МАТЕМАТИЧЕСКИЕ ВОПРОСЫ КРИПТОГРАФИИ
2016 Т. 7 № 2 С. 121–130

УДК 519.671

A timing attack on CUDA implementations of an AES-type block cipher

D. B. Fomin

Technical committee for standardization (*TC 26*), *Moscow*

Получено 15.II.2015

Abstract. A timing attack against an AES-type block cipher CUDA implementation is presented. Our experiments show that it is possible to extract a secret AES 128-bit key with complexity of 2^{32} chosen plaintext encryptions. This approach may be applied to AES with other key sizes and, moreover, to any block cipher with a linear transform that is a composition of two types of linear transformations on a substate.

Keywords: AES, Kuznyechik, Grasshopper, timing attack, cache attack, software timing attacks, CUDA, GPU

Атака по времени на CUDA-реализацию блочного шифра типа AES

Д. Б. Фомин

Технический комитет по стандартизации (*ТК 26*), *Москва*

Аннотация. Предлагается атака на реализацию блочного шифра типа AES на процессорах CUDA, основанная на времени выполнения шифрования. Эксперименты показали, что можно найти секретный 128-битовый ключ AES по шифрованию 2^{32} подобранных блоков открытого текста. Этот подход применим к AES с ключами других размеров и, более того, к любому блочному шифру, в котором линейное преобразование состоит из линейных преобразований двух типов, действующих на части состояния.

Ключевые слова: AES, Кузнечик, атака по времени, атака на кэш, программные атаки по времени, CUDA, GPU

Citation: *Mathematical Aspects of Cryptography*, 2016, v. 7, № 2, pp. 121–130 (Russian)

1. Introduction

A timing attack is a variant of a side-channel attack when an attacker exploits a correlation between the running times of the implementation of a cryptographic algorithm and the values of its input data (plaintext and key) to recover the secret key. Such attacks have been widely studied recently by researchers (see, for example, [1–4]).

Till now no successful timing attacks against GPU implementations of ciphers have been published. But there are a lot of different cache-timing attacks against CPU implementation. In two publications [8, 9] the timing attacks on CUDA implementations of AES and Blowfish were considered, but the overall conclusion of these papers were that this type of attack is impossible. At the same time these results show that information on the bank conflicts in shared memory of NVIDIA GPU may leak some information on the data. Here we use the same ideas to realize our attack.

We describe an attack against an AES-type block cipher CUDA implementation and also study a theoretical possibility of this type of attacks. This attack is based on a NVIDIA GPU architecture, that differs from CPU architecture. In Section 2 we explain CUDA architecture and the core idea of the attack. In Section 5 we discuss experimental results of the implementation of our attack on AES-128 block cipher.

2. CUDA in brief

CUDA is a software and hardware architecture developed by the NVIDIA company. It allows to produce non-graphical computing using GPUs. A detailed information on this architecture is available on [5].

CUDA architecture uses its own model of parallelism called SIMT (single instructions multiple thread). Virtually all threads work in parallel and have the same priority of memory access. Threads are grouped into blocks. The global synchronization between threads in different blocks is generally impossible, and for one block of threads synchronization is performed through a special memory called shared memory. All threads in a block are divided into groups of size 32; these groups are called warps. All threads in a warp at the same time perform the same instruction.

There are six types of memory in CUDA architecture: global, shared, registers, constant, texture and local. Shared memory is much faster than local and global memories. Blocks of the shared memory correspond to thread blocks, so all threads in the block have access to the same block of shared memory and it is always used as a user cache.

To achieve high memory bandwidth for concurrent accesses, shared memory is divided into equally sized memory modules (banks) that may be accessed simultaneously.

Shared memory of modern GPU has 32 banks consisting of successive 32 or 64 bits words. Each bank has a bandwidth of a word (32 or 64 bit long) per two clock cycles. If $b > 1$ addresses of memory requests fall in the same memory bank, then it is a *bank conflict* and the access has to be serialized and takes b times longer. On the other hand, shared memory features a *broadcast* mechanism whereby a word may be read and broadcast to several threads simultaneously when servicing one memory read request. This reduces the number of bank conflicts when several threads read from an address within the same 32-bit word. *A common conflict-free case is when all threads of a warp read from an address within the same word* [5].

3. Description of AES-type ciphers

We consider an AES-type block cipher such that its internal state is represented by two-dimensional arrays of words (as in AES, see [6]). Every internal state consists of N rows and N columns of words. Each round in AES-type block cipher consists of the following transformations:

- X — round key addition (`AddRoundKey` [1]),
- S — non-linear bijective transform – permutation (`SubBytes`[1]),
- L — linear transformation which is a composition of rows transformation (`ShiftRows` [1]) followed by columns transformation (`MixColumns`[1]).

So, the core feature of a linear transformation is that it may be decomposed into two special invertible linear transformations. Further we consider AES with 128-bit key, but our technique is also applicable to AES with other key sizes.

Let N be equal to 4 and the internal state be a square matrix of bytes, that is stored row by row:

$$\begin{pmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{pmatrix}.$$

Consider the following property of the linear transformation.

Proposition 1. *Let bytes* $x_{0,0}, \ldots, x_{3,3}$ *constitute an internal state of* AES *block cipher after the first* `AddRoundKey` *transformation and bytes* $a_{0,0}, \ldots, a_{3,3}$ *constitute an internal state after the first round, i. e. before the second* `AddRoundKey`. *If we fix one value from the column* $\{a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}\}$ *and any three values from the diagonal* $\{x_{0,0}, x_{1,1}, x_{2,2}, x_{3,3}\}$, *then the remaining variable* $x_{i,i}$ *is defined uniquely*.

[1] See AES description [6].

In terms of Proposition 1 for any fixed value of i we have 2^{24} possible values for the set $x_{0,0}, x_{1,1}, x_{2,2}, x_{3,3}$ to ensure the given value for $a_{i,0}$.

Apparently we can do the same operation with other sets:

we set specific values to:	to get given value for one of:
$\{x_{0,0}, x_{1,1}, x_{2,2}, x_{3,3}\}$	$\{a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}\}$
$\{x_{0,1}, x_{1,2}, x_{2,3}, x_{3,0}\}$	$\{a_{0,1}, a_{1,1}, a_{2,1}, a_{3,1}\}$
$\{x_{0,2}, x_{1,3}, x_{2,0}, x_{3,1}\}$	$\{a_{0,2}, a_{1,2}, a_{2,2}, a_{3,2}\}$
$\{x_{0,3}, x_{1,0}, x_{2,1}, x_{3,2}\}$	$\{a_{0,3}, a_{1,3}, a_{2,3}, a_{3,3}\}$

and can do it in 2^{24} different variants.

4. CUDA implementations of a block cipher

A GPU program generally consists of three main parts:

1) loading data to the internal memory of GPU,

2) evaluation on the GPU,

3) extracting data from the internal memory of GPU.

Modern CUDA architecture makes it possible to exclude the first and the third points of this list via using unified memory or page-locked host memory on some devices. But the implementation of an algorithm generally consists of these three parts.

To achieve maximal throughput of a block cipher implementation T-box lookup tables are used (merged an S-box and a linear transform) [7, 11–13]. This tables may be placed in a shared memory, texture memory or constant memory. Generally, the best place for the lookup tables is a shared memory because a thread has a high access speed to this type of memory. Sometimes, when lookup tables can't be placed in a shared memory (in view of their large sizes) the implementation with texture memory appears to be the most efficient one [7].

Shared memory access has one important property. When we encrypt random data we have to read random data from the memory, so throughput of the implementation with a shared memory might be faster without bank conflicts. If we can reduce bank conflicts we can increase the throughput of our implementation. For example, in Table 2 we present the encryption throughput for three types of encryption:

- the same block encryption multiple times — without bank conflicts,
- random data encryption — a lot of bank conflicts,
- counter encryption — less bank conflicts than random data encryption because the highest bit of counter is changed very rarely.

In Table 1 we present hardware and software specifications that we have used for experiments.

Table 1. Hardware and software specifications

OS	OpenSUSE 13.1	
CUDA compiler	nvcc ver. 6.0	
Graphics accelerator	NVIDIA GTX 285	NVIDIA GTX Titan
CUDA-cores	240	2688
GPU memory	1 GB	6 GB
Processor clock (MHz)	1 476	876
CPU	Intel Core i7-4770K	

Table 2. Encryption throughput for different encryption modes (GTX Titan), MB/sec

Algorithm	Constant encryption	Random data encryption	Counter encryption
AES 128	31113	12878	14195
AES 256	26489	10960	11867

We have detected that the encryption time constitutes a main part of the working time (without moving data between GPU memory and RAM). It is shown in Table 2 that the throughput of a counter mode encryption is smaller than that of a random data encryption in CBC mode; moreover, evaluation time is different for this two modes of operations. The core idea of our approach is based on this observation: we can encrypt different selected data and find a secret key as a function of the data minimizing the encryption time.

5. A timing attack on AES-128 block cipher

In [8, 9] the authors have shown that bank conflicts may leak some information on the data. Also they suggested a way to avoid it:

> "If the lookup tables are small enough (as in the case of AES) we can create multiple copies of them in the cache and stripe them across banks to make sure that there is always one entire copy of the table available through each bank." (Cf. [9].)

From our point of view one could face the following problems while implementing this approach. The first one is that threads in a bank may evaluate only the same instructions at a time. The second one is the growth of evaluation time. And the third one is that for $32 \cdot 4$ tables (without last round) we need 128KB of shared memory (32 is the number of banks) — more than the latest GPUs have.

There are many implementations of AES block cipher with different encryption data, keys, T-box allocation, see [9, 11–13]. The fastest one has parameters that are presented in Table 3.

Unfortunately we can't use ideas from [10] because for faster implementation we have to use a number of threads, but we may use only several 32-bit registers. CUDA architecture doesn't allow to use 128-bit registers (only as a built-in vector of four 32-bit registers).

Table 3. The best AES implementation parameters

T-box allocation	Shared memory
Data allocation	Registers
Round keys allocation	Shared memory
Size of data in thread	128 bit/thread — each thread encrypts a full data block

Timing attacks may be implemented when there exists some sort of correlation between the running times of the algorithm and the nature of the key and (possibly) data. Typically this happens when the same operation takes different times for different inputs. As we have showed earlier, data reading from a shared memory takes different amounts of time and it takes smaller time when there are no bank conflicts.

Let $K = \{k_{i,j},\ i,j = 0,1,2,3\}$ be a secret key. Assume that $k_{0,0} = 0, k_{1,1} = 0$, $k_{2,2} = 0$, and $k_{3,3} = 0$. In this way (in terms of Proposition 1 in Section 3) we can choose $x_{0,0}, x_{1,1}, x_{2,2}, x_{3,3}$ values in 2^{24} variants to fix $a_{0,0} = 0$ (value and indexes are presented as an example). So we can set encryption data array $\mathbf{Pt}$ as follows. Let array size equals M plaintext blocks. Each byte of the plaintext block we set up by a random value except the first one. Each first byte we'll choose in such a way that $a_{0,0} = 0$. Then the first byte in each block after the second `AddRoundKey` transformation will be the same and the encryption time will be reduced because in each warp all threads of a warp read from an address within the same word (see Section 2). So we can get 32 bits of the key K: $k_{0,0} = 0, k_{1,1} = 0, k_{2,2} = 0, k_{3,3} = 0$ if the encryption time will be smaller than the encryption of M random plaintext blocks.

Let $\alpha, \beta, \gamma, \delta \in GF\left(2^8\right)$. If we add $\alpha, \beta, \gamma, \delta$ to each plaintext in $\mathbf{Pt}$ as follows:

add α to $(0,0)$ byte, add β to $(1,1)$ byte,
add γ to $(2,2)$ byte, add δ to $(3,3)$ byte,

we can specify 32 bits of a key $K = \{k_{i,j},\ i,j = 0,\ldots,3\}$: $k_{0,0} = \alpha$, $k_{1,1} = \beta$, $k_{2,2} = \gamma, k_{3,3} = \delta$. So we can find 32 bits of a key if we can encrypt our array $\mathbf{Pt}$ (that is chosen for each encryption) 2^{32} times and find in what step the encryption time is the minimal. The number of the step with the minimal encryption time corresponds to the right choice of $\alpha, \beta, \gamma, \delta$ for 32 bits of the key.

Apparently, we can find other 12 bytes of the key independently and find all bytes of the key only for $3 \cdot 2^{32}$ array encryptions. So we can find all 128 bits of the key with complexity $4 \cdot 2^{32}$ array encryptions. Moreover, we can find all 128 bits of the key with only 2^{32} data encryptions, if we can encrypt specially $a_{0,0}=0$, $a_{0,1}=0$, $a_{0,2}=0$, $a_{0,3}=0$ for each of four key parts.

In Table 4 we present the size of data array and the encryption time with broadcast ($\mathbf{Pt}$ array in key find case) and without it (random data encryption time).

Encryption time constitutes the main part of the working time (without moving data between GPU memory and RAM). According to Table 4 the minimum possible time to encrypt one array is about 2 ms.

Table 4. Encryption time for different M with broadcast and without it, sec

GPU	encryption time by GTX Titan		encryption time by GTX 285	
M	the wrong choice of $\alpha, \beta, \gamma, \delta$	the right choice of $\alpha, \beta, \gamma, \delta$	the wrong choice of $\alpha, \beta, \gamma, \delta$	the right choice of $\alpha, \beta, \gamma, \delta$
524 288	—	—	0.00196643	0.00196088
1 048 576	0.00224006	0.0022334	0.00374532	0.00373562
2 097 152	0.00436898	0.00435408	0.00730337	0.00728249
4 194 304	0.00862449	0.00859434	0.0144648	0.0144242
8 388 608	0.0142342	0.0141866	0.02874	0.0286593
16 777 216	0.0283605	0.0282638	0.0572906	0.057132
33 554 432	0.0566222	0.0564296	—	—
67 108 864	0.113143	0.112767	—	—
134 217 728	0.226171	0.225415	—	—

To realize this attack an attacker should know the evaluation time. We suppose that he has access to the computer by malware or in some other manner but hasn't got a key. Also we suppose that he can encrypt any data with an unknown AES key. In 2014 some new software bugs (like Shellshock) were found permitting an attacker to gain unauthorized access to a computer system [14]. It seems not too difficult to estimate a kernel working time (if the access to a computer is possible) because, for example, we can evaluate any program from NVIDIA Visual Profiler. There is no need to have root privileges to run such a program and furthermore the profiler may be used in a command-line mode.

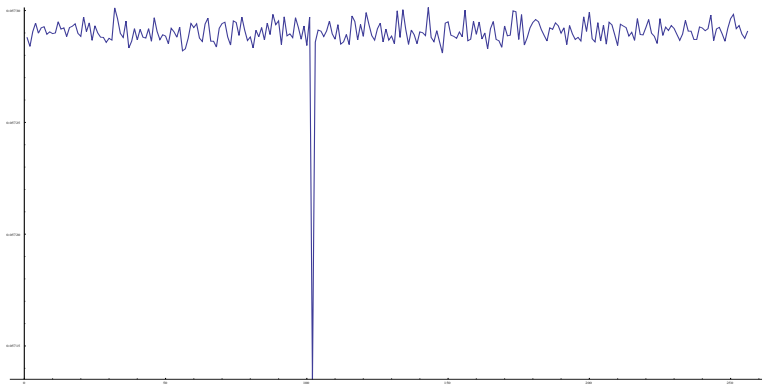

Fig. 1. Timing attack on GTX 285 (256MB array size)

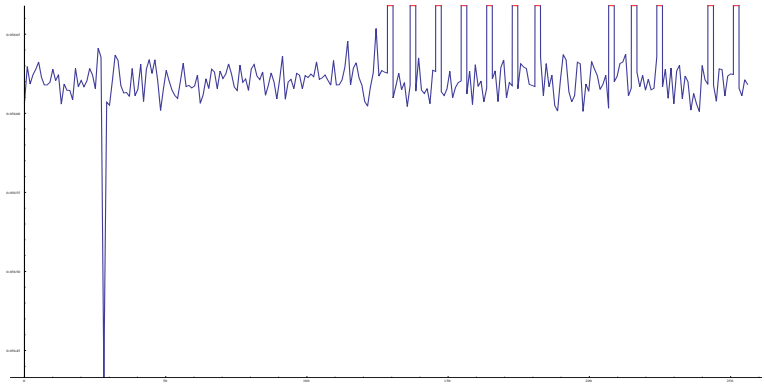

Fig. 2. Timing attack on GTX Titan (512MB array size)

We run an experiment to find a secret key in the following way: we have suppose that 15 bytes of a key are known and tried to find all other bytes of a key. We encrypted data with $\mathbf{Pt}$ array size $M = 33\,554\,432$ on GTX Titan and $M = 16\,777\,216$ on GTX 285. We tried to find $(3, 3)$ byte of the key. Based on Fig. 1 and 2 the shortest encryption time was detected and so the right key was found. It is quite easy to implement such an attack.

On February 2015 recovering attacks for all 128 bits of a key were implemented on eight NVIDIA Tesla K10.G2.8GB that were allowed by the NVIDIA Technology Center. It takes shorter time than 12 days to recover the whole key.

6. Comparison with other LSX-block ciphers

In this paper we use specific properties of AES block cipher suitable for implementation: small lookup tables and simple linear transformation. In the case of an LSX-block cipher with MDS-linear transform like in Kuznyechik (Grasshopper) [15] we can't use this attack: to fix any value in the second round after the second X transform we have to fix all 16 bytes of internal state. In the case of AES we may fix only 4 bytes. So, to find the first 128-bit round key of Kuznyechik we have to encrypt 2^{128} modifications of a specific array, but to find the first 128-bit round key of an AES block cipher we may encrypt only 2^{32} modifications of a specific array.

On the other hand, if a linear transform of a LSX block cipher may be represented as a composition of two or more linear transforms with suitable invariant subspaces of the state space, the attack might be applicable.

7. Conclusions

It this paper we have shown a possibility of a timing attack on an AES-type block cipher CUDA implementations. We demonstrate that it is possible to recover a secret 128-bit key with the complexity of 2^{32} specific data encryptions. We use specific properties of AES block cipher: small lookup tables and simple linear transformation. Also we analyse GPU architecture to show a theoretic efficiency of this type of attacks.

This attack may be applied also to AES with other key sizes and, moreover, to any AES-type block cipher. Nevertheless this attack is not applicable to LSX-block ciphers with MDS-linear transform like in Kuznyechik [15].

We would like to thank the reviewers for their helpful remarks.

References

[1] Page D., "Theoretical use of cache memory as a cryptanalytic side-channel", *IACR Cryptology ePrint Archive*, Report 2002/169 (2002), 14 p., https://eprint.iacr.org/2002/169.pdf.

[2] Bernstein D. J., *Cache-timing attacks on AES*, Tech. Rept., Chicago, IL: Dept. Math., Statist. and Comput. Sci., Univ. Illinois, 2005, 37 p., https://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[3] Kocher P. C., "Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems". In: "*Advances in Cryptology–CRYPTO'96*", Lect. Notes Comput. Sci., **1109**, 1996, 104–113.

[4] Schindler W., "A timing attack against RSA with the Chinese Remainder Theorem". In: "*Cryptographic Hardware and Embedded Systems–CHES 2000*", Lect. Notes Comput. Sci., **1965**, 2000, 109–124.

[5] *CUDA Toolkit documentation*, Santa Clara, CA: NVIDIA Corporation, http://docs.nvidia.com/cuda.

[6] FIPS PUB 197: *Advanced Encryption Standard* (AES), Gaithersburg, MA: Nat. Inst. Stand. Technol. (NIST), 2001, 47 pp., http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[7] Fomin D. B., "Implementation of an XSL block cipher with MDS-matrix linear transformation on NVIDIA CUDA", *Математические вопросы криптографии* (*Math. Aspects Cryptogr.*), **6**:2 (2015), 99–108.

[8] Mukherjee R., Rehman M. S., Kothapalli K., Narayanan P. J., Srinathan, K., *Fast, Scalable, and Secure Encryption on the GPU*, Hyderabad: Internat. Inst. Inform. Technology, 2011, 10 pp., http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.624.5065rep=rep1type=pdf.

[9] Mukherjee R., *A Performance Prediction Model for the CUDA GPGPU Platform*, M. S. Thesis, Hyderabad: Internat. Inst. Inform. Technology, 2010, 58 pp., http://web2py.iiit.ac.in/research_centres/publications/view_publication/mastersthesis/48.

[10] Käsper E., Schwabe P.,, "Faster and timing-attack resistant AES-GCM". In: "*Cryptographic Hardware and Embedded Systems – CHES 2009*", Lect. Notes Comput. Sci., **5747**, 2009, 1–17.

[11] Iwai K, Nishikawa N., Kurokawa T., "Acceleration of AES encryption on CUDA GPU", *Int. J. Network. Comput.*, **2**:1 (2012), 131–145, http://www.ijnc.org/index.php/ijnc/article/view/38/37.

[12] Kipper M., Slavkin J., Denisenko D., *Implementing AES on GPU. Final Report*, Toronto: Univ. Toronto, 2009, 10 pp., http://www.eecg.toronto.edu/∼moshovos/CUDA08/ arx/AES_ON_GPU_report.pdf.

[13] Manavski S. A., "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography". In: "*2007 IEEE International Conference on Signal Processing and Communications – ICSPC 2007*", (CD-edition), Los Alamitos, CA: IEEE Computer Soc., 2007, 65–68.

[14] L. Seltzer, "Shellshock makes Heartbleed look insignificant", *ZDNet* (E-edition), September 29 (2014), http://www.zdnet.com/article/shellshock-makes-heartbleed-look-insignificant/.

[15] Dygin D. M., Lavrikov I. V., Marshalko G. B., Rudskoy V. I., Trifonov D. I., Shishkin V. A., "On a new Russian Encryption Standard", *Математические вопросы криптографии* (*Math. Aspects Cryptogr.*), **6**:2 (2015), 29–34.