\max\{w_0,w_1,...,w_{255}\}$
return $[k_0,k_1,...,k_{15}]$

$$\mathcal{B}=[1,8,7,8,1,8,8,8,1,8,7,8,1,8,7,8]$$

We compare the performance of MB-FCA and MB-DCA with MSB-CEMA in our experiments. The experimental results shows that a complete key-recovery attack with MSB-CEMA requires 900 EM traces at least (Fig. 7), while MB-FCA and MB-DCA are almost equivalent and require 500 EM traces at least. In fact, any scalar of $\mathcal{B}$ is very likely to be null if all values of the corresponding row in Γ are within $[-4.5,4.5]$. In this case, 16 key bytes can not be recovered completely. Formally, a complete key-recovery with MB-FCA or MB-DCA is feasible only if $\mathcal{F}$ satisfies

$$\sum_{i=1}^{16}\left(\bigvee_{j=1}^{8}\mathcal{F}[i,j]\right)=16. \tag{14}$$

We have to note that the MB-FCA or MB-DCA can not work if no prior leakage detection test is available, because both methods are based on the prior knowledge of Γ. That is to say, MB-FCA and MF-DCA are essentially profiled side-channel analysis techniques like template attacks (Chari et al. 2002). However, our methods are more practical than template attacks, because for devices of certain architectural model the profiling is done only once to extract Γ of the architecture before attacking any device of this architecture, while both profiling and attacking in template attacks usually target identical device, which makes template attacks less practical than our methods. In addition, template attacks require very low noise level, so they are almost ineffective for GPU-based cryptographic implementations.

The vulnerabilities of thread-level parallelism

GPU-based bitsliced implementation achieves thread-level parallelism because of the SIMT execution fashion of GPUs, which means multiple threads execute the same program in parallel. For simplicity, we consider the parallelism within a GPU warp, because the threads in a warp execute the same instruction at any moment if warp divergence does not happen. In other words, the threads in a warp achieve a full synchronization in the time domain. However, the full synchronization among multiple thread executions does not necessarily imply a full synchronization of their respective EM leakages. In this section, we investigate the vulnerabilities of a GPU-based bitsliced

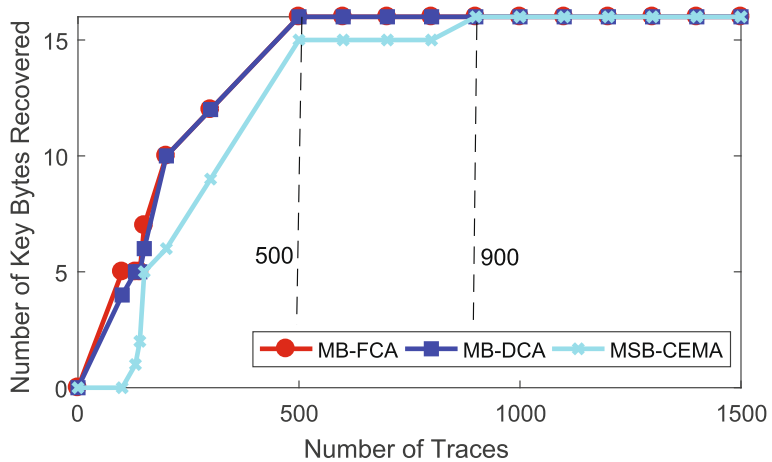

Fig. 7 Experimental results of MB-FCA, MB-DCA and MSB-CEMA, where MSB-CEMA denotes the single-bit CEMA with the MSB

implementation from the perspective of thread-level parallelism.

Developing a simple attack

As mentioned above, multiple thread executions within a GPU warp are synchronous, so their leakages are always considered to be synchronous as well. Suppose the Hamming weight leakages in multiple thread executions be summable, say $E\left(\sum_{n=1}^{N} \mathtt{HW}(I_n)\right) = \sum_{n=1}^{N} E(\mathtt{HW}(I_n))$, where I_1, I_2, I_3, ... I_N are some intermediates within N threads, and E denotes an ideal EM leakage model without any noise, say $E(x) = a \cdot x$ for a positive a. Then we construct a simple synchronous model (SSM) for multi-thread leakages in a warp:

$$\mathcal{L}_n^m = a \cdot \sum_{i=1}^{32} \mathtt{HW}\left(\mathcal{I}'[\,(n-1) \times 8 + m, i]\,\right) + \mathcal{N}_{noise}. \quad (15)$$

$\mathcal{I}'$ in the Equation is defined as follows:

$$\mathcal{I}' := \begin{pmatrix} B_0^1 & B_0^2 & B_0^3 & \cdots & B_0^{32} \\ B_1^1 & B_1^2 & B_1^3 & \cdots & B_1^{32} \\ B_2^1 & B_2^2 & B_2^3 & \cdots & B_2^{32} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ B_{127}^1 & B_{127}^2 & B_{127}^3 & \cdots & B_{127}^{32} \end{pmatrix} \quad (16)$$

where $\mathcal{L}_n^m$ denotes the predicted leakages based on the m-th intermediate bit, $B_i^\star := b_i^1 b_i^2 ... b_i^{32}$ is a 32-bit value, and $\mathcal{I}'[\,\cdot, j] := \left[B_0^j, B_1^j, B_2^j, ..., B_{127}^j\right]^{\mathrm{T}}$ corresponds to the $\mathcal{I}$ (Eq. 2) within the j-th thread of a warp.

The above SSM is based on another assumption, that is, the Hamming weight leakages in different threads are of the same scale. Otherwise, the model should be:

$$\mathcal{L}_n^m = \sum_{i=1}^{32} \left(a_i \cdot \mathtt{HW}\left(\mathcal{I}'[\,(n-1) \times 8 + m, i]\,\right)\right) + \mathcal{N}_{noise}. \quad (17)$$

The model essentially gives different weights to the predicted leakages from 32 threads, which should be more accurate than the previous SSM. Nevertheless, we will not study any analysis methods based on this model, because the value of $a_1 : a_2 : a_3 : ... : a_{32}$ has to be known somehow in advance, which we think is not very practical unless exhaustion.

SSM assumes that the EM leakage of multiple threads are synchronous. If all of multi-thread leakages are not synchronous, it comes to another leakage model called Partial Synchronous Model (PSM):

$$\mathcal{L}_n^m = a \cdot \sum_{i \in \mathcal{I}} \mathtt{HW}\left(\mathcal{I}'[\,(n-1) \times 8 + m, i]\,\right) + \mathcal{N}_{noise}. \quad (18)$$

where $\mathcal{I} \subset \{1, 2, ..., 32\}$ if the EM leakages of 32 threads are considered. Obviously, the key problem of constructing a PSM is to obtain the $\mathcal{I}$ some way.

Evaluations and Discussions In order to evaluate the performance of SSM-based CEMA (SSM-CEMA) to the GPU-based bitsliced AES implementation, we set up experiments in *T2G-S1G*, *T8G-S1G* and *T32G-S1G* encryption modes. As shown in Fig. 8, the 16 secret key bytes of AES can be recovered with about 1,500 EM traces

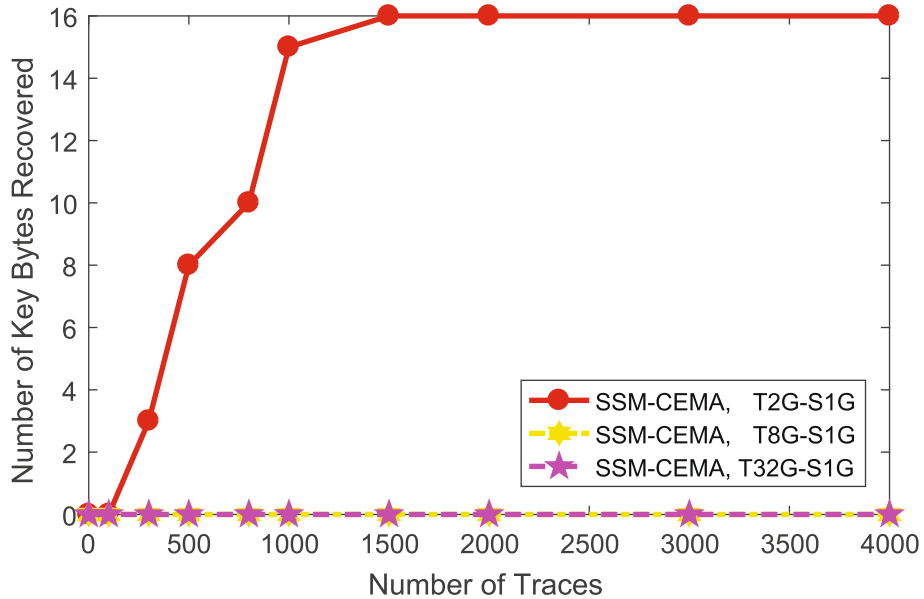

Fig. 8 The experimental results of SSM-CEMA in *T2G-S1G*, *T8G-S1G* and *T32G-S1G* encryption mode, respectively

in *T2G-S1G* mode, while none of the 16 secret key bytes is recovered with up to 4,000 EM traces in *T8G-S1G* or *T32G-S1G* mode. This suggests that the performance of SSM-CEMA becomes worse in more grouped thread encryption mode, because more noises are introduced and less accurate the SSM becomes. The latter reason will be verified in the next experiment.

In order to verify the accuracy of the SSM in *T2G-S1G* encryption mode. we compare the SSM with a PSM selecting half of 32 threads in a warp. As shown in Fig. 9, the PSM-CEMA with the first half of 32 threads performs better than the SSM-CEMA, which suggests that the PSM is more accurate than SSM. Therefore, the EM leakages in a warp may not be synchronous and a further study is needed.

Profiled correlation-based leakage detection test

To further understand the nature of the above parallel EM leakage, we employ a profiled leakage detection test method to analyze the individual leakages of multiple threads in a warp. With profiled methods, we do not have to make any assumptions about the leakage model of the target implementation, which thereby lowers the prerequisite for an attack and simplifies the procedure. The profiled ρ-test method we use is originally due to Durvaux and Standaert Durvaux and Standaert (2016). The method takes advantage of the cross-validation techniques introduced in Durvaux et al. (2014) and applies to the leakage detection of all threads in a warp. For the leakage test of one thread, the leakages from any other threads are treated as random noises. Specifically, the ρ-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 $\mathcal{T}$ is split into k (we set $k = 10$) non-overlapping subsets $\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, ..., \mathcal{T}^{(k)}$ of (approximately) the same size. For $i = 1, 2, 3, ..., k$, we define the profiling sets $\mathcal{T}_p^{(j)} = \bigcup_{i \neq j} \mathcal{T}^{(i)}$ and the test sets $\mathcal{T}_t^{(j)} = \mathcal{T} \backslash \mathcal{T}_p^{(j)}$. For each target plaintext byte variable $X_{m,n}$ with $m \in \{1, 2, 3, ..., 32\}$, $n \in \{1, 2, 3, ..., 16\}$ and for each cross-validation set j with $j \in \{1, 2, 3, ..., k\}$, a model is estimated: $\hat{model}_\tau^{(j)}(X_{m,n}) \leftarrow \mathcal{T}_p^{(j)}(\tau, m, n)$. 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 $\mathcal{T}_t^{(j)}(\tau, m, n)$:

$$\hat{r}_{m,n}(\tau) = \frac{1}{k} \cdot \sum_{j=1}^{k} corr(\mathcal{T}_t^{(j)}(\tau, m, n), \hat{model}_\tau^{(j)}(X_{m,n})) \tag{19}$$

where $m \in \{1, 2, 3, ..., 32\}$ and $n \in \{1, 2, 3, ..., 16\}$.

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

$$\hat{\rho}_{m,n}(\tau) = \frac{1}{2} \cdot ln\left(\frac{1 + \hat{r_{m,n}}(\tau)}{1 - \hat{r_{m,n}}(\tau)}\right) \cdot \sqrt{\frac{N}{k} - 3} \tag{20}$$

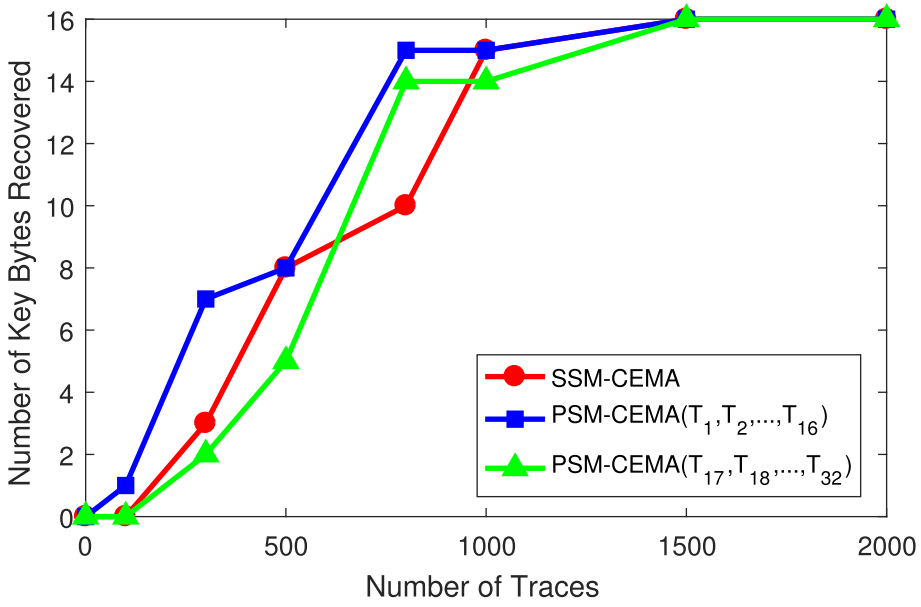

Fig. 9 The experimental results of SSM-CEMA and PSM-CEMA in *T2G-S1G* encryption mode. PSM-CEMA($T_1, T_2, ..., T_{16}$) and PSM-CEMA($T_{17}, T_{18}, ..., T_{32}$) denote CEMAs with PSM based on the leakages from respectively the first half and the second half of threads in a warp

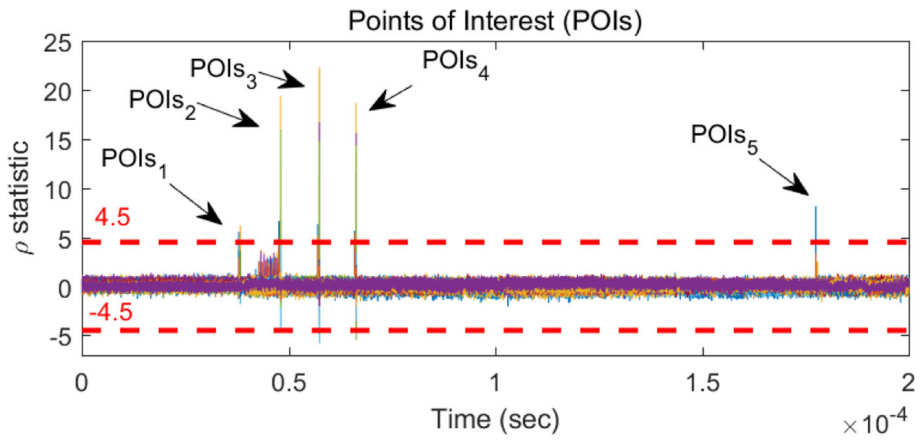

Fig. 10 The experimental results of ρ-test in *T32G-S1G* encryption mode. Point of Interest (POI) represents the point-in-time at which leakage happens

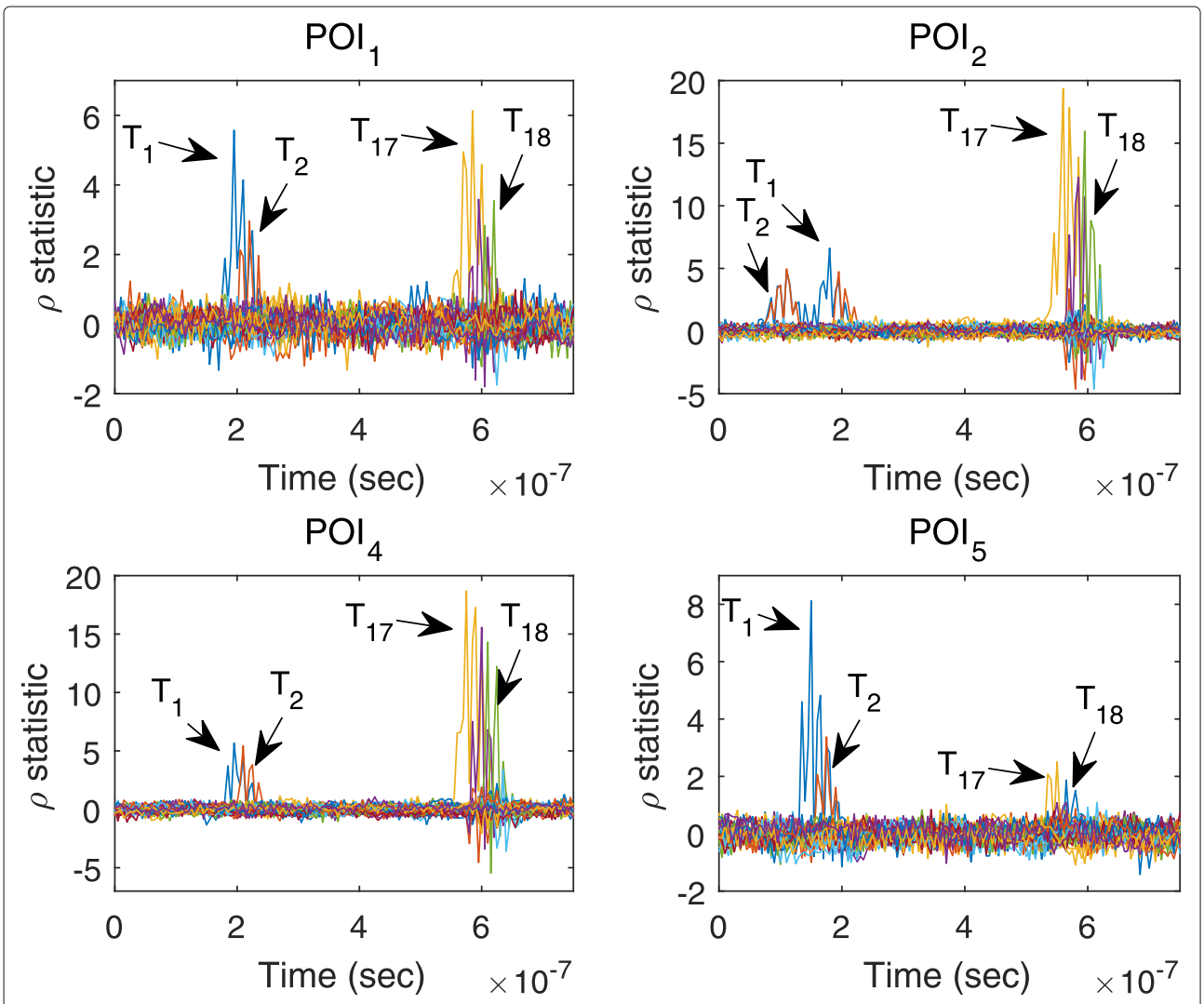

Fig. 11 The experimental results when zooming in POI_1, POI_2, POI_4 and POI_5. T_1, T_2, T_{17} and T_{18} represent the POIs from the 1st, 2nd, 17th and 18th thread within a warp, respectively

where N is the number of EM traces. Since $\rho_{\hat{m,n}}(\tau)$ satisfies standard normal distribution at any time τ, $|\hat{\rho}_{m,n}(\tau)| > 4.5$ can conclude the existence of leakage at τ with a much high probability (Schneider and Moradi 2015).

For each $m \in \{1, 2, 3, ..., 32\}$ and $n \in \{1, 2, 3, ..., 16\}$, we define $\rho_n^m := \max_\tau |\hat{\rho}_{m,n}(\tau)|$, so the following two matrices are obtained:

$$\Gamma' := \begin{pmatrix} \rho_1^1 & \rho_1^2 & \rho_1^3 & \cdots & \rho_1^{32} \\ \rho_2^1 & \rho_2^2 & \rho_2^3 & \cdots & \rho_2^{32} \\ \rho_3^1 & \rho_3^2 & \rho_3^3 & \cdots & \rho_3^{32} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho_{16}^1 & \rho_{16}^2 & \rho_{16}^3 & \cdots & \rho_{16}^{32} \end{pmatrix}, \mathcal{F}' := \begin{pmatrix} \xi_1^1 & \xi_1^2 & \xi_1^3 & \cdots & \xi_1^{32} \\ \xi_2^1 & \xi_2^2 & \xi_2^3 & \cdots & \xi_2^{32} \\ \xi_3^1 & \xi_3^2 & \xi_3^3 & \cdots & \xi_3^{32} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \xi_{16}^1 & \xi_{16}^2 & \xi_{16}^3 & \cdots & \xi_{16}^{32} \end{pmatrix} \tag{21}$$

where for $i = 1, 2, 3, ..., 16$ and $j = 1, 2, 3, ..., 32$, $\xi_i^j = 1$, if $\rho_i^j > 4.5$; otherwise, $\xi_i^j = 0$. In addition, the corresponding moments of occurrence is:

$$\Upsilon' := \begin{pmatrix} \tau_1^1 & \tau_1^2 & \tau_1^3 & \cdots & \tau_1^{32} \\ \tau_2^1 & \tau_2^2 & \tau_2^3 & \cdots & \tau_2^{32} \\ \tau_3^1 & \tau_3^2 & \tau_3^3 & \cdots & \tau_3^{32} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \tau_{16}^1 & \tau_{16}^2 & \tau_{16}^3 & \cdots & \tau_{16}^{32} \end{pmatrix} \tag{22}$$

where $\tau_i^j := argmax_\tau \max_\tau |\hat{\rho}_{m,n}(\tau)|$. We define a weak Multi-Thread Leakage Feature (wMTLF) Λ based on $\mathcal{F}'$:

$$\Lambda = [\Lambda_1, \Lambda_2, ..., \Lambda_{16}] \tag{23}$$

Table 1 Multi-thread combinational methods and non-combinational methods

Method	Leakge Feature	Leakage Model	Type
SSM-CEMA	-	SSM	Non-combinational
wPSM-CEMA	wMTLF	PSM	Non-combinational
MT-DCA	wMTLF	PSM	Combinational
sPSM-CEMA	sMTLF	PSM	Non-combinational
MT-HCA	sMTLF	PSM	Combinational

where $\Lambda_n \subseteq \{1, 2, ..., 32\}$ denotes a set of indices of scalars that equals to 1 in $\mathcal{F}'[n, \cdot]$. At the same time, we also define a strong Multi-Thread Leakage Feature (sMTLF) Ω based on $\mathcal{F}'$ and Υ':

$$\Omega = [\Omega_1, \Omega_2, ..., \Omega_{16}] \tag{24}$$

where Ω_n denotes a partition of Λ_n. The notation $\Omega_n\{i, j\}$ denotes the j-th element in the i-th subset of Ω_n. For any $i \in \{1, 2, ..., |\Omega_n|\}$ and $x, y \in \Omega_n\{i, \cdot\}$, $|\tau_n^x - \tau_n^y| < \delta$ satisfies, while for any $i, j \in \{1, 2, ..., |\Omega_n|\}$, $i \neq j$, $x \in \Omega_n\{i, \cdot\}$ and $y \in \Omega_n\{j, \cdot\}$, $|\tau_n^x - \tau_n^y| \geq \delta$ satisfies. δ is a threshold that determine whether the leakages of two threads are thought to be synchronous or not.

In our experiment, the ρ-statistic of 32 individual tests at each time τ are evaluated and plotted within a single figure as Fig. 10. It shows that there are approximately five groups of leakage points marked as POI_1, POI_2, POI_3, POI_4 and POI_5, respectively. Around any of the five groups, multiple colors are accumulated, which seems that the leakages from 32 threads happen at the same time.

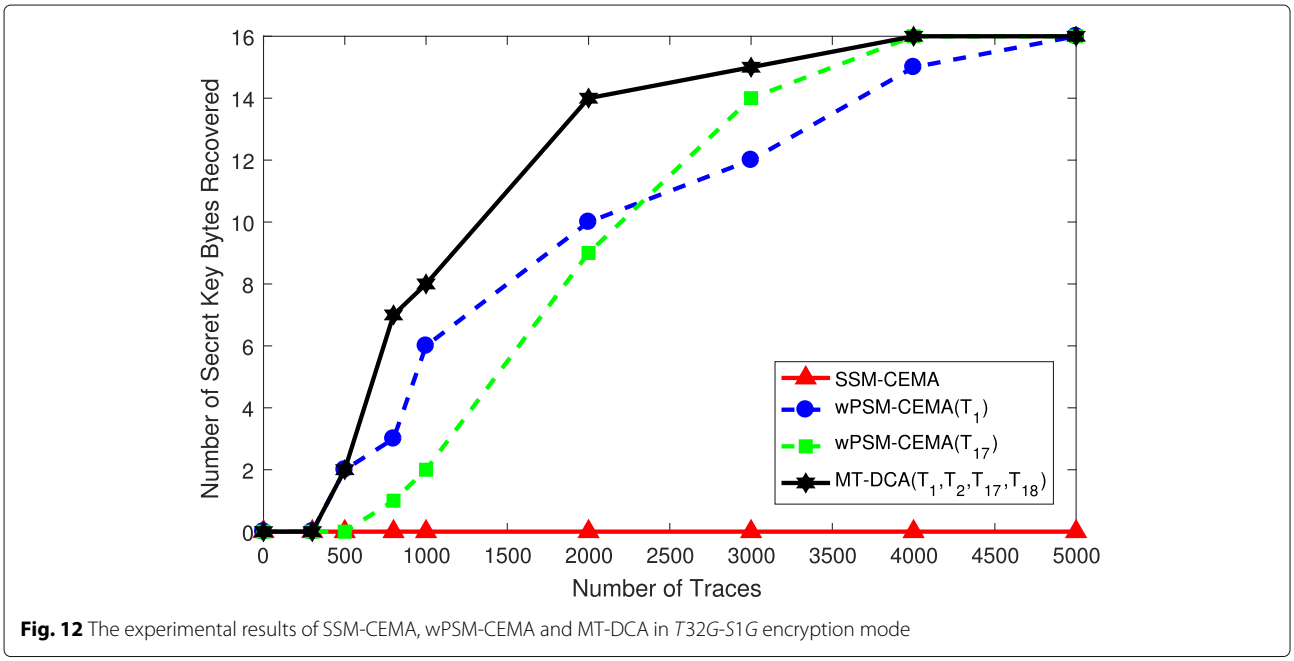

Fig. 12 The experimental results of SSM-CEMA, wPSM-CEMA and MT-DCA in *T32G-S1G* encryption mode

However, it is not like this when zooming in any of the five groups of leakage points. As is showed in Fig. 11, the detail of POI_1, POI_2, POI_4 and POI_5 tells 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 T_1 and T_2 are almost overlapping, the same with T_{17} and T_{18}. Since we cannot deny the existence of leakages when there is not any indication in the figure, we still do not know whether any leakage happens or not in other threads except T_1, T_2, T_{17} and T_{18}. We do not know why the device leaks information in these special threads, and we think it may have something to do with the hardware architecture of CUDA-enabled GPU as well as the target software implementation.

Multi-thread combinational analysis

The above experiments have demonstrated that the EM leakages from multiple threads in a warp is not exactly synchronous. To make full use of the EM leakages from multiple asynchronous threads, we propose two combinational methods.

The first method we propose is Multi-Thread Decision Combinational Analysis (MT-DCA) based on the wMTLF. If the j-th bit of a intermediate byte is used to model the leakage of single thread, then the MT-DCA is based on the following leakage model:

$$\mathcal{L}_n^{\Lambda_n\{i\}} = a \cdot \mathtt{HW}(\mathcal{I}'[\,(n-1)\times 8+j, \Lambda_n\{i\}) + \mathcal{N}_{noise} \quad (25)$$

where $\mathcal{L}_n^{\Lambda_n\{i\}}$ denotes the predicted leakage of the j-th bit of the n-th intermediate in the $\Lambda_n\{i\}$-th thread.

To recover the n-th secret key byte, $|\Lambda_n|$ CEMAs should be performed before making a decision:

$$k_n = \underset{\tilde{k}}{argmax}\left(\max_{\tilde{k}=0}^{255}\frac{\sum_{i\in\{1,2,\ldots,|\Lambda_n|\}}\left|\rho_n^{\Lambda_n\{i\}}(\tilde{k})\right|}{|\Lambda_n|}\right) \quad (26)$$

where $\rho_n^{\Lambda_n\{i\}}(\tilde{k})$ is a matrix of Pearson Correlation Coefficient obtained from CEMAs based on the model $\mathcal{L}_n^{\Lambda_n\{i\}}$. Obviously, it is feasible to recover the 16 secret key bytes of AES.

The above MT-DCA depends on wMTLF that indicates which of the 32 threads in a warp is leaky rather than whether these leakages are synchronous or not. To make full use of the synchronization of leakages, we propose the second combinational method called Multi-Thread Hybrid Combinational Analysis (MT-HCA) based on sMTLF. If the j-th bit of a intermediate byte is used to model the leakage of single thread, then the MT-HCA is based on the following leakage model:

$$\mathcal{L}_n^w = a \cdot \sum_{m\in\Omega_n\{w,\cdot\}} \mathtt{HW}(\mathcal{I}'[\,(n-1)\times 8+j, m]\,) + \mathcal{N}_{noise} \quad (27)$$

where $w \in \{1,2,\ldots,|\Omega_n\{w,\cdot\}|\}$ and $\mathcal{L}_n^w$ denotes the predicted leakage of the j-th bit of the n-th intermediate in the w-th grouped threads whose leakages are thought to be synchronous.

The same way as MT-DCA does, $|\Omega_n|$ CEMAs should be performed before making a decision:

$$k_n = \underset{\tilde{k}}{argmax}\left(\max_{\tilde{k}=0}^{255}\frac{\sum_{w=1}^{|\Omega_n|}\left|\rho_n^w(\tilde{k})\right|}{|\Omega_n|}\right). \quad (28)$$

where ρ_n^w is a matrix of Pearson Correlation Coefficient obtained from CEMAs based on the model $\mathcal{L}_n^w$. Obviously, it is feasible to recover the 16 secret key bytes of AES. More details of MT-HCA is showed in Algorithm 2.

Experimental results and discussion

Since both MT-DCA and MT-HCA take the advantage of EM leakages from multiple threads, our experiments are performed in $T32G$-$S1G$ encryption mode. MT-DCA and MT-HCA depend on wMTLF and sMTLF, respectively, so we extract the wMTLF and sMTLF of our experimental

Algorithm 2 Multi-Thread Hybrid Combinational Analysis (MT-HCA)

NOTE: The attack is based on the leakage of the MSB of each intermediate, so $\alpha = 8$ in the following procedure.

Input:

$[p_{i,n}^{(j,l)}]_{i\in\{1,2,\ldots,N\},j,l\in\{1,2,\ldots,32\},n\in[0,15]\cap\mathbb{Z}}$: $32\times32\times N$ plaintexts.

$[\varsigma_{i,j}]_{i\in\{1,2,\ldots,N\},j\in\{1,2,\ldots,M\}}$: N EM traces with M sampling points on every traces.

$\Omega = [\,\Omega_1, \Omega_2, \ldots, \Omega_{16}]$: sMTLF from ρ-test.

Output:

$[\,k_0, k_1, \ldots, k_{15}]$: 16-byte secret key recovered.

for $n \leftarrow 0$ to 15 **do**
  for $\tilde{k} \leftarrow 0$ to 255 **do**
    $r_{\cdot,n}^{(\cdot,\cdot)} \leftarrow \mathtt{SBox}(p_{\cdot,n}^{(\cdot,\cdot)} \oplus \tilde{k})$
    for $i \leftarrow 1$ to N **do**
      for $j \leftarrow 1$ to $|\Omega_{n+1}|$ **do**
        $s_{i,j} \leftarrow \sum_{e\in\Omega_{n+1}\{j,\cdot\}}\sum_{b=1}^{32}\mathtt{Bit}_\alpha(r_i^{(b,e)})$
    for $m \leftarrow 1$ to M **do**
      $\rho_m \leftarrow \frac{1}{|\Omega_{n+1}|}\cdot\sum_{w\leftarrow1}^{|\Omega_{n+1}|}\mathtt{Corr}(\varsigma_{\cdot,m}, s_{\cdot,w})$
    $w_{\tilde{k}} \leftarrow \max\{\rho_1, \rho_2, \ldots, \rho_M\}$
  $k_n \leftarrow \underset{k}{argmax}(\max\{w_0, w_1, \ldots, w_{255}\})$
return $[\,k_0, k_1, \ldots, k_{15}]$

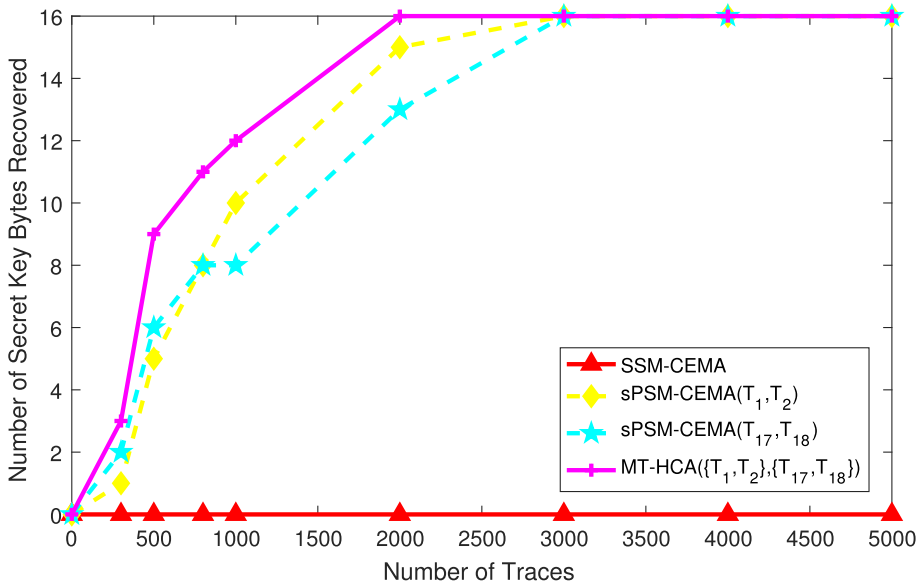

Fig. 13 The experimental results of SSM-CEMA, sPSM-CEMA and MT-HCA in *T32G-S1G* encryption mode

setting by ρ-test before attacking. The extracted wMTLF Λ and sMTLF Ω are as follows:

$$\begin{aligned}\Lambda =&[\{1,2,17,18\},\{1,2,17,18\},...,\{1,2,17,18\}]\\ \Omega =&[\{\{1,2\},\{17,18\}\},\{\{1,2\},\{17,18\}\},...,\{\{1,2\},\{17,18\}\}]\end{aligned} \tag{29}$$

that is, $\Lambda_1 = \Lambda_2 = ... = \Lambda_{16} = \{1,2,17,18\}$ and $\Omega_1 = \Omega_2 = ... = \Omega_{16} = \{\{1,2\},\{17,18\}\}$.

We compare the performance of five methods showed in Table 1. wPSM-CEMA computes predicted leakages by one thread in wMTLF and sPSM-CEMA computes predicted leakages by one grouped threads in sMTLF. In fact, both wPSM-CEMA and sPSM-CEMA are non-combinational methods.

First, we compare the performance of combinational method with non-combinational ones when wMTLF is available. The experimental results are showed in Fig. 12. wPSM-CEMA is a wMTLF-based non-combinational method and it performs worse than the wMTLF-based combinational method MT-DCA.

Second, we compare the performance of combinational method with non-combinational ones when sMTLF is available. The experimental results are showed in Fig. 13. sPSM-CEMA is a sMTLF-based non-combinational method and it perform worse than the sMTLF-based combinational method MT-HCA.

Both MT-DCA and MT-HCA performs better than their non-combinational counterparts because they make use of multi-thread leakages instead of a single one. It is reasonable that more usable leakages make better performance. In addition, the experimental results show that SSM-CEMA is not effective at all with up to 5,000 traces, because SSM-CEMA does not depend on any leakage feature, thus more noises are introduced when modelling with SSM.

Conclusions

In this paper, we investigate efficient electro-magnetic analysis of a GPU bitsliced AES implementation in order to give a deep insight into the vulnerabilities of bit-level parallelism and thread-level parallelism. We propose GPU-specific efficient combinational analysis methods and the methods are experimentally proved to be more efficient than the non-combinational ones. Our research suggests that multi-thread leakages can be used to improve attacks if the multi-thread leakages are not synchronous in the time domain.

Acknowledgements
This work was supported in part by National Natural Science Foundation of China (No. 61632020, UI936209) and Beijing National Science Foundation (No. 4192067).

Authors' contributions
YG provided the general idea of the work and write the whole manuscript, YZ provided valuable advices on the organization and expression of the paper; WC gave a great help on the experimental setups and the optimization of analysis methods. All authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
Not applicable.

Received: 16 January 2019 Accepted: 5 February 2020
Published online: 19 February 2020

References

Agrawal D, Archambeault B, Rao JR, Rohatgi P (2002) The EM side-channel(s). In: Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Revised Papers, Redwood Shores. pp 29–45. https://doi.org/10.1007/3-540-36400-5_4

Biham E (1997) A fast new DES implementation in software. In: Fast Software Encryption, 4th International Workshop, FSE '97, Proceedings, Haifa. pp 260–272. https://doi.org/10.1007/BFb0052352

Biham E, Anderson RJ, Knudsen LR (1998) Serpent: A new block cipher proposal. International workshop on fast software encryption. Springer, Berlin, Heilderberg

Brier E, Clavier C, Olivier F (2004) 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. pp 16–29. https://doi.org/10.1007/978-3-540-28632-5_2

Chari S, Rao JR, Rohatgi P (2002) Template attacks. In: Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15 2002, Revised Papers. pp 13–28. https://doi.org/10.1007/3-540-36400-5_3

Di B, Sun J, Chen H (2016) 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. pp 103–115. https://doi.org/10.1007/978-3-319-47099-3_9

Duncan AJ, Creese S, Goldsmith M (2015) An overview of insider attacks in cloud computing. Concurr Comput Pract Experience 27(12):2964–2981. https://doi.org/10.1002/cpe.3243

Durvaux F, Standaert F (2016) 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. pp 240–262. https://doi.org/10.1007/978-3-662-49890-3_10

Durvaux F, Standaert F, Veyrat-Charvillon N (2014) 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. pp 459–476. https://doi.org/10.1007/978-3-642-55220-5_26

Gao Y, Cheng W, Zhang H, Zhou Y (2018) 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. pp 300–306. https://doi.org/10.1109/TrustCom/BigDataSE.2018.00053

Gao Y, Zhang H, Cheng W, Zhou Y, Cao Y (2018) Electro-magnetic analysis of GPU-based AES implementation. In: Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29 2018. pp 121:1?-121:6. https://doi.org/10.1145/3195970.3196042

Goodwill JG, Jaffe J, Rohatgi P (2011) A testing methodology for side-channel resistance validation. In: NIST non-invasive attack testing workshop, Vol. 7

Jiang ZH, Fei Y, Kaeli DR (2016) A complete key recovery timing attack on a GPU. In: 2016 IEEE International Symposium on High Performance Computer Architecture, HPCA 2016, Barcelona, Spain, March 12-16 2016. pp 394–405. https://doi.org/10.1109/HPCA.2016.7446081

Jiang ZH, Fei Y, Kaeli DR (2017) 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. pp 167–172. https://doi.org/10.1145/3060403.3060462

Lim RK, Petzold LR, Koç ÇK (2016) Bitsliced high-performance AES-ECB on GPUs. In: The New Codebreakers - Essays Dedicated to David Kahn on the Occasion of His 85th Birthday. pp 125–133. https://doi.org/10.1007/978-3-662-49301-4_8

Luo C, Fei Y, Luo P, Mukherjee S, Kaeli DR (2015) 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. pp 281–288. https://doi.org/10.1109/ICCD.2015.7357115

Naghibijouybari H, Neupane A, Qian Z, Abu-Ghazaleh NB (2018) Rendered insecure: GPU side channel attacks are practical. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19 2018. pp 2139–2153. https://doi.org/10.1145/3243734.3243831

Nishikawa N, Amano H, Iwai K (2017) 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. pp 273–287. https://doi.org/10.1007/978-3-319-64701-2_20

Patrick C A bitsliced implementation of ECB and CTR AES. https://github.com/conorpp/bitsliced-aes. Accessed Mar 2016

Schneider T, Moradi A (2015) 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. pp 495–513. https://doi.org/10.1007/978-3-662-48324-4_25

Publisher's Note