GPU-assisted AES encryption using GCM

Georg Schönberger [*] and Jürgen Fuß[**]

Upper Austria University of Applied Sciences
Dept. of Secure Information Systems
Softwarepark 11, 4232 Hagenberg
georg.schoenberger@students.fh-hagenberg.at
juergen.fuss@fh-hagenberg.at
http://www.fh-ooe.at/sim

Abstract. We are presenting an implementation of the Galois/Counter Mode (GCM) for the Advanced Encryption Standard (AES) in IPsec in this paper. GCM is a so called "authenticated encryption" as it can ensure confidentiality, integrity and authentication. It uses the Counter Mode for encryption, therefore counters are encrypted for an exclusive-OR with the plaintext. We describe a technique where these encryptions are precomputed on a Graphic Processing Unit (GPU) and can later be used to encrypt the plaintext, whereupon only the exclusive-OR and authentication part of GCM are left to be computed. This technique should primarily not limit the performance to the speed of the AES implementation but allow Gigabit throughput and at the same time minimize the CPU load.

Keywords: AES, Galois/Counter Mode (GCM), IPsec, GPU, CUDA, Gbit/s, high-performance

1 Introduction

Todays need of high-performance cryptography implementations is rapidly increasing. As the most calculating intensive part often belongs to an encryption or hashing algorithm, the overall performance of a system highly depends on the speed of the underlying cryptography. To increase performance developers optimize the algorithms tailored for several CPU-architectures. Alternatively one can implement parallel versions of the algorithms using multiple CPU-cores or via GPUs. Especially with the "Compute Unified Device Architecure" (CUDA) and the programming language "C for CUDA" GPUs gained a boost in popularity [1]. An extremely important area concerning speed of cryptographic operations is in the protection of network traffic. In protocol suites like IP Security (IPsec) [6], AES has become a favourite encryption scheme to ensure data confidentiality. In terms of the encryption mode the Counter mode (CTR) of operation is

[*] Funded through a private research by Barracuda Networks, Inc.

[**] Funded by the KIRAS programme of the Austrian Federal Ministry for Transport, Innovation and Technology.

preferred for high-speed connections as it can be implemented in hardware and allows pipelining and parallelism in software [8].

In this paper we focus on GCM (cf. [2] and [17]) and its usage as a mode for Encapsulation Security Payload (ESP) in IPsec (standardised in RFC 4106 [5]) and constitute a new way to compute it:

– We show that—accepting slight modifications of the standards—it is possible to precompute the AES-CTR part of AES-GCM without reducing security.
– As a practical example we show a prototype using GPUs for the precomputation that puts our ideas into practice. First of all this prototype shows operational reliability, moreover we considered some benchmarks.
– The resulting challenges posed by the introduced architecture are analysed. At the same time proposals for problem solutions are discussed as well as their practicability and impacts will be observed.

1.1 Related Work

The implementation of AES-GCM on GPUs itself has not yet been a common topic in research. GCM is defined in the NIST Special Publication 800-38D [2] and used as the default cipher mode in the IEEE Standard 802.1AE—Media Access Control (MAC) Security [3], mainly due to its good performance. Another aspect why GCM in combination with AES can be of value is the fact that the Intel Cooperation provides theirs own instruction set for implementing AES on the Westmere architecture. The most famous set is called "AES-NI" that consists of six new instructions to realize the AES-algorithm [9]. Moreover, there is an instruction called "PCLMULQDQ" for carry-less multiplication which can be used to increase the performance of the universal hash function "GHASH" in GCM [10]. In an AES-NI-GCM implementation within the Linux kernel Intel has also carefully examined the performance gains of these instructions compared to a traditional implementation [11].

The fast development and enhancements of CUDA pushed new and innovative applications on GPUs. One of the first AES implementations using CUDA was by Svetlin A. Manavski who also parallelised AES on instruction level using four threads to produce the 16 encrypted bytes [12]. Other papers followed analysing how to realise AES with CUDA efficiently. A good example of how to examine how well AES can perform on a GPU is the master thesis of Alexander Ottesen [13]. He analysed the differences of AES with CUDA by first using the traditional processing way and then compare this to a lookup table version. He also tried to optimize these applications by fully utilising the different memory spaces of a GPU.

There also have been some recent approaches to use AES on GPUs for tasks in networking environments. The researchers of the "Networked & Distributed Computing Systems Lab" in Korea released papers about how to speed-up SSL [15] or how to accelerate a software router—also in connection with IPsec [16]—with GPUs.

2 Our Approach

In our work we combine the potential of GPUs[1] to accelerate AES with the benefits of AES-GCM as authenticated encryption. In contrast to recent works on only AES-CTR using CUDA ([13], [14]) we split the AES-GCM into two separate stages. The first realizes the encryption part on the GPU[2] and the second stage is responsible for the authentication part. Moreover stage one needs not to be executed with stage two at the same time, as the plaintext is only indispensable at authentication time. This separation is a completely new way to implement AES-GCM's authenticated encryption.

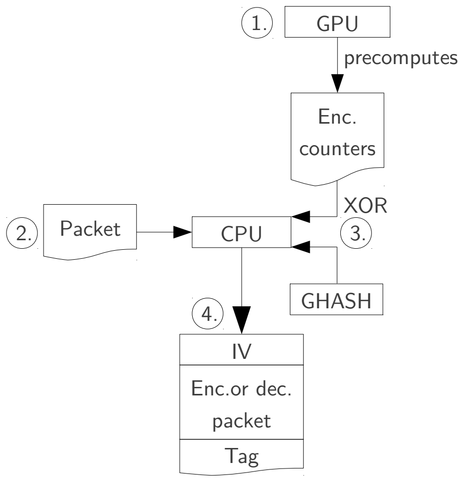

Fig. 1. Schematic procedure of en-/decrypting a packet.

In Fig. 1 we show how we en-/decrypt a packet using AES-GCM with precomputation:

1. As soon as the secret-key and the initialization vector ("Nonce") for AES-GCM are negotiated, we start encrypting continuing counters with the key on the GPU.
2. As soon as packets are arriving the CPU can start en-/decrypting it. Additional Authentication Data (AAD) can be added as well (q.v. [17, sec. 2.3]).
3. The CPU uses the precomputed counters generated by the GPU to XOR with the plaintext for encryption and for authentication GHASH (universal hashing over a binary Galois field [2, Chap. 5]).

[1] We would like to emphasize the fact that the precomputation described here need not be done on a GPU. Also free cores on a multi-core CPU may be used for this purpose. Nevertheless, in this paper we refer to the processor that does the precomputation as the GPU.

[2] At this point we can use the results of the recent papers on AES with CUDA.

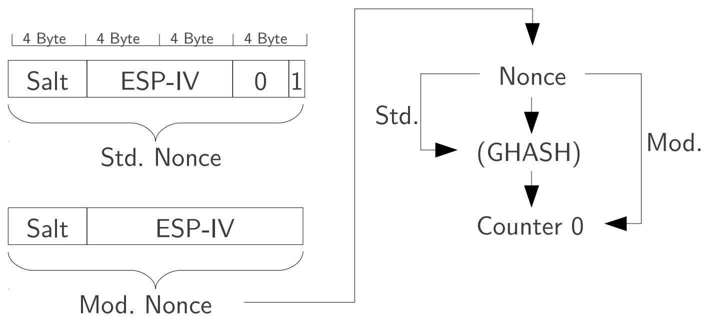

Fig. 2. Modifications for the construction of the nonce and the usage of GHASH for non-12 byte IVs.

4. After a packet is encrypted and the tag is computed, the packet can be composed. In case of decryption the computed tag must be compared with the encrypted packet's original tag.

To use the counters generated by the GPU efficiently we have to adapt the format of initialization vectors (IV) for AES-GCM. A nonce for AES-GCM consists of the salt (4 bytes) generated by Internet Key Exchange Protocol (IKE) [4] and the ESP-IV (8 bytes) [5, p. 4]. To form the initial counter for AES-GCM this 12 byte sequence is padded with the 32 bit sequence "`00...1`" to become 16 byte long [17, p. 5] (see Fig.2). For the subsequent packet the ESP-IV is incremented by one, the salt prefixed and then again padded. This usage of the ESP-IV and the padding is not suited for precomputation as we cannot use a continuous stream of encrypted counters. We only need **one** counter getting incremented and not two (ESP-IV per packet and the padded-nonce within a packet). Therefore we suggest the following changes for the construction of the nonce (cf. Fig. 2):

– Extension of the ESP-IV to 12 bytes so that padding is not necessary. Otherwise we would have to estimate the size of a packet and increase the initial counter to generate enough encrypted data for the XOR. This would go along with a large number of encrypted counters that must discarded or missing counters for long packets. That's why we use the salt and the 12 byte ESP-IV as the initial counter for AES-GCM so as to have one counter and no estimation of the packet sizes is needed.
– Normally, GHASH is applied for nonces not equal to 12 bytes. Due to the new ESP-IV length GHASH should be used to form the 16 byte initial counter. Again this usage of GHASH does not allow us to predict what the initial counter for the next packet will be. For this reason we propose to skip GHASH for nonces of 16 bytes. As long as one counter is not encrypted with the same key more than once, this has no security impact.

A modification of the format of the IV in a counter mode may have negative impact on the overall security of the encryption scheme. We claim that our modifications do not reduce the security of the encryption method. Two issues must be considered.

- Firstly, the original IV format makes sure that IVs will not repeat, neither in one packet (using a 4 byte per packet counter) nor in different packets (using a packet dependant 12 byte ESP-IV). The modified format can also guarantee unique IVs for AES-GCM; inside a packet as well as in different packets with the same 16 byte counter. Finally, a regular rekeying in the IPsec protocol will prevent this 16 byte counter.
- Secondly, we do not use GHASH on a non-12 byte IV. With respect to this modification it can be said that the purpose of GHASH in this case is to have a simple method to guarantee as an output an IV that can be used for AES-GCM. The standard also skips the GHASH in the case of a 12 byte IV (for higher speed).

3 Challenges

As a result of precomputation it is essential to schedule encryption—in matters of XOR and GHASH—asynchronously to the encryption of counters. This means the need for a parallel implementation of encrypting and authenticating packets while new counters for the subsequent packets are encrypted. From this it follows that the overall performance of our AES-GCM depends primarily on the speed of GHASH and `memcpy` operations to fetch the precomputed encrypted counters. If we reach the point where we can process a packet as fast or even faster as we can precompute, then again the limiting factor is the AES encryption of the counters.

In conjunction with GPUs as coprocessors the challenging task is the transfer of the encrypted counters from the GPU to RAM. One might also consider that one GPU could precompute counters for several IPsec connections. Then the management of the GPU as a shared resource is as important as the handling of parallel processing packets and precomputing. As a solution for the management of precomputed counters the CPU can provide double buffers. So, if one buffer gets empty, a switch to the second buffer can be performed and the empty buffer gets refilled asynchronously by the GPU. Sometimes network packets can get lost or mixed up on their way to the receiver. In this case counters from the previous buffer are needed. In this case we can insert some sort of "history" at the beginning of each buffer that contains a certain amount of old counters from the buffer just been overwritten.

4 Benchmarks

We used our implementations on the GPU to test if the precomputation can speed up an existing AES-GCM application. As current AES-implementations with CUDA nearly reach the 10 Gbit/s [13][16] we focused in the first place on the interaction of CPU and GPU. For our benchmarking we patched the

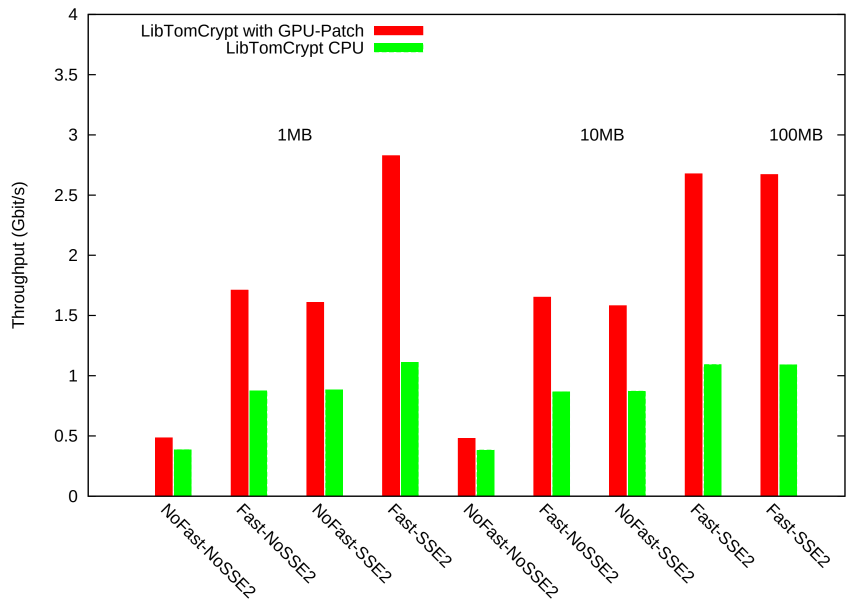

Fig. 3. Comparison of the encryption throughputs: (i) LibTomCrypt on the CPU and (ii) LibTomCrypt with our GPU-Patch (both with and without LTC_FAST and SSE2 instructions).

AES-GCM algorithm of the cryptographic toolkit “LibTomCrypt”[3]. The patch included the replacement of encrypting counters with fetching counters from precomputed memory. As precomputation can be done in parallel to encryption the throughput depends on the time taken by copying encrypted counters and performing GHASH.

In Fig. 3 we compare the runtime of encrypting 1, 10 and 100 Megabyte (MB) of random data. To simulate the encryption of network packets we process the data in chunks of 1024 Byte as this could be a suitable packet size. Fig. 3 shows that our patch is faster in every combination of used instruction sets and for all sizes of data. Also note the fact that with activated LTC_FAST and SSE2 our version has more than twice the throughput of the traditional LibTomCrypt library. We also examined the statistical spread of the runtimes by conducting the tests a thousand times. As a result the variation is negligible because only sporadic outliers were detected with a variation around one millisecond.

We used an Intel Core i7 940, 12GB 1333MHz RAM and a GTX480 on an Intel DX58SO Motherboard for testing. The operating system was Ubuntu 10.4 and for compiling LibTomCrypt in version 1.17 we used gcc-4.4.

[3] Available under public domain at `http://libtom.org/`.

5 Conclusion

Our implementation of AES-GCM separated into stages shows that this mode of operation has benefits from a cryptographic point of view and also solves performance issues. The fact that the coprocessors can perform encryption in parallel to processing data offers new challenges to high-performance network applications. Current standards for the use of this mode in IPsec are not perfectly suitable for an implementation with precomputation. However, only small modifications are necessary and they do not affect security. We are looking forward how well an implementation running in kernel-mode will perform. Finally, a comparison of AES-GCM with classical combinations of block cipher and hash function will be an important next step.

References

1. NVIDIA Corporation: NVIDIA CUDA C Programming Guide, Developer Manual (2010), `http://developer.nvidia.com/object/gpucomputing.html`
2. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-38D (2007)
3. IEEE Computer Society: Standard for Local and metropolitan area networks: Media Access Control (MAC) Security, New York (2006)
4. Kaufman, C.: Internet Key Exchange (IKEv2) Protocol, RFC 4306 (2005)
5. Viega, J. and D. McGrew: The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP), RFC 4106 (2005)
6. Kent, S. and K. Seo: Security Architecture for the Internet Protocol, RFC 4301 (2005)
7. Kent, S.: IP Encapsulating Security Payload (ESP), Request for Comments 4303 (2005)
8. Dworkin, M.: Recommendation for Block Cipher Modes of Operation: Methods and Techniques, NIST Special Publication 800-38A (2001)
9. Akdemir, K. e.a.: Breakthrough AES Performance with Intel AES New Instructions, Intel Whitepaper (2010), `http://software.intel.com/file/27067`
10. Gopal, V. e.a.: Optimized Galois-Counter-Mode Implementation on Intel Architecture Processors, Intel Whitepaper (2010), `http://download.intel.com/design/intarch/PAPERS/324194.pdf`
11. Hoban, A.: Using Intel AES New Instructions and PCLMULQDQ to Significantly Improve IPSec Performance on Linux, Intel Whitepaper (2010), `download.intel.com/design/intarch/papers/324238.pdf`
12. Manavski, S.A.: Cuda compatible GPU as an efficient hardware accelerator for AES cryptography, In Proceedings IEEE International Conference on Signal Processing and Communication, ICSPC (2007)
13. Ottesen, A.: Efficient parallelisation techniques for applications running on GPUs using the CUDA framework, Universitt Oslo (2009), `http://www.duo.uio.no/sok/work.html?WORKID=91432`
14. Di Biagio, A. and Barenghi, A. and G. Agosta: Design of a Parallel AES for Graphics Hardware using the CUDA framework, Parallel and Distributed Processing Symposium, International (2009)

15. Jang, K. e.a.: SSLShader: Cheap SSL Acceleration with Commodity Processors, In Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (2011)
16. Han, S. e.a.: PacketShader: a GPU-Accelerated Software Router, In Proceedings of ACM SIGCOMM (2010)
17. McGrew, D. A. and J. Viega: The Galois/Counter Mode of Operation (GCM) - revised, Technical Report (2005), `http://www.csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-revised-spec.pdf`