High-Performance Symmetric Cryptography Server with GPU Acceleration

Wangzhao Cheng[1,2,3], Fangyu Zheng[1,2(✉)], Wuqiong Pan[1,2], Jingqiang Lin[1,2], Huorong Li[1,2,3], and Bingyu Li[1,2,3]

[1] Data Assurance and Communication Security Research Center, Beijing, China
[2] State Key Laboratory of Information Security, Institute of Information Engineering, CAS, Beijing, China
{chengwangzhao,zhengfangyu,panwuqiong,linjingqiang,lihuorong,libingyu}@iie.ac.cn
[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Abstract. With more and more sensitive and private data transferred on the Internet, various security protocols have been developed to secure end-to-end communication. However, in practical situations, applying these protocols would decline the overall performance of the whole system, of which frequently-used symmetric cryptographic operations on the server side is the bottleneck. In this contribution, we present a high-performance symmetric cryptography server. Firstly, a symmetric algorithm SM4 is carefully scheduled in GPUs, including instruction-level implementation and variable location improvement. Secondly, optimization methods is provided to speed up the inefficient data transfer between CPU and GPU. Finally, the overall server architecture is adopted for mass data encryption, which can deliver 15.96 Gbps data encryption through network, 1.23 times of the existing fastest symmetric cryptographic server. Furthermore, the server can be boosted by 2.02 times with the high-speed pre-calculation technique for long-term-key applications such as IPSec VPN gateways.

Keywords: Symmetric Cryptographic Algorithm
Graphics Processing Unit (GPU) · CUDA · SM4 implementation
Symmetric cryptography server · Performance

1 Introduction

Cloud computing, e-commerce, online bank and other Internet services are developing rapidly, more and more sensitive and private data is transferred on the

W. Cheng—This work was partially supported by the National 973 Program of China under award No. 2014CB340603 and the National Cryptography Development Fund under award No. MMJJ20170213.

S. Qing et al. (Eds.): ICICS 2017, LNCS 10631, pp. 529–540, 2018.
https://doi.org/10.1007/978-3-319-89500-0_46

Internet. SSL, TLS, IPSec and other security protocols have been developed to support secure and reliable end-to-end communication. Under these security protocols, servers and clients first use key exchange protocols to negotiate a symmetric key, and then use the symmetric key to encrypt the communication content. From 2011 to 2015, the total amount of global data have increased for more than 10 times (from 0.7 ZB to 8.6 ZB), and CPU performance is only about three times higher, CPU's development can hardly meet the expanding demand.

In order to satisfy the need for symmetric calculation power, we present a high-performance symmetric cryptography server. Our server can be deployed in cloud computing, database encryption, end-to-end encrypted communication and other applications. It works as a proxy and outsources these complex and onerous symmetric computation from the original server.

To build the high-performance symmetric cryptography server, we accomplished a high speed SM4 kernel, used a GPU card as an SM4 algorithm accelerator, and optimized the network service based on the characteristics of GPU card. Our work is to gradually optimize the following three aspects.

1. Speeding up the SM4 kernel. We used CUDA's PTX (Parallel Thread Execution) instructions to accomplish bitwise exclusive OR operation and circular shift operation in the algorithm. At the same time, we adjusted the order of plain-text in global memory and modified its accessing method to increase plain-text's accessing rate. S-Box and round keys were also carefully arranged. On NVIDIA GeForce GTX 1080, our SM4 kernel is able to encrypt 535.68 Gb data per second. It achieves 12.59 times of the existing fastest SM4 implementation by Martínez-Herrera et al. [12].
2. Enhancing the overall throughput of the GPU card. Using GPUs as an accelerator, data transfer between GPUs and CPUs limits its overall throughput. We took advantage of multi-stream's parallel operation, and overlapped data transfer and calculation process with each other. Finally, the overall throughput of GPU card is enhanced to 76.89 Gbps, and our optimizations methods make use of 85.4% of GTX 1080's PCI-E bandwidth.
3. Optimizing the network service. Based on the characteristics that the GPU card is weak performance under single thread, and strong performance under multiple threads, we designed a queue to cache network service requests, and the GPU card can handles multiple encryption requests in the queue at the same time. The server's peak throughput through network is 15.96 Gbps. For the case that using a single key, like large IPSec VPN gateways, we designed a memory management framework and used pre-calculation technique to decrease data copy operations and reduce the degree of coupling. With these optimizations, the server's peak throughput reaches 32.23 Gbps, and it is 2.48 times faster than SSLShader [6].

The remainder of the paper is organized as follow. Section 2 introduces the related work. Section 3 presents the overview of GPU, CUDA, and SM4 symmetric encryption. Section 4 describes in detail about the GPU-accelerated SM4 implementation, the optimization for data transfer, and how we enhance our

server's performance. Section 5 analyses the performance of proposed algorithm and server, and it also compares them with previous works. Section 6 concludes the paper.

2 Related Work

The SM4 algorithm is Chinese standard symmetric cipher for data protection, and it is first declassified in 2006 and standardized in 2012. Researchers have done a lot of works on its security and attacking methods [2,3]. Chinese government is also vigorously promoting the SM4 algorithm as a standard for data protection. However, the heavy cryptographic computation of SM4 limits its using scene, and there are few studies to solve this problem.

As a graphics dedicated processor, GPUs are concentrated in the field of computer graphics [1]. After CUDA was introduced, GPU became widely used in the field of general purpose computing, many symmetric algorithms were scheduled in GPUs for better performance. Manavski et al. [11] took the lead in using CUDA to accelerate AES, which followed the Rijndael reference implementation, and provided optimal location for storing the T-tables to enhance the benchmark performances, and their work was followed by most teams. Harrison et al. completed AES-CTR on CUDA enabled GPUs [5]. While optimizing the algorithm's calculation rate, this work also contributed on how to schedule serial and parallel execution of block ciphers on GPUs. On Tesla P100, Nishikawa [13] increased the speed of the AES-ECB to 605.9 Gbps. Except for AES, the optimization of other symmetric algorithms on the GPU has also been extensively studied by researchers, including DES [10], Blowfish, IDEA, CAST-5, Camellia [4], MD5-RC4 [8].

In addition to symmetric cryptography, GPUs were also widely applied in asymmetric cryptography, including RSA [6,16] and ECC [17]. GPU-based cryptographic servers were also growing rapidly. Using GPU as a general-purpose SSL accelerators, SSLShader accelerated SSL cryptographic operations, and it handled 29K SSL TPS and achieved 13 Gbps bulk encryption throughput [6]. Guess [15] was a dedicated equipment for signature generation and verification, and it was capable of 8.71×10^6 operations per second (OPS) for signature generation or 9.29×10^5 OPS for verification.

3 Background

3.1 GPU and CUDA

Compared with CPUs, modern GPUs devote most of the transistor to arithmetic processing unit with much less data cache and flow controler. This unique hardware design is specialized for manipulating computer graphics, and it is also quiet fit for compute-intensive operations and high parallel computation. CUDA is a parallel computing platform, which was created and firstly introduced in

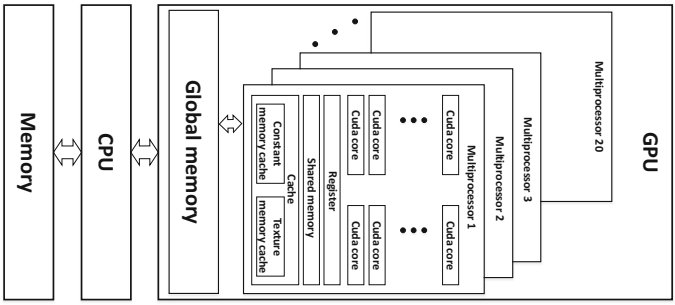

Fig. 1. NVIDIA GeForce GTX 1080's architecture and how it works in the system

2006 by NVIDIA [14]. By harnessing the power of GPUs, it enables dramatic computing performance increases on GPUs.

The target platform in this paper is NVIDIA GeForce GTX 1080. As shown in Fig. 1, GeForce GTX 1080 contains 20 streaming multiprocessors (SM) and each SM owns 128 single precision CUDA cores, 8 texture mapping units (TMU), and 256 KB L2 cache. 32 threads (grouped as a warp) within one SM run in a clock concurrently. All GPU threads follow the Single Instruction Multiple Threads (SIMT) architecture. When a warp is stalled due to memory access delay, it may be preempted, and the scheduler may switch the runtime context to another available warp. For better utilization of the pipeline, multiple warps of threads are usually assigned to one SM and called one block. The maximum number of GPU threads per block is 1024. Each block could access 96 KB fast shared memory, 48 KB L1 cache, and 64K 32-bit registers. All SMs share 8 GB global memory, cached read-only texture memory and cached read-only constant memory. Global memory is the off-chip video RAM, and it is accessible to both the device and the host through special functions provided by CUDA. Texture memory and constant memory are special area of global memory, both of them have separate on-chip catch.

3.2 SM4 Symmetric Encryption

SM4 is a 32 round unbalanced Feistel cipher, and it is Chinese standard symmetric cipher for data confidentiality [9]. The processing block and the encryption key are both 128-bit, and its security strength is same to that of AES-128.

4 Implementation Architecture

In this work, our main contributions are on these aspects: (*i*) Algorithm level optimizations for scheduling SM4 encryption on GPUs. (*ii*) Optimizations for data transfer between CPUs and GPUs. (*iii*) Optimizing the overall performance of the server.

4.1 Optimizing SM4 for GPU

For the SM4 kernel, naive porting of CPU algorithms to a GPU would waste most GPU computational resources and cause server performance degradation, In this part, we describe our approaches and design choices to maximize performance of SM4 kernel on GPUs, and the key point in maximizing SM4 performance lies in rational use of GPU storage resources and reducing GPU computational resources consumption.

The plani-text of SM4 encryption is divided into 4 32-bit blocks. With the corresponding round key, the round function uses these 4 blocks to generation a new 32-bit block. Generated block and the last 3 blocks are the input for the next round function, and the first block will never be used again. In order to reduce the usage of registers, we use the first block of the original input to store the generated block. SM4 symmetric encryption operation consists of 32 round functions, and the entire encryption process originally required 36 32-bit registers. With our tricks, only 4 32-bit registers are used in the entire process.

Then we optimized the implementation of the round function. For the round function, it carries out bitwise exclusive OR, left circular shift operations and non-linear permutations on these 32-bit inputs. PTX instruction *xor.b32* is used to complete bitwise exclusive OR operation for two 32-bit units. The implementation for the left circular shift operation is more complicated, we used *shl.b32*, *shr.b32* and *or.b32* these three instructions.

For non-linear permutations, we build a 256 bytes size S-Box, and it needs to query the S-Box for 4 times to implement the non-linear permutations for a 32-bit block, and we carefully considered the arrangement for the S-Box. The naive way is to store the S-box in global memory. Global memory's accessing rate is slow, and it results in SM4 kernel performance degradation. Except for global memory, GPUs have 32-bit register, constant memory and shared memory, which also can be used to store S-Box. Through comparative tests, we find when S-box is stored in constant memory, the performance of SM4 kernel is the best. GPUs have specially on-chip cache for constant memory, and after data in constant memory was loaded in the on-chip cache, the program get the data directly from the cache, rather than device memory. Register is also optional in certain circumstances, compared with using constant memory, the implementation of SM4 kernel that uses register to store S-Box is resistant to key recovery timing attack [7], although it has compromised in terms of performance.

After optimizing the round function, we also need a reasonable arrangement of the data for the SM4 kernel. During the encryption process, there are three kinds data: plain-text, round keys and S-box. As mentioned above, S-Box is stored in constant memory. The arrangement for round keys varies depending on the applications. When the application uses multiple keys, round keys for each GPU thread is different, so each GPU thread must derives its own round keys and store them in global memory. When only a single key is used in the application, we use CPU to derive round keys in advance and store them in shared memory for faster accessing rate. The plain-text is stored in global memory, because other storage space on GPUs are not suitable. Register is very limited, and normally

the size of plain-text is quite big, using 64K 32-bit registers to store plain-text results in degradation of the SM4 kernel. Every GPU thread access different plain-text, so shared memory is not a good choice. Plain-text is not constant value, so constant memory is not suitable too.

As plain-text is stored in global memory, and compared with other memory space, global memory has lower bandwidth, and we then optimized the inefficient accessing to the plain-text in global memory. We used the coalesced access to improve the accessing efficiency. Coalesced access needs two conditions: (i) The size of data accessed by a thread each time must be 4 bytes, 8 bytes or 16 bytes; (ii) The memory space accessed by a warp must be continuous. We used INT, INT2, and INT4 these unique instructions provided by the CUDA API to access global memory. INT, INT2, and INT4 are used to access 4,8,16 bytes global memory respectively. To meet the second condition, we adjust the order of the data blocks, and every GPU thread accesses the global memory in a special method. Under normal circumstances, each piece of plain-text encrypted by the same key is processed by one GPU threads, and multiple pieces are combined into one data block and then transferred to GPU's global memory. In this case, every thread is accessing a continuous memory space, but each time when a warp performs a memory accessing operation, it accesses multiple discrete data fragments, just as shown in Fig. 2(a). To benefit from coalesced access, we adjust the order of the data blocks. First, every piece of plain-text is split into several blocks, the size of the block depends on the instruction used to access global memory, instruction INT, INT2 and INT4 corresponds to 4-bytes, 8-bytes and 16-bytes block size respectively. Second, rearranging these blocks, make sure that all pieces's first blocks are combined in the natural order, and so it is with other blocks, as shown in Fig. 2(b). Every time a GPU thread accesses global memory, the next hop of the pointer also needs to point to the corresponding block. Under this condition, the accessed memory space by a warp is continuous, and each thread is processing its corresponding data.

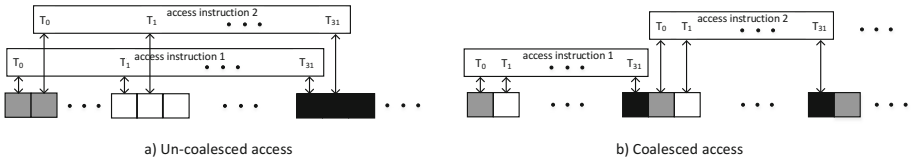

Fig. 2. How global memory is accessed by a warp

We designed a set of comparative experiments to evaluate which is the best combination of instructions and access methods. In the experiments, except for the difference in the methods of accessing global memory and the used instructions, other conditions are the same. We test the throughput of these SM4 kernel implementations, and the result is shown in Table 1.

Compared with non-coalesced access, the throughput of coalesced access is about 5 or 6 times higher. This is because that global memory resides in the

device memory, which is accessed via 32, 64 or 128 bytes memory transactions [14]. When a threads accesses global memory, the data need to be read in the cache by generating 32, 64 or 128 bytes memory transactions, then the thread gets data from the cache. As the size of the memory transactions is fixed, so more data than the actual demand will be cached. When the conditions of coalesced access are met, most threads directly gets the data from the cache without generating more memory transactions. The result also shows that instruction INT4 is the best choice.

Table 1. Comparing the throughput of non-coalesced access and coalesced access.

	INT	INT2	INT4
Non-coalesced access (Gbps)	78.58	81.29	81.80
Coalesced access (Gbps)	420.48	422.60	423.67

At last, we set the SM4 kernel's parameters to 20 blocks and each block owns 1024 GPU threads, so that the SM4 kernel maximize the use of GPU's computational resources.

4.2 Optimizing Data Transfer Between GPUs and CPUs

After optimizing SM4 kernel on GPUs, we used a GPU card as an SM4 accelerator. Using GPU card to complete the data encryption operation, first, it needs to copy plain-text from the host memory to GPU's global memory, then encrypt plain-text. After the encryption operation is complete, it needs to copy the cipher-text back to the host memory. These processes must be executed serially. To make the most of GeForce GTX 1080's two copy engines, we used multiple streams to complete these operations in parallel, so that data transfer between CPUs and GPUs overlaps with encryption operation, and the overall throughput of GPU card increases. The specific optimization methods are as follows: First, we initialize multiple streams and divide the data into corresponding fragments. Then these streams begin to perform operations such as data transfer and calculation process on theirs respective data asynchronously, as shown in Fig. 3.

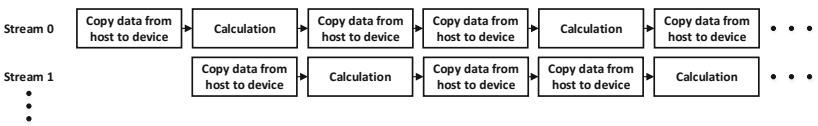

Fig. 3. How to overlap data transfer with calculation process

To achieve the highest throughput of the GPU card, the parameter of the SM4 kernel changed. We reduce the threads per block from 1024 to 512, every stream

only hold one block, and the stream number is 10. This is because our SM4 kernel is much faster than the data transfer, reducing the number of blocks and the number of threads per block improve the efficiency of the parallel execution between these streams.

After our optimization, the throughput of the GPU card reaches to 76.89 Gbps. Compared with the SM4 kernel, it is much slower. The test tool provided by the NVIDIA CUDA Toolkit shows that the bandwidth of our target platform's (GeForce GTX 1080) PCI-E is about 90 Gbps. So Our optimization makes use of 85.4% of the bandwidth, and this result is satisfactory.

4.3 Optimizing the Performance of the Server

We build a scalable SM4 encryption proxy that uses the high-performance SM4 encryption operations of GPU to outsources symmetric computation from the original server. Meanwhile, we complete ECB, CBC and CTR these three mainstream encryption modes, and the server is capable of providing data encryption through TCP/IP protocol.

In the server, CPU accepts service requests from network and caches them in a queue. When the GPU is task-free, CPU takes out requests and organize the plain-text, then GPU is called for data encryption. In this way, GPU provide data encryption for multiple service requests at the same time, and its parallel execution ability can be fully exploited. For the case of using multiple keys, GPU threads use different keys, so each GPU threads derives its own round keys, and then uses them to encrypt plain-text.

In some applications, end to end communication data is encrypted by a single key over a period of time, like large IPSec VPN gateways. For these applications, round keys are derived for only one time by CPU, then stored in shared memory and used by all GPU threads.

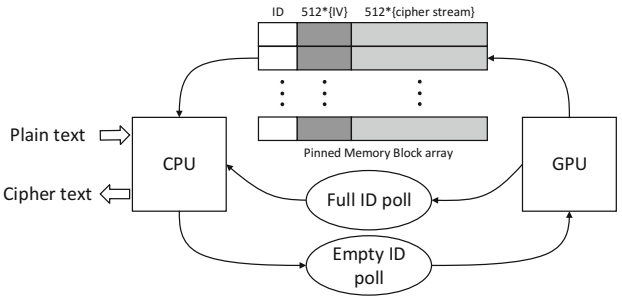

Fig. 4. The specially designed memory management framework

In the single key case, we have further optimizations. Based on the feature that CTR mode supports pre-calculation, we use the GPU to generate stream ciphers and store them in host memory. When CPU received service request,

it just encrypts the plain-text with the stream ciphers, and send the cipher-text back. In the case of multiple keys, before plain-text is encrypted, it would be copied for three times, including pushed in the queue, popped out from the queue and copied to a piece of continuous pinned host memory. In the single key case, with our optimizations, plain-text is directly encrypted with no other copy operation. And at the same time, through our optimizations, CPU and GPU work independently, the degree of the system's coupling is greatly reduced. It is worth mentioning that technique can be applied to enhance the overall performance of GPU servers for other symmetric algorithms, such as AES-CTR, and DES-CTR.

Figure 4 shows how our optimizations work in detail. First, we designed a memory management framework. In the framework, there is a pinned memory block array, each pinned memory block is pre-allocated and specially designed to store 512×16 bytes IVs and the generated cipher streams. Every block has an unique ID, and all these IVs and cipher streams could be accessed by querying the ID. In the initialization process, CPU fills the IVs with random numbers and pushes these IDs into the empty ID poll. After the initialization is completed, GPU begins to generate cipher streams: it pulls out IDs from the empty ID poll and uses the IVs in the block to generate cipher streams. When the cipher streams are full filled, these IDs are pushed into the full ID poll. When GPU is generating cipher streams, CPU gets an ID from the full ID poll and deals with network requests. When it receives a request, it generates the cipher-text with the cipher streams, and then sends the encrypted data back. If the cipher streams are used up, CPU refills the IVs, pushes the ID back to the empty ID poll and gets another ID from the full ID poll.

5 Performance Assessment

In this section, we evaluated our SM4 kernel, and compared it with other implementations. We tested the capabilities of our symmetric cryptography server, and compared it with other servers. Our server platform was equipped with one Intel E5-2699 v3 CPU, 8 GB memory, and one NVIDIA GTX 1080 cards.

Table 2. Comparison of the best obtained results including AES-128.

	Our SM4 kernel	Our GPU card	FPGA-based SM4 [12]	Nishikawa's AES-128 kernel [13]
Platform	GTX1080	GTX1080	xc6vhx380t-3ff1155	Tesla P100
Throughput (Gbps)	**535.68**	**76.8**	42.54	605.9
Gbps per CUDA core	**0.21**	-	-	0.17
Gbps per $	**0.90**	-	-	0.1

5.1 Performance of Our SM4 Kernel and Comparison with Other Implementations

The existing fastest SM4 implementation belongs to Martínez-Herrera et al. [12]. Through speeding-up the polynomial multiplier in SM4 algorithm on an FGPA (which is xc6vhx380t-3ff1155), their implementation reaches 42.54 Gbps.

In order to test the performance of our SM4 kernel, plain-text and secret key are generated by CPUs with random value and then transferred to GPU's global memory. Each GPU thread generates its own round keys and loops the encryption for 1 KB plain-text. Our test result is shown in Table 2. The throughput of our SM4 kernel reaches 535.68 Gbps, it is 12.59 times faster than Martínez-Herrera et al.'s work. The throughput of our GPU card is still satisfactory, and it reaches 76.8 Gbps.

The security strength of AES-128 is same to that of SM4, we compared our SM4 kernel with the fastest AES-128 kernel. Naoki Nishikawa et al. [13] accelerate AES-128 on Tesla P100. We think their hardware has a huge advantage, and our SM4 kernel has better performance on each CUDA core. Considering that Tesla P100 is about 10 times more expensive than GTX 1080, our work is very competitive.

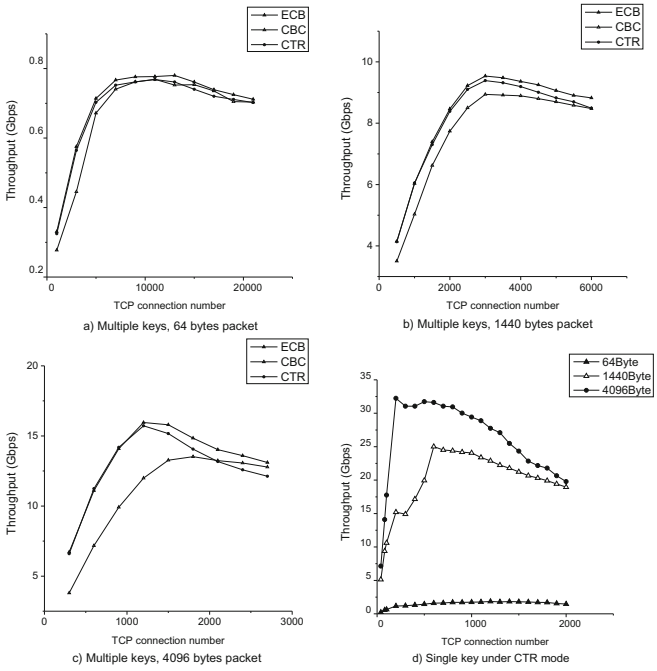

Fig. 5. The throughput of our symmetric cryptography server

5.2 Performance of the Our Server and Comparison with Other GPU Servers

We test the performance of the server when it encrypts different size packets: 64 bytes, 1440 bytes and 4096 bytes packet. Our server is capable of providing CBC, CTR and ECB these three modes of data encryption, and the test result is shown in Fig. 5. As the packet size becomes larger, the peak throughput of the server increases. When the packet size is 4096 bytes, the peak throughput is 15.96 Gbps.

For the case that using a single key under CTR mode, our optimizations raised the throughput of the server almost two to three times. As shown in Fig. 5(d), when using a single key, our server's peak throughput reaches 32.23 Gbps.

In SSLShader [6], they used 2 CPUs and 2 GPUs in a NUMA (Non Uniform Memory Access Architecture) system. NUMA can make a set of CPU and GPU run independently of another set, just like two servers, and SSLShader achieves 13 Gbps symmetric encryption throughput. Our server only used 1 CPU and 1 GPU, the peak throughput of our server is 15.96 Gbps, and for the case that using a single key to do data encryption under CTR mode, our server's throughput reaches 32.23 Gbps, which is 2.48 times faster than SSLShader.

6 Conclusion

In this work, we have presented how to use GPUs to accelerate SM4 encryption. On GeForce GTX 1080, the speed of our SM4 kernel reaches 535.68 Gbps, it is 12.59 times faster than the existing fastest SM4 implementation. We have sped up inefficient data transfer between GPUs and CPUs, and our optimization makes use of 85.4% bandwidth of GTX 1080's PCI-E. We also have showed the potential of using GPUs to enhance the performance of symmetric cryptography servers. Our symmetric cryptography server is capable of providing data encryption under ECB mode, CBC mode and CTR mode. For the case that uses a single key for a long term, like IPSec VPN gateways, the throughput of our server reaches 32.23 Gbps, it is 2.48 times faster than SSLShader

For SM4 symmetric algorithm, the kernel rate is much higher than data transfer rate. While the GPU performs SM4 operations, other algorithms can be operated at the same time to avoid the waste of GPU's computational resources. The inefficiency in the Linux TCP/IP stack is limiting the potential of our server, and Intel's DPDK seems to be a possible solution. These issues are our future work.

References

1. Bolz, J., Farmer, I., Grinspun, E., Schröoder, P.: Sparse matrix solvers on the GPU: conjugate gradients and multigrid. ACM Trans. Graph. (TOG) **22**(3), 917–924 (2003)
2. Erickson, J., Ding, J., Christensen, C.: Algebraic cryptanalysis of SMS4: Gröbner basis attack and SAT attack compared. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 73–86. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14423-3_6

3. Etrog, J., Robshaw, M.J.B.: The cryptanalysis of reduced-round SMS4. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 51–65. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04159-4_4
4. Gilger, J., Barnickel, J., Meyer, U.: GPU-acceleration of block ciphers in the OpenSSL cryptographic library. In: Gollmann, D., Freiling, F.C. (eds.) ISC 2012. LNCS, vol. 7483, pp. 338–353. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33383-5_21
5. Harrison, O., Waldron, J.: Practical symmetric key cryptography on modern graphics hardware. In: USENIX Security Symposium, vol. 2008 (2008)
6. Jang, K., Han, S., Han, S., Moon, S.B., Park, K.: SSLShader: cheap SSL acceleration with commodity processors. In: NSDI (2011)
7. Jiang, Z.H., Fei, Y., Kaeli, D.: A complete key recovery timing attack on a GPU. In: 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 394–405. IEEE (2016)
8. Li, C., Wu, H., Chen, S., Li, X., Guo, D.: Efficient implementation for MD5-RC4 encryption using GPU with CUDA. In: 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication. ASID 2009, pp. 167–170. IEEE (2009)
9. Liu, F., Ji, W., Hu, L., Ding, J., Lv, S., Pyshkin, A., Weinmann, R.-P.: Analysis of the SMS4 block cipher. In: Pieprzyk, J., Ghodosi, H., Dawson, E. (eds.) ACISP 2007. LNCS, vol. 4586, pp. 158–170. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73458-1_13
10. Luken, B.P., Ouyang, M., Desoky, A.H.: AES and DES encryption with GPU. In: ISCA PDCCS, pp. 67–70 (2009)
11. Manavski, S.A.: CUDA compatible GPU as an efficient hardware accelerator for AES cryptography. In: IEEE International Conference on Signal Processing and Communications. ICSPC 2007, pp. 65–68. IEEE (2007)
12. Martínez-Herrera, A.F., Mancillas-López, C., Mex-Perera, C.: GCM implementations of Camellia-128 and SMS4 by optimizing the polynomial multiplier. Microprocess. Microsyst. **45**, 129–140 (2016)
13. Nishikawa, N., Amano, H., Iwai, K.: Implementation of bitsliced AES encryption on CUDA-enabled GPU. In: Yan, Z., Molva, R., Mazurczyk, W., Kantola, R. (eds.) NSS 2017. LNCS, vol. 10394, pp. 273–287. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64701-2_20
14. NVIDIA. CUDA C Programming Guide 8.0 (2017). http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#introduction
15. Pan, W., Zheng, F., Zhao, Y., Zhu, W.-T., Jing, J.: An efficient elliptic curve cryptography signature server with GPU acceleration. IEEE Trans. Inf. Forensics Secur. **12**(1), 111–122 (2017)
16. Zheng, F., Pan, W., Lin, J., Jing, J., Zhao, Y.: Exploiting the floating-point computing power of GPUs for RSA. In: Chow, S.S.M., Camenisch, J., Hui, L.C.K., Yiu, S.M. (eds.) ISC 2014. LNCS, vol. 8783, pp. 198–215. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13257-0_12
17. Zheng, F., Pan, W., Lin, J., Jing, J., Zhao, Y.: Exploiting the potential of GPUs for modular multiplication in ECC. In: Rhee, K.-H., Yi, J.H. (eds.) WISA 2014. LNCS, vol. 8909, pp. 295–306. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15087-1_23