Brief Contributions

Packed AES-GCM Algorithm Suitable for AES/PCLMULQDQ Instructions

Krzysztof Jankowski and Pierre Laurent

Abstract—The level of interest in Galois Counter Mode (GCM) Authenticated Encryption rose significantly within the last few years. GCM is interesting because it is the only authenticated encryption standard that can be implemented in a fully pipelined or parallelized way and it is the most appropriate for encrypting packetized data. McGrew and Viega [1] described (but did not detail) how GHASH can be implemented with more than one multiplier operating in parallel. This paper details how that can be done and shows that, when N multipliers are used, and the multipliers use the approach of multiplying polynomials then applying a modular reduction, a single modular reduction can be used instead on N separate operations. This optimization can be used even when there is a single multiplier, which makes this implementation strategy have a broader appeal. Recently Intel has introduced new ISA instructions into the next generation CPU core, namely: AES family and PCLMULQDQ operating in XMM registers domain. In this paper, we discuss the example implementation of proposed GHASH modifications using above instructions.

Index Terms—Software, data encryption, AES, GCM, performance evaluation of algorithms.

1 Introduction

1.1 AEC-GCM Overview

AES-GCM algorithm is a combination of AES Counter Mode encryption with Galois Hash authentication—authenticated-encryption. It produces an encrypted data payload and an authentication tag in one data pass. Full AES-GCM implementation includes three phases: Preprocessing (encryption of Counter0, authentication of Additional Authentication Data); Processing Loop; and Post Processing—see Fig. 1.

In this paper, we will solely focus on the second step of the full AES-GCM implementation, an efficient implementation of the main data processing loop. For detailed description of entire AES-GCM algorithm implementations, please refer to [1] and [2].

1.2 Encryption

In AES-GCM mode, the underlying encryption algorithm is based on AES block cipher working in Counter Mode (CTR). AES defines three block ciphers: AES-128, AES-192, and AES-256. Each AES cipher has a constant 128-bit block size, with key sizes of 128, 192, and 256 bits. In CTR mode, every encryption step comprises of a 128-bit counter value increment, counter encryption with key K (key stream generation), and finally XOR'ing the encrypted counter with plaintext data to produce the ciphertext (top of Fig. 1). A property of CTR mode is that the counter values are independent and they can be incremented and encrypted in parallel.

1.3 Authentication

GCM authentication is based on a GHASH function, a multiplication in the $\mathrm{GF}(2^{128})$ field of two factors giving a 128-bit product—an authentication tag.

• *The authors are with Intel, Dromore House, East Park, Shannon, Co. Clare, Ireland. E-mail: {krzysztof.jankowski, pierre.laurent}@intel.com.*

Manuscript received 12 Nov. 2009; revised 29 Jan. 2010; accepted 16 Feb. 2010; published online 11 June 2010.
Recommended for acceptance by M. Eltoweissy.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2009-11-0571.
Digital Object Identifier no. 10.1109/TC.2010.147.

In order to get a 128-bit GHASH authentication result, $128\text{-bit} \cdot 128\text{-bit} = 256\text{-bit}$ binary multiplication has to be first calculated and then 256-bit number has to be reduced modulo predefined binary polynomial: $x^{128} + x^7 + x^2 + x + 1$. Note that above non-trivial calculations have to be performed for every 128-bit data block coming from the encryption phase (bottom of Fig. 1).

An important fact is that in the GCM algorithm case, one of the factors of the binary multiplication is a constant 128-bit value (H variable—hash key, in rounded MULT box in Fig. 1) for a given session.

It is worth noting that there are implementation strategies for $\mathrm{GF}(2^{128})$ multiplication which do not use a separate modular reduction stage (such as [1, Section 4.1, (6)]). GHASH method, described in this paper, is independent of underlying multiplication stage implementations, therefore, these strategies should benefit from proposed modifications as well.

2 Proposed Modifications

In order to reduce the per-block computational latency, we propose to process N 128-bit data blocks (block set) in single-loop iteration. Note that the exact number of simultaneously processed blocks depends on the number of available registers and cache size. This approach by its nature is designed for larger data payloads.

Our proposed method allows also for pipelining (or even parallelization in case of out of order execution machines) of encryption and authentication calculations assuming they are executed by separate hardware units. Fig. 2 depicts the data flow of the proposed approach.

2.1 Processing Loop

The highest performance improvements come from overlapping of two independent processing phases (encrypt and authenticate) and reduced computational complexity of $\mathrm{MULT\_N_H}$ module.

The precompute step is required for both encryption $\mathrm{AES\_N_K}$ and authentication $\mathrm{MULT\_N_H}$ modules before entering the main loop. Precomputation will get executed once per authenticated-encryption session setup only.

2.2 Encryption

CTR mode is pipeline-friendly encryption mode, as cipher execution unit(s) can be saturated with a number of independent encryption requests.

2.2.1 Precompute

AES cipher implementation requires encryption key K to be expanded into a number of separate round keys by using Rijndael's key scheduling algorithm [5].

2.2.2 Data Processing

AES encryption converts 128-bit cleartext block into ciphertext block by executing a number of complex transformation rounds [5]. There is a strict data dependency between one transformation round and another working with the same data block. But there is no dependency when a number of data blocks (N in our case) are transformed within the same round [5].

2.3 Authentication

Section 1.3 of this paper describes the generic $\mathrm{GHASH_H}$ calculation, which requires binary multiplication and reduction to be performed on every 128-bit encrypted data block. Authentication of N data blocks at a time allows for compute savings, especially in terms of modular reduction. Proposed algorithm "packs" N intermediate results into one, and then performs single compute intensive calculation at the end.

Authorized licensed use limited to: Chengdu University of Technology. Downloaded on May 15,2024 at 00:54:20 UTC from IEEE Xplore. Restrictions apply.
0018-9340/11/$26.00 © 2011 IEEE Published by the IEEE Computer Society

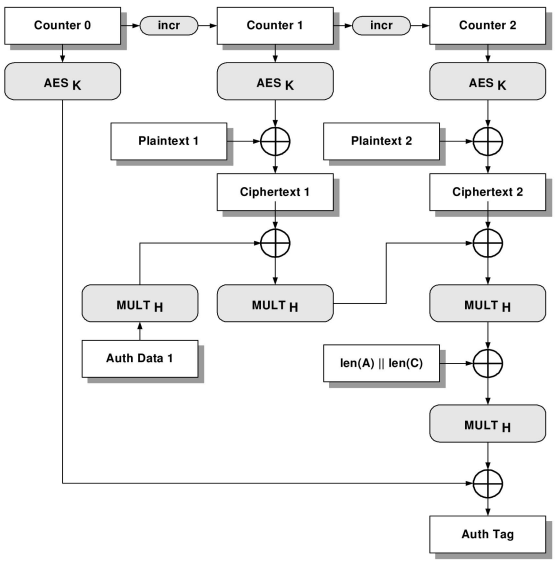

Fig. 1. AES-GCM mode.

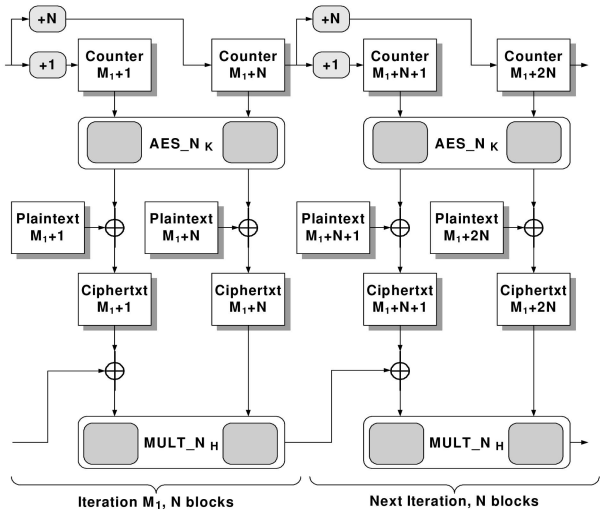

Fig. 2. Pipelined AES-GCM.

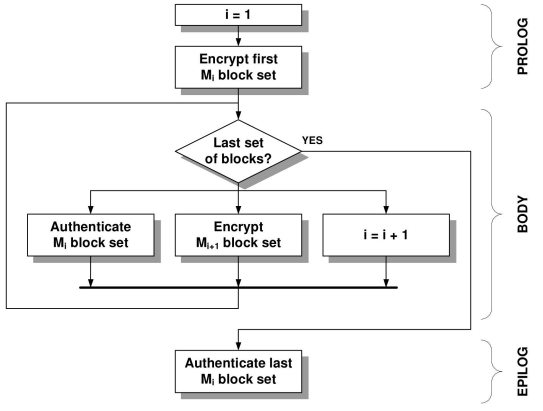

Fig. 3. Data processing loop.

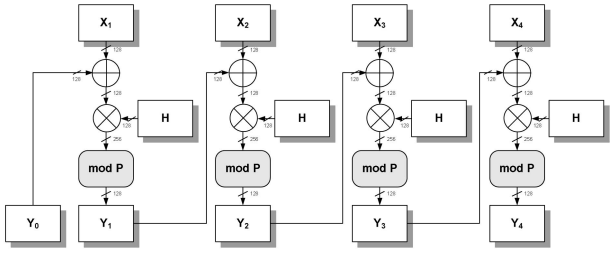

Fig. 4. Standard $\mathrm{GHASH_H}$ function.

2.3.1 Packed $GHASH_H$ Formula

Let's introduce math symbols allowing for the $\mathrm{GHASH_H}$ formula definition:

- $\mathrm{X_i}$ ith 128-bit data block
- $\mathrm{Y_i}$ ith 128-bit GHASH result
- H hash key, 128-bit value
  $\mathrm{H} = \mathrm{AES_K}(0^{128})$
  $\mathrm{H}^2 = (\mathrm{H} \otimes \mathrm{H}) \bmod \mathrm{P}$
  $\mathrm{H}^3 = (\mathrm{H} \otimes \mathrm{H} \otimes \mathrm{H}) \bmod \mathrm{P}$
- P field polynomial degree 128:
  $\mathrm{P} = \mathrm{P(x)} = \mathrm{x}^{128} + \mathrm{x}^7 + \mathrm{x}^2 + \mathrm{x} + 1$
- $+$ carry-less GF(2) addition (XOR)
- $\otimes$ carry-less GF(2) multiplication

$\mathrm{GHASH_H}$ function for block "i" is defined as follows [2]:

$$\{\mathrm{Y_i} = [(\mathrm{X_i} + \mathrm{Y_{i-1}}) \otimes \mathrm{H}]\} \bmod \mathrm{P}.$$

Packed $\mathrm{GHASH_H}$ function for authenticating, e.g., four data blocks at a time ($\mathrm{N} = 4$), corresponding to diagram on Fig. 4 can be obtained from below calculations:

$$\begin{aligned}\{\mathrm{Y_i} &= [(\mathrm{X_i} + \mathrm{Y_{i-1}}) \otimes \mathrm{H}] \\ &= [(\mathrm{X_i} \otimes \mathrm{H}) + (\mathrm{Y_{i-1}} \otimes \mathrm{H})] \\ &= (\mathrm{X_i} \otimes \mathrm{H}) + [(\mathrm{X_{i-1}} + \mathrm{Y_{i-2}}) \otimes \mathrm{H}^2] \\ &= (\mathrm{X_i} \otimes \mathrm{H}) + (\mathrm{X_{i-1}} \otimes \mathrm{H}^2) + [(\mathrm{X_{i-2}} + \mathrm{Y_{i-3}}) \otimes \mathrm{H}^3] \\ &= (\mathrm{X_i} \otimes \mathrm{H}) + (\mathrm{X_{i-1}} \otimes \mathrm{H}^2) + (\mathrm{X_{i-2}} \otimes \mathrm{H}^3) \\ &\quad + [(\mathrm{X_{i-3}} + \mathrm{Y_{i-4}}) \otimes \mathrm{H}^4]\} \bmod \mathrm{P}.\end{aligned}$$

The main advantage of the Packed $\mathrm{GHASH_H}$ definition is that it requires only one "mod P" modular reduction per N 128-bit of data, and that storage of temporary $\mathrm{Y_i}$ results is not required.

2.3.2 Precompute

For a given N value (block set size), N powers of H modulo P have to be precalculated. As with AES precomputations, it has to be done at the session setup time only because $\mathrm{H} = \mathrm{AES_K}(0^{128})$. For that purpose, use formula: $\mathrm{H^j} = (\mathrm{H^{j-1}} \otimes \mathrm{H}) \bmod \mathrm{P},\ \mathrm{j} = 2 \ldots \mathrm{N}$.

2.3.3 Data Processing

Authentication of N blocks at a time requires N $128\text{-bit} \cdot 128\text{-bit} = 256\text{-bit}$ carry-less multiplications, N 128-bit XOR operations, and one 128-bit modular reduction. For architectures where the carry-less multiplication word is smaller than 128-bit, the Karatsuba method should be investigated.

3 Implementation with PCLMULQDQ

In order to verify the correctness and potential speed improvements of proposed method, two AES-GCM functions were implemented. Both were written in a handcrafted 64-bit assembler, which allowed taking advantage of all 16 available 128-bit XMM registers. Also, both used Intel's AES instructions family to implement AES-GCM's encryption phase. AES-GCM function with "Standard $\mathrm{GHASH_H}$" followed implementation guidelines described in [1] ($\mathrm{N} = 1$, one 128-bit block multiply and one reduction). AES-GCM with "Packed $\mathrm{GHASH_H}$" implemented method proposed in this paper with block set size $\mathrm{N} = 4$ (four

Authorized licensed use limited to: Chengdu University of Technology. Downloaded on May 15,2024 at 00:54:20 UTC from IEEE Xplore. Restrictions apply.

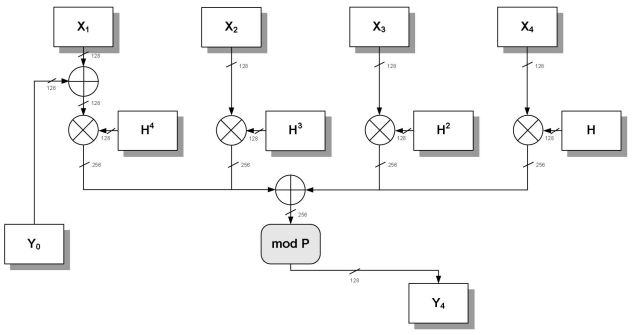

Fig. 5. Packed $\mathrm{GHASH_H}$ function.

128-bit multiplications and one reduction, 64 B encrypted and authenticated in a single-loop's iteration).

The development environment consisted of Intel SDE [7], YASM assembler [8], and MS Visual Studio 2005.

3.1 $128\text{-Bit} \times 128\text{-Bit}$ Multiplication

The PCLMULQDQ instruction performs a carry-less multiplication on two 64-bit operands and gives a 128-bit result. In order to compute the carry-less product of two 128-bit operands with PCLMULQDQ, we used the carry-less Karatsuba algorithm (see [6]) to get the required 256-bit result. A total of 55 SSE instructions were required for performing four $(\mathrm{N} = 4)$ $128\text{-bit} \cdot 128\text{-bit}$ multiplications, and merging each result into a final 256-bit number (compare with Fig. 5).

3.2 Bit Reflection

The GCM standard [1] specifies that the bits inside their 128-bit double quad words are reflected (compare the declaration of pentanomial P in Section 2.3.1 with its numerical representation in Section 6 "Test Vectors"). For the detailed discussion about the bit reflection peculiarity, please refer to [6]. Following [6], our implementation shifted the 256-bit carry-less product to the left by one bit before the modular reduction, in order to account for that phenomenon (15 SSE instructions added).

3.3 Modular Reduction

Modular reduction utilizes a fixed GF(2) pentanomial P, therefore, for that purpose, an implementation based on Linear Folding and hard-coded bit shifts was chosen [6]. A hard-coded pentanomial implementation required 28 SSE instructions. Modular reduction for a generic reduction polynomial P based on Horner's representation of polynomials and folding (256-bit -> 192-bit, then 192-bit -> 128-bit) utilizing PCLMULQDQ was implemented, but proved to be slower.

4 Instructions Count

Fig. 6 shows the total number of IA instructions as included in prolog, body, and epilog code sections for the two AES-GCM functions implemented. Depending on payload size (always multiple of 64 bytes, in this case), the body of the function will get executed zero or more times (code sections as in Fig. 3).

Section	Standard	Packed
Prolog	47	95
Body	90	188
Epilog	74	116

Fig. 6. IA CPU instructions.

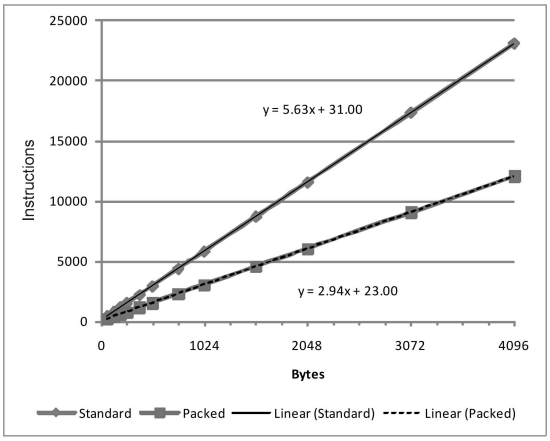

Fig. 7. Total executed instructions.

4.1 AES-GCM with Standard $\mathrm{GHASH_H}$

Bytes	Blocks	Body Runs	Instructions
64	4	3	391
128	8	7	751
192	12	11	1111
256	16	15	1471
384	24	23	2191
512	32	31	2911
768	48	47	4351
1024	64	63	5791
1536	96	95	8671
2048	128	127	11551
3072	192	191	17311
4096	256	255	23071

4.2 AES-GCM with Packed $\mathrm{GHASH_H}$

Bytes	Blocks	Body Runs	Instructions
64	4	0	211
128	8	1	399
192	12	2	587
256	16	3	775
384	24	5	1151
512	32	7	1527
768	48	11	2279
1024	64	15	3031
1536	96	23	4535
2048	128	31	6039
3072	192	47	9047
4096	256	63	12055

5 Summary

As shown in Fig. 7, Packed AES-GCM implementation processes the same payload in $\sim 1.9\times$ less instructions.

Packed AES-GCM implementation proposed in this paper proved to be very efficient. However, further optimizations are possible and are under investigation (e.g., instructions count reduction for runtime dealing with bit reflection peculiarity; modular reduction utilizing PCLMULQDQ instruction).

Authorized licensed use limited to: Chengdu University of Technology. Downloaded on May 15,2024 at 00:54:20 UTC from IEEE Xplore. Restrictions apply.

6 Test Vectors

Variable	Value
P	E1000000000000000000000000000000
K	feffe9928665731c6d6a8f9467308308
H	b83b533708bf535d0aa6e52980d53b78
H^2	8a6ff5aca561c0d865805055eb728397
H^3	c414cb8f1152eb71563a5ca9ddcbddb5
H^4	3c4b0daa91e6b35f9b9e89d8510dd431

References

[1] D. McGrew and J. Viega, "The Galois/Counter Mode of Operation (GCM)," http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf, May 2005.
[2] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf, Nov. 2007.
[3] J. Viega and D. McGrew, "The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)," RFC 4106, http://www.ietf.org/rfc/rfc4106.txt, June 2005.
[4] M. Badra, "Pre-Shared Key Cipher Suites for TLS with SHA-256/384 and AES Galois Counter Mode," RFC 5487, http://www.ietf.org/rfc/rfc5487.txt, Mar. 2009.
[5] S. Gueron, "Advanced Encryption Standard (AES) Instructions Set," Intel Software Network, http://software.intel.com/en-us/articles/advanced-encryption-standard-aes-instructions-set, Apr. 2009.
[6] S. Gueron and M. Kounavis, "Carry-Less Multiplication and Its Usage for Computing the GCM Mode," Intel Software Network, http://software.intel.com/en-us/articles/carry-less-multiplication-and-its-usage-for-computing-the-gcm-mode, May 2009.
[7] Intel Software Network, "Intel Software Development Emulator," http://software.intel.com/en-us/articles/intel-software-development-emulator, Mar. 2009.
[8] The Yasm Modular Assembler Project, http://www.tortall.net/projects/yasm/, 2010.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.

Authorized licensed use limited to: Chengdu University of Technology. Downloaded on May 15,2024 at 00:54:20 UTC from IEEE Xplore. Restrictions apply.