Workshop on Real-World Cryptography
Stanford University Jan. 9-11, 2013

AES-GCM for Efficient Authenticated Encryption – Ending the Reign of HMAC-SHA-1?

Shay Gueron

University of Haifa
Department of Mathematics, Faculty of Natural Sciences, University of Haifa, Israel

Intel Corporation
Intel Corporation, Israel Development Center, Haifa, Israel

shay@math.haifa.ac.il, shay.gueron@intel.com

Agenda

- Why is the ecosystem using HMAC SHA-1 for authenticated encryption?
  - What can be done to change this?
- AES-GCM dirty secrets… and how to optimize it

(*… and save the honor of AES-GCM after Adam's talk*)

Optimizing cryptographic primitives

- Why care? Who cares?
  - The need for end-to-end security in the internet, constantly increases the world-wide number (and percentage) of SSL/TLS connections.
  - Why aren't all connections http**s**:// ? **Overheads' costs**
    - Cryptographic algorithms for secure communications = computational overhead
    - Mainly on the servers side
  - Any latency client side influences (indirectly) the ecosystem
- Authenticated Encryption: a fundamental cryptographic primitive
- Is the ecosystem using an efficient AE scheme?
  - Apparently no… **a better alternative exists**

Ciphers in use in SSL/TLS connections

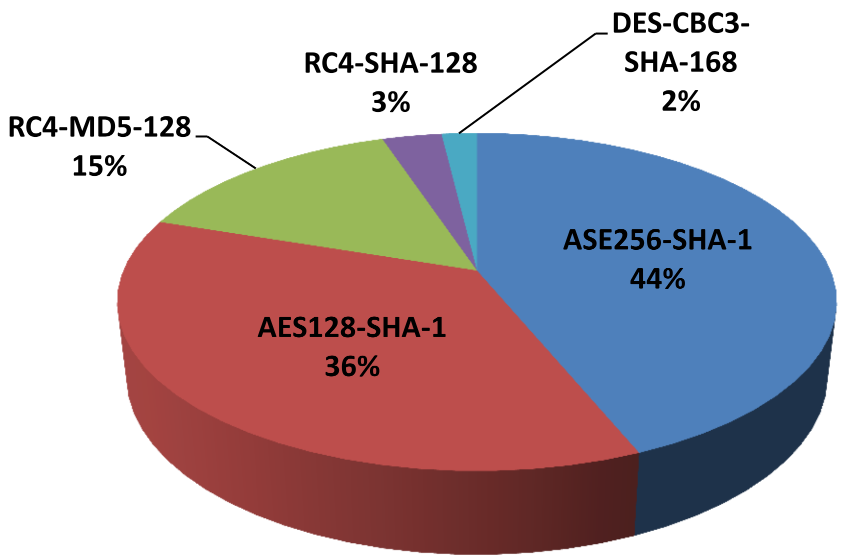
Today's most frequently used AE in browser/server connections
RC4 + HMAC-MD5 (don't care)
RC4 + HMAC-SHA-1
AES + HMAC-SHA-1

➔ authentication: mostly HMAC SHA-1

Is it the best AE (performance wise)?

No – a faster alternative exists

We already know that HMAC is not an efficient MAC scheme, and as an ingredient in AE – it makes an inefficient AE



- Akamai serves service millions of requests per sec. for secure content over HTTPS/SSL
- Observed the client-side SSL ciphers in popular use
- Statistics for SSLv3 and TLSv1
- http://www.akamai.com/stateoftheinternet

AES-GCM is a more efficient Authenticated Encryption scheme

AES-GCM Authenticated Encryption

- AES-GCM Authenticated Encryption (D. McGrew & J. Viega)
  - Designed for high performance (Mainly with a HW viewpoint)
  - A NIST standard FIPS 800-38D (since 2008)
    - Included in the NSA Suite B Cryptography.
- Also in:
  – IPsec (RFC 4106)
  – IEEE P1619 Security in Storage Working Group http://siswg.net/
  - TLS 1.2
- How it works:
  - Encryption is done with AES in CTR mode
  - Authentication tag computations - “Galois Hash” :
    - A Carter-Wegman-Shoup universal hash construction: polynomial evaluation over a binary field
    - Uses $GF(2^{128})$ defined by the “lowest” irreducible polynomial

$$g = g(x) = x^{128} + x^7 + x^2 + x + 1$$

  - Computations based on $GF(2^{128})$ arithmetic

But not really the standard $GF(2^{128})$ arithmetic

AES-GCM and Intel's AES-NI / PCLMULQDQ

- Intel introduced a new set of instructions (2010)
- AES-NI:
  - Facilitate high performance AES encryption and decryption
- PCLMULQDQ **64 x 64 → 128 (carry-less)**
  - Binary polynomial multiplication; speeds up computations in binary fields
- Has several usages --- AES-GCM is one
- To use it for the GHASH computations: $GF(2^{128})$ multiplication:
  1. Compute **128 x 128 → 256** via carry-less multiplication (of 64-bit operands)
  2. Reduction: **256 → 128 modulo** $x^{128} + x^7 + x^2 + x + 1$ (done efficiently via software)

It ain't necessarily so

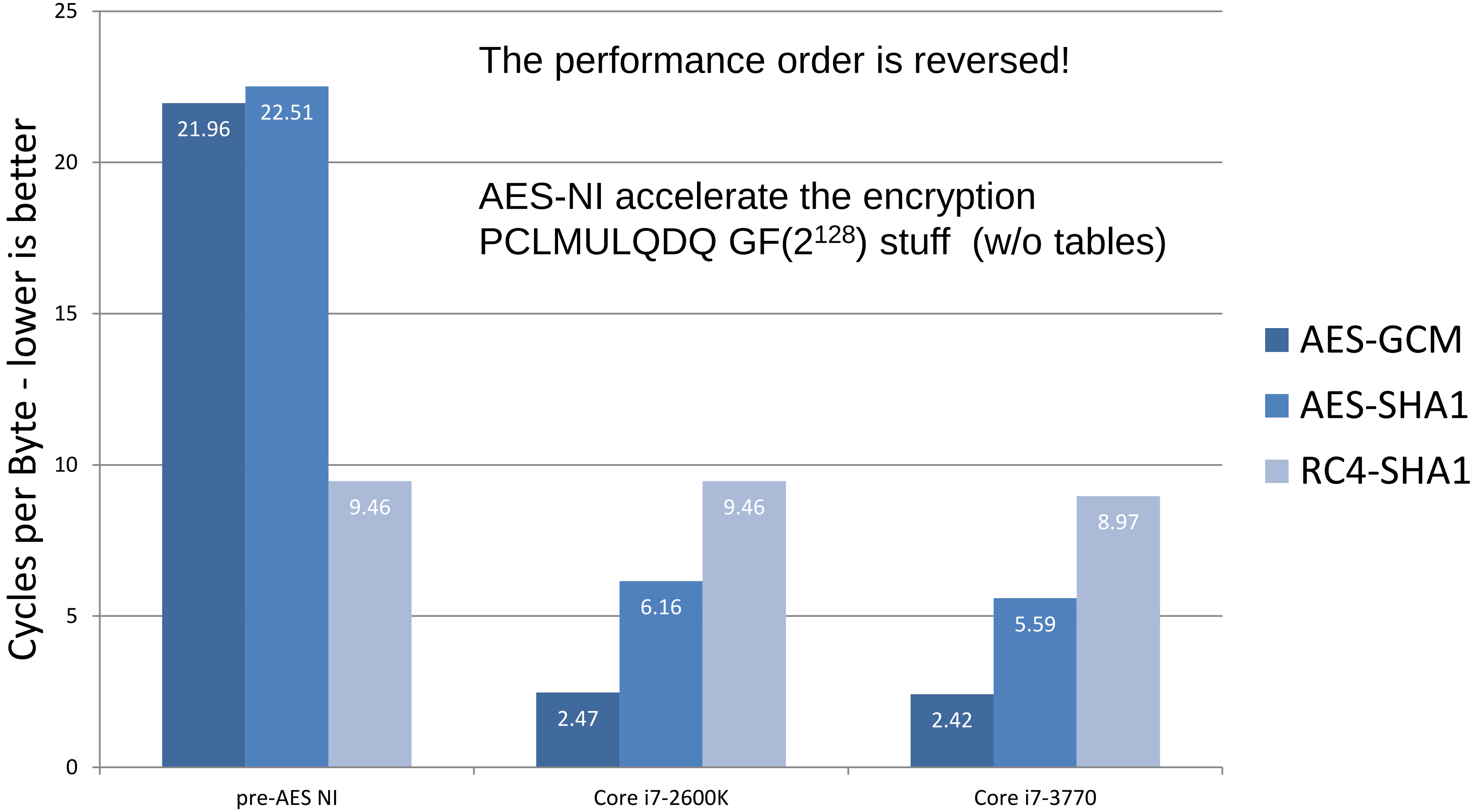
AES-NI and PCLMULQDQ can be used for speeding up AES-GCM Authenticated Encryption

Some Authenticated Encryption performance

PRE AES-NI / CLMUL(lookup tables)

- RC4 + HMAC SHA-1
- AES + HMAC SHA-1
- AES-GCM

2010 -… POST AES-NI / CLMUL
2nd Generation; 3rd Generation Core



If AES-GCM is so good, why everyone is still using SHA-1 HMAC?

- **Inertia:** If is works – don't upgrade it
  - Migration costs and effort
  - Problem is not painful enough / Painful – but to whom?
  - "Legacy": RC4/AES + HMAC-SHA1 is all over the place
- **Ecosystem awareness:** performance benefit & progression - not fully understood
- **Kickoff latency**
  - AES-GCM is a relatively new standard (2008);
    - Part of TLS -- only from TLS 1.2 (which is not proliferated yet)
  - Superior performance: only from 2010 (emergence of AES-NI & PCLMULQDQ)
- **The chicken and the egg problem:**
  - Browsers (client) will not upgrade (TLS1.2) and implement (GCM) before "all" servers support TLS 1.2
  - Servers will not upgrade/implement before "all" browsers have TLS1.2 and offer GCM as an option

In an ideal world: all servers and clients support TLS 1.2, clients offer AES-GCM at handshake
And the ecosystem would see performance gain
But how can we get there?

What needs to happen?

- **Clients** (browsers): add TLS 1.2, as well as GCM support.
  - The client will then offer that as one of their ciphers
- **Server**: support TLS 1.2 and GCM (today ~9% of the servers)
  - Servers with AES-NI/CLMUL would enjoy the faster cipher
- **What happens now?**
  - OpenSSL 1.0.1 already has GCM and TLS 1.2. (and that is slowly deploying)
  - Internet Explorer and MSFT server support TLS 1.2
    - AES-GCM (version 8 on Win 7)
  - Safari (?) (announced TLS 1.2 and AES-GCM)
- **The next big move:** --- NSS to add support
  - (NSS is the stack behind Firefox and Chrome)
  - There is ongoing work there on both GCM and TLS 1.2

Wan-Teh Chang (Google), Bob Relyea (Red Hat), Brian Smith (Mozilla),
Eric Rescorla, Shay Gueron (Intel)

What did we contribute to this?
The new AES-GCM patches (2012)

- Sept./Oct. 2012: We published two patches for two popular open source distributions: OpenSSL and NSS
  - Authors: S. Gueron and V. Krasnov

✓ Inherently side channel protected
  - ✓ “constant time” in the strict definition

✓ Fast on the current x86_64 processors (2^{nd} and 3^{rd} Generation Core)
  - ✓ Fastest we know of

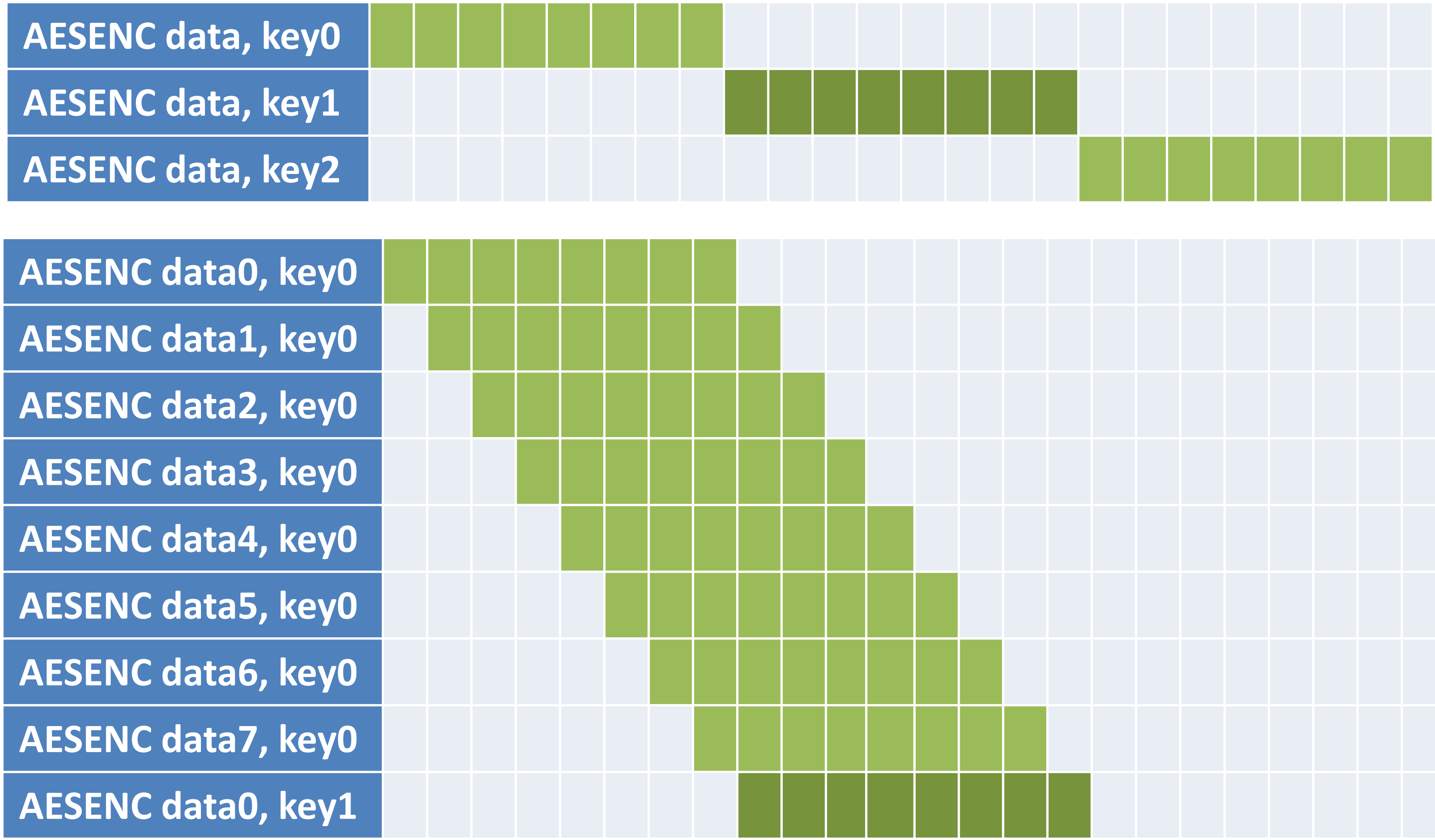
✓ And also ready to boost performance on the coming processors generation (4^{th} Generation Core)

- *Let’s review how this was done*

AES-GCM optimization

1. The encryption
2. The Galois Hash
3. Putting them together

AES-NI: Throughput vs. Latency

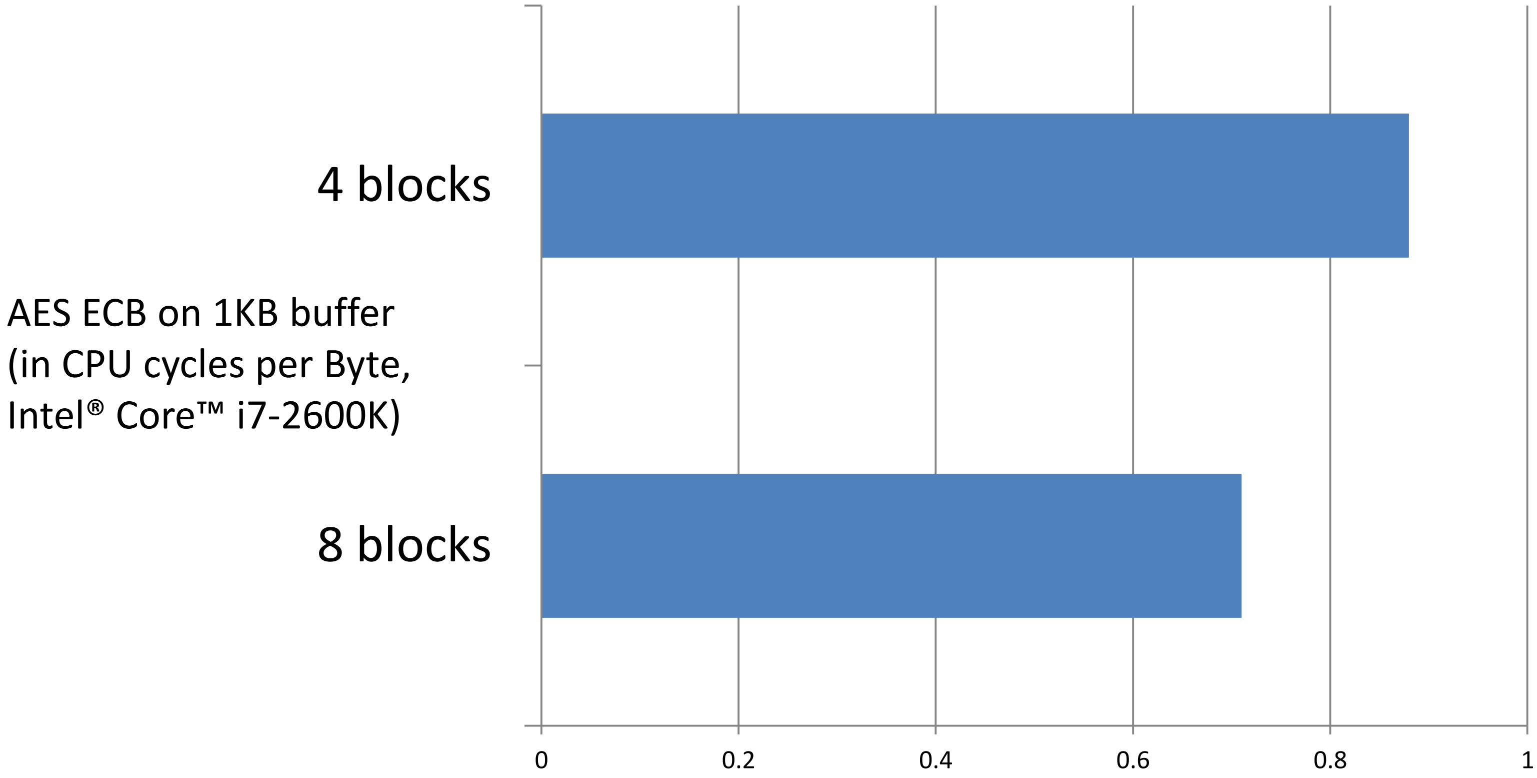


Parallelizable modes (CTR, CBC decryption, XTS) can interleave processing of multiple messages
They become much faster with AES-NI

How much to parallelize?

The effect of the parallelization parameter

Encryption of **8** blocks in parallel vs. encryption of **4** blocks in parallel



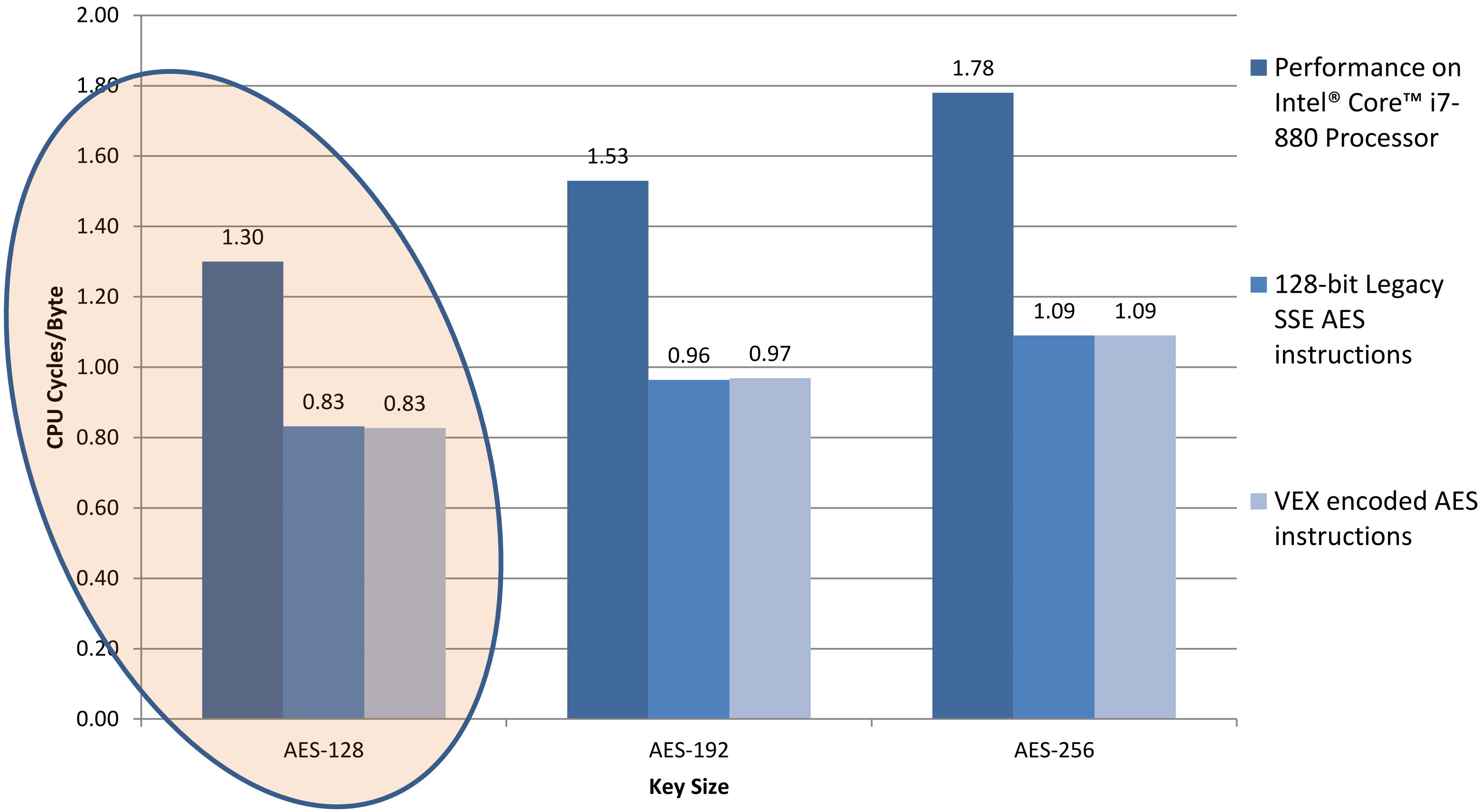
We found the 8 blocks in parallel is a sweat point

AES-CTR performance

Previous Generation Core, Second Generation Core, Thirds Generation Core

Intel® Core™ i7-2600K vs. Intel® Core™ i7-880 Processor

(1KB buffer; performance in CPU cycles per Byte)



128-bit Carry-less Multiplication using PCLMULQDQ

This is fixed

(Gueron Kounavis, 2009) Multiply 128 x 128 → 256 $[A_1 : A_0]\bullet[B_1 : B_0]$

- Schoolbook (4 PCLMULQDQ invocations)

$$A_0\bullet B_0 = [C_1 : C_0], \qquad A_1\bullet B_1 = [D_1 : D_0]$$
$$A_0\bullet B_1 = [E_1 : E_0], \qquad A_1\bullet B_0 = [F_1 : F_0]$$

So this is also fixed

$$[A_1 : A_0]\bullet[B_1 : B_0] = [D_1 : D_0 \oplus E_1 \oplus F_1 : C_1 \oplus E_0 \oplus F_0 : C_0]$$

- Carry-less Karatsuba (3 PCLMULQDQ invocations)

$$A_1\bullet B_1 = [C_1 : C_0], \quad A_0\bullet B_0 = [D_1 : D_0]$$
$$(A_1 \oplus A_0)\bullet(B_1 \oplus B_0) = [E_0 : E_1]$$

$$[A_1 : A_0]\bullet[B_1 : B_0] = [C_1 : C_0 \oplus C_1 \oplus D_1 \oplus E_1 : D_1 \oplus C_0 \oplus D_0 \oplus E_0 : D_0]$$

AES-GCM dirty secrets revealed

A new interpretation to GHASH operations

poly is desrever

This is fixed

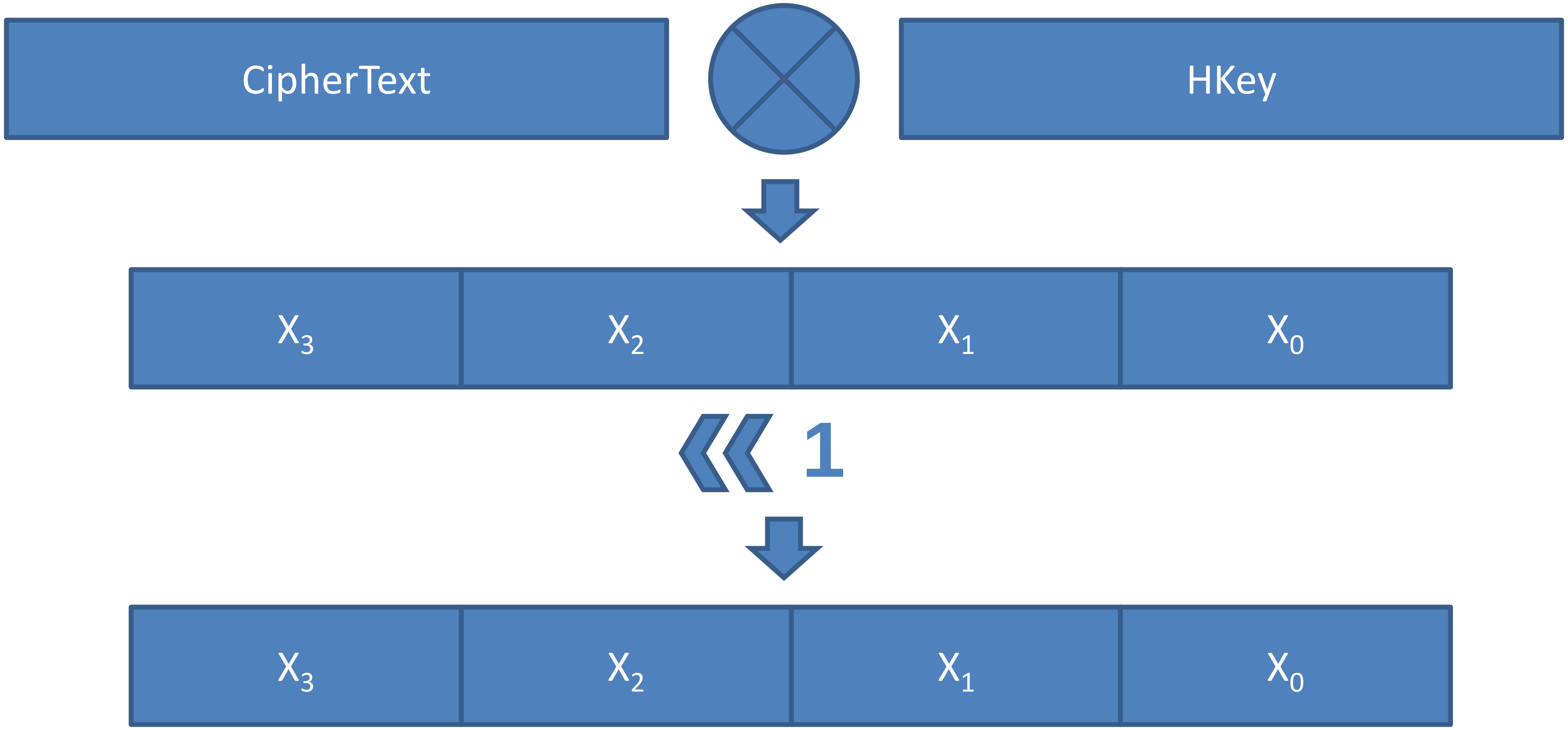
No need to reflect the data

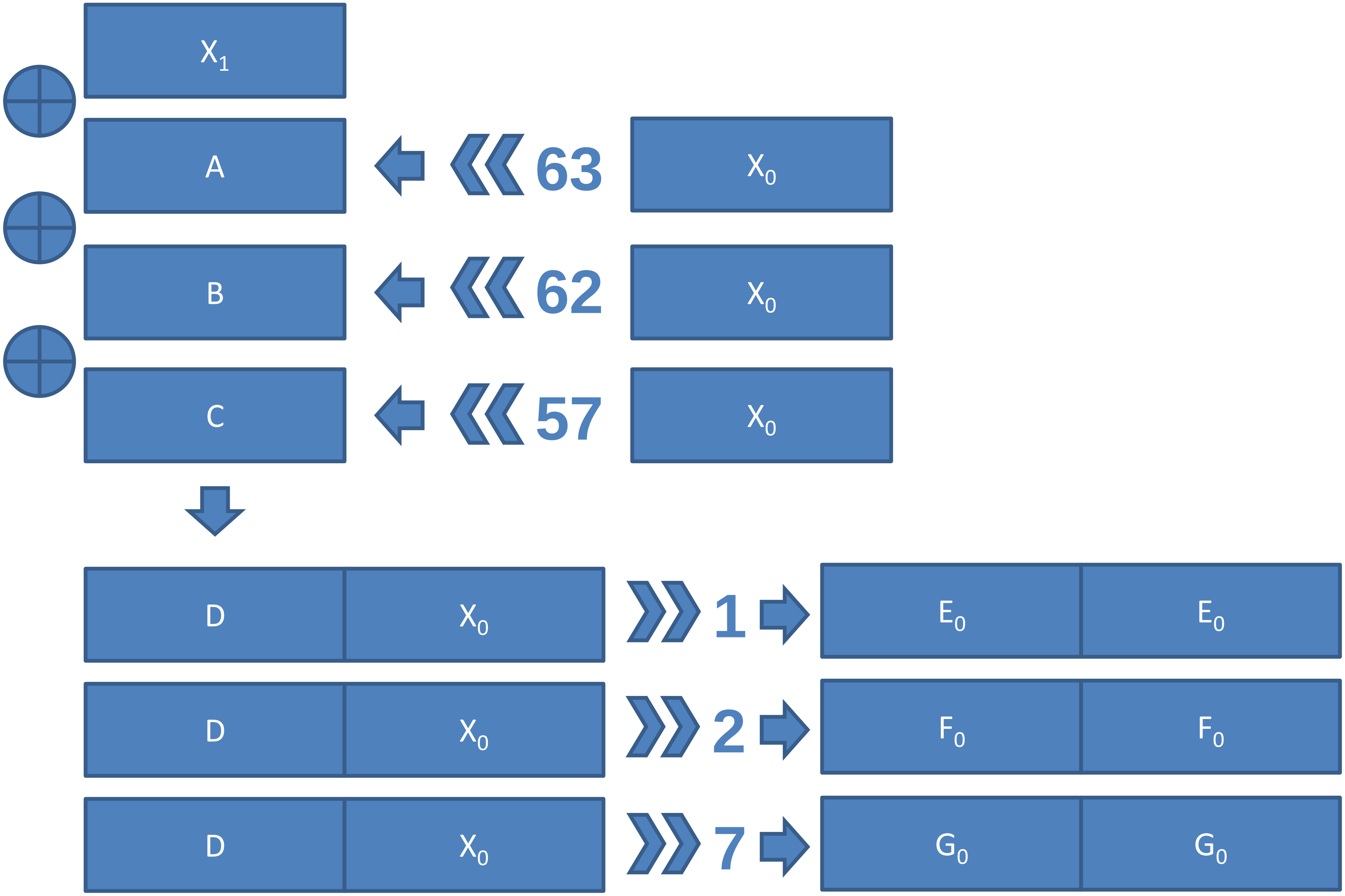
- Not what you expected: GHASH *does not* use $GF(2^{128})$ computations
  - At least not in the usual polynomial representation convention
  - The bits inside the 128-bit operands are reflected
  - Actually - it is an operation on a permutation of the elements of GF(
    - T1 = reflect (A)
    - T2 = reflect (B)
    - $T3 = T1 \times T2$ modulo $x^{128} + x^7 + x^2 + x + 1$ ($GF(2^{128})$ multiplication)
    - Reflect (T3)
- We can prove (a new interpretation) that this operation is:
  - $A \times B \times x^{-127}$ mod $\mathbf{x^{128} + x^{127} + x^{126} + x^{121} + 1}$
  - i.e., a weird Montgomery Multiplication in $GF(2^{128})$ modulo a *reversed* poly
  - Better written as

$$A \times B \times x \times x^{-128} \bmod \mathbf{x^{128} + x^{127} + x^{126} + x^{121} + 1}$$

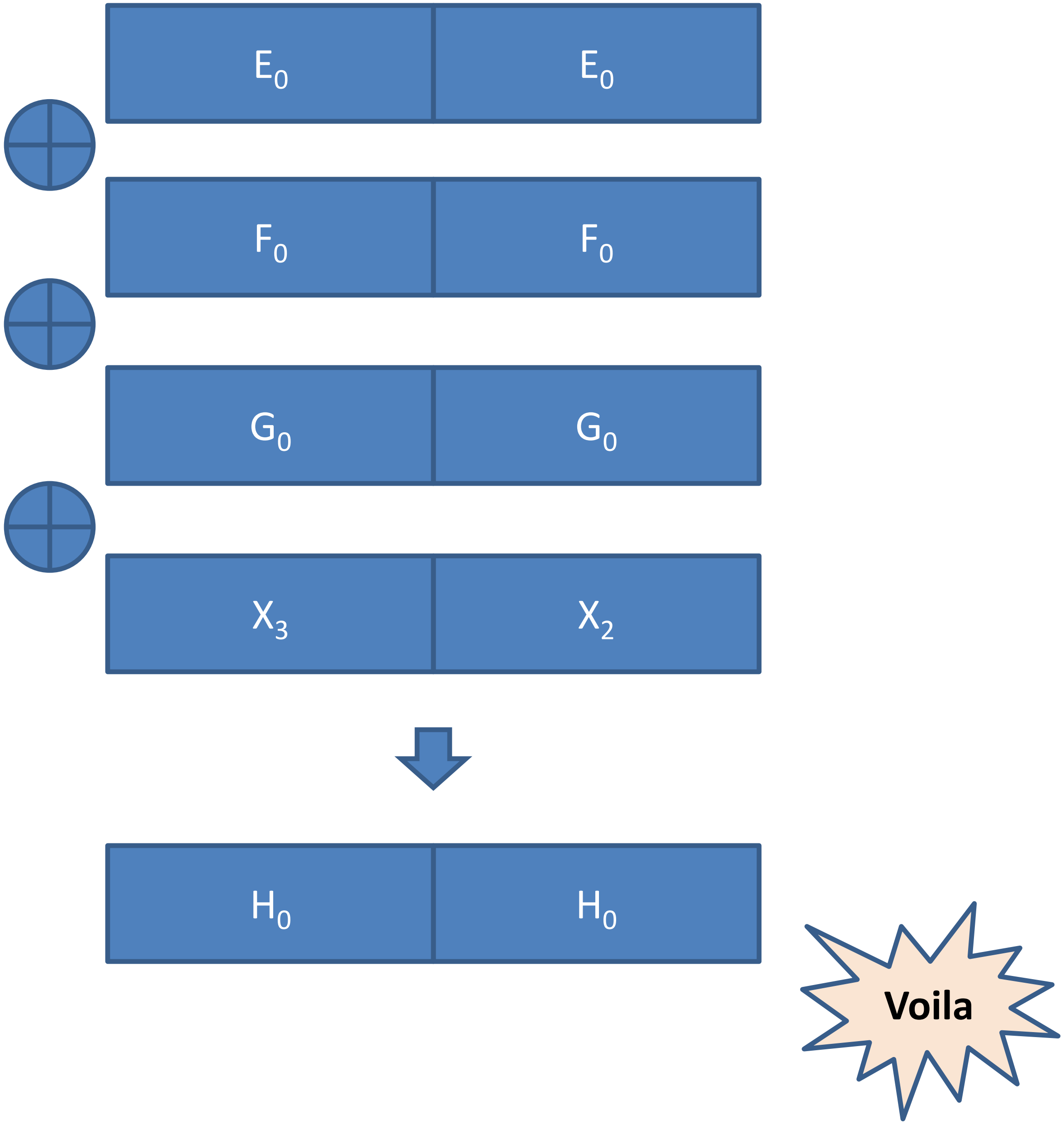
The Shift-XOR reflected reduction

(Gueron Kounavis 2009)





X1
A
B
C
63
62
57
X0
X0
X0
D
X0
D
X0
D
X0
1
2
7
E0
E0
F0
F0
G0
G0


E_0
E_0
F_0
F_0
G_0
G_0
X_3
X_2
H_0
H_0
Voila

Fast reduction modulo $x^{128}+x^{127}+x^{126}+x^{121}+1$

(Gueron 2012)

Algorithm 4: "Montgomery reduction"

Input 256-bit operand $[X_3:X_2:X_1:X_0]$

$[A_1:A_0] = X_0 \bullet 0xc200000000000000$

$[B_1:B_0] = [X_0 \oplus A_1 : X_1 \oplus A_0]$

$[C_1:C_0] = B_0 \bullet 0xc200000000000000$

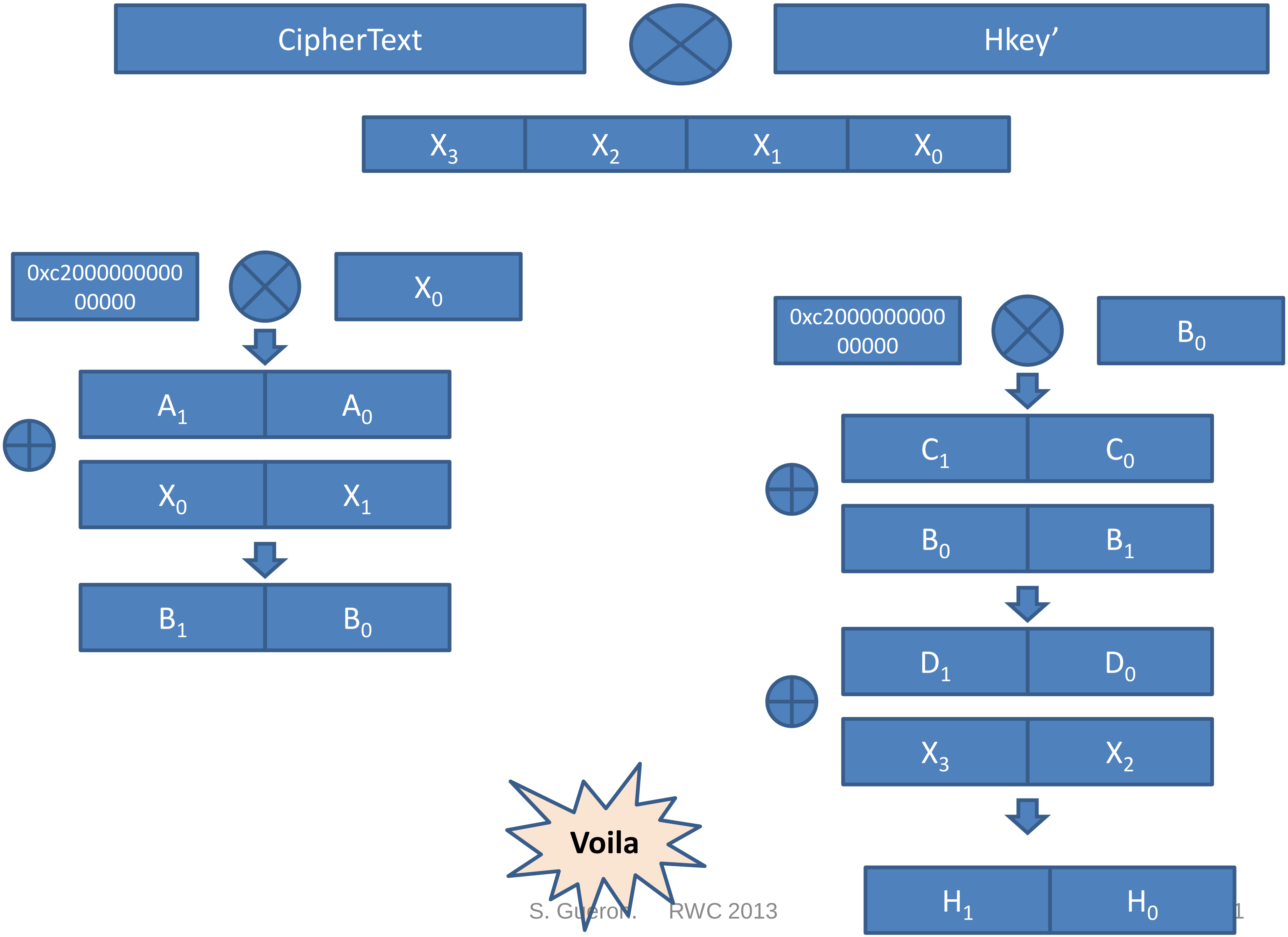
$[D_1:D_0] = [B_0 \oplus C_1 : B_1 \oplus C_0]$

Output: $[D_1 \oplus X_3 : D_0 \oplus X_2]$

The cost:
2 x PCLMULQDQ
3 x shift/xor
Ideal with fast PCLMULQDQ

```
; Input is in T1:T7
vmovdqa    T3, [W]
vpclmulqdq T2, T3, T7, 0x01
vpshufd    T4, T7, 78
vpxor      T4, T4, T2
vpclmulqdq T2, T3, T4, 0x01
vpshufd    T4, T4, 78
vpxor      T4, T4, T2
vpxor      T1, T4   ; result in T1
```

The optimized reflected reduction



Aggregated Reduction

The Ghash operation is:

$$\mathbf{MM\ (CT_1\ ,\ Hx^{"m"}) + MM\ (CT_2\ ,\ Hx^{"m-1"}) + \ldots + MM\ (CT_m\ ,\ Hx)}$$

$$\mathbf{mod\ x^{128} + x^{127} + x^{126} + x^{121} + 1}$$

- In a Horner form (facilitating iterative computation)
  - Y_i = MM [(X_i + Y_{i-1}), Hx] ...everything mod Q= $x^{128} + x^{127} + x^{126} + x^{121} + 1$
- 4-way expanded Horner form (aggregate results & defer the reduction step)
  - Yi = MM [(X_i + Y_{i-1}) , Hx] = MM [(X_i , Hx)] + MM [(Y_{i-1} , Hx)]

    = MM [(X_i , Hx)] + MM [(X_{i-1}+ Y_{i-2}) , Hx"2"] =

    = MM [(X_i , H)] + MM [(X_{i-1} , Hx"2")] + MM [(X_{i-2}+Y_{i-3}) , Hx"3"]

    = **MM [(X_i , Hx)] + MM [(X_{i-1} , Hx"2")] + MM [(X_{i-2} , Hx"3")] + MM [(X_{i-3}+Y_{i-4}) , Hx"4"]**
  - Can be expanded further
  - The gain: reduction deferred to once per "N" blocks
  - Overhead: pre-calculate the powers of H (amortized for reasonably long buffer)

Interleaving CTR and GHASH

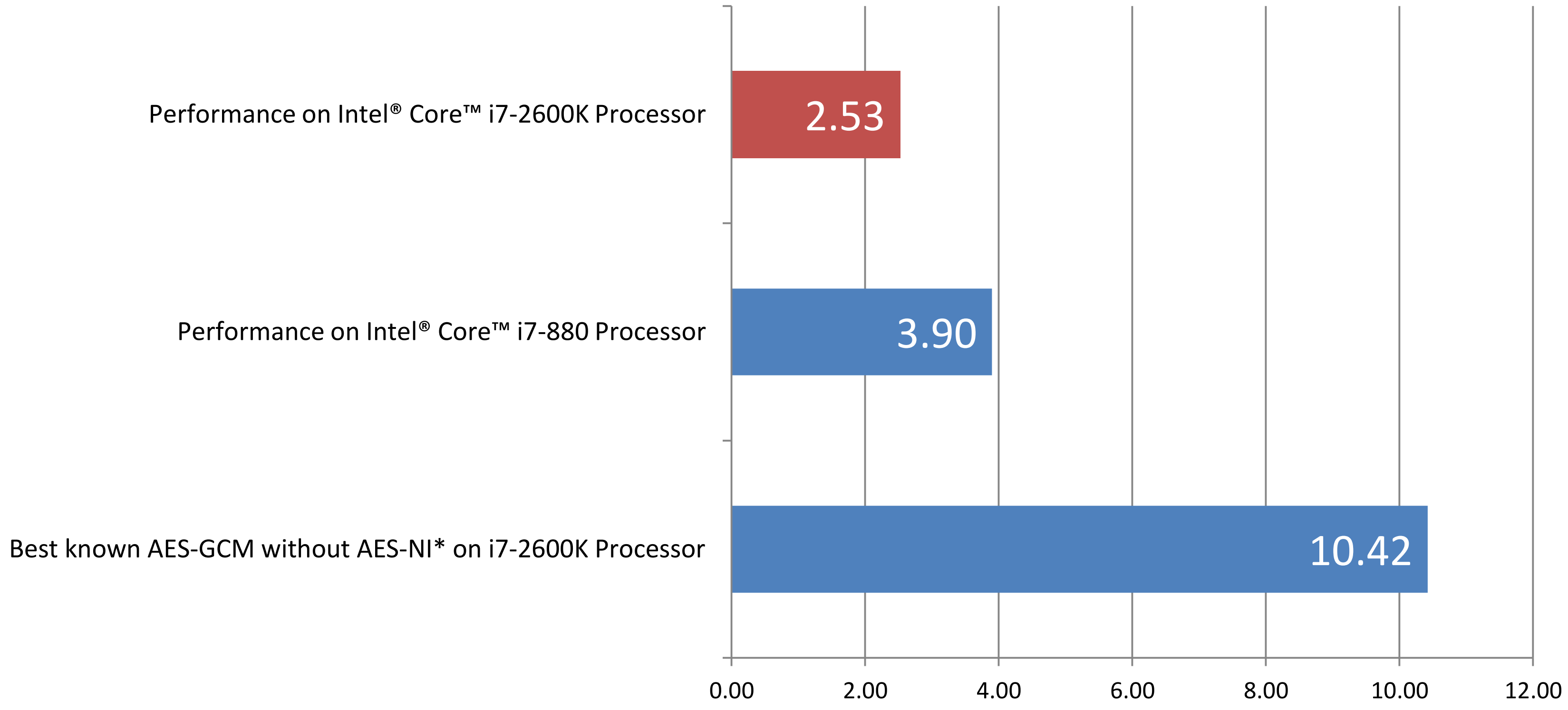
- There are two approaches to GCM
  - Use dedicated AES-CTR function for the encryption and another GHASH function to generate the MAC
  - Gain additional performance by interleaving the calculation of CTR and GHASH in a single function
- The first approach can only achieve the performance of “CTR+GHASH”
- The second approach achieves a better performance
  - Filling the execution pipe more efficiently.

The new AES-GCM patches (2012)
putting it (and more…) all together

- Sept./Oct. 2012: We published two patches for two popular open source distributions: OpenSSL and NSS
  - NSS patch to be committed into version 3.14.2
- Both patches share similar code and use :
  - Carry-less Karatsuba multiplication
  - Reduce using “Montgomery”
  - Encrypt 8 counter blocks
  - Deferred reduction (using 8 block aggregation)
  - Fixed elements outside the brackets
  - Interleave CTR and GHASH
- Inherently side channel protected
  - “constant time” in the strict definition
- Fast on current processors (2nd and 3rd Generation Core)
- And also ready to boost on the coming processors (4th Generation Core)

Results

The performance of AES-128 GCM Encryption on 4KB buffer in CPU cycles per Byte, Intel® Core™ i7-2600K vs. Intel® Core™ i7-880 Processor, Lower is better



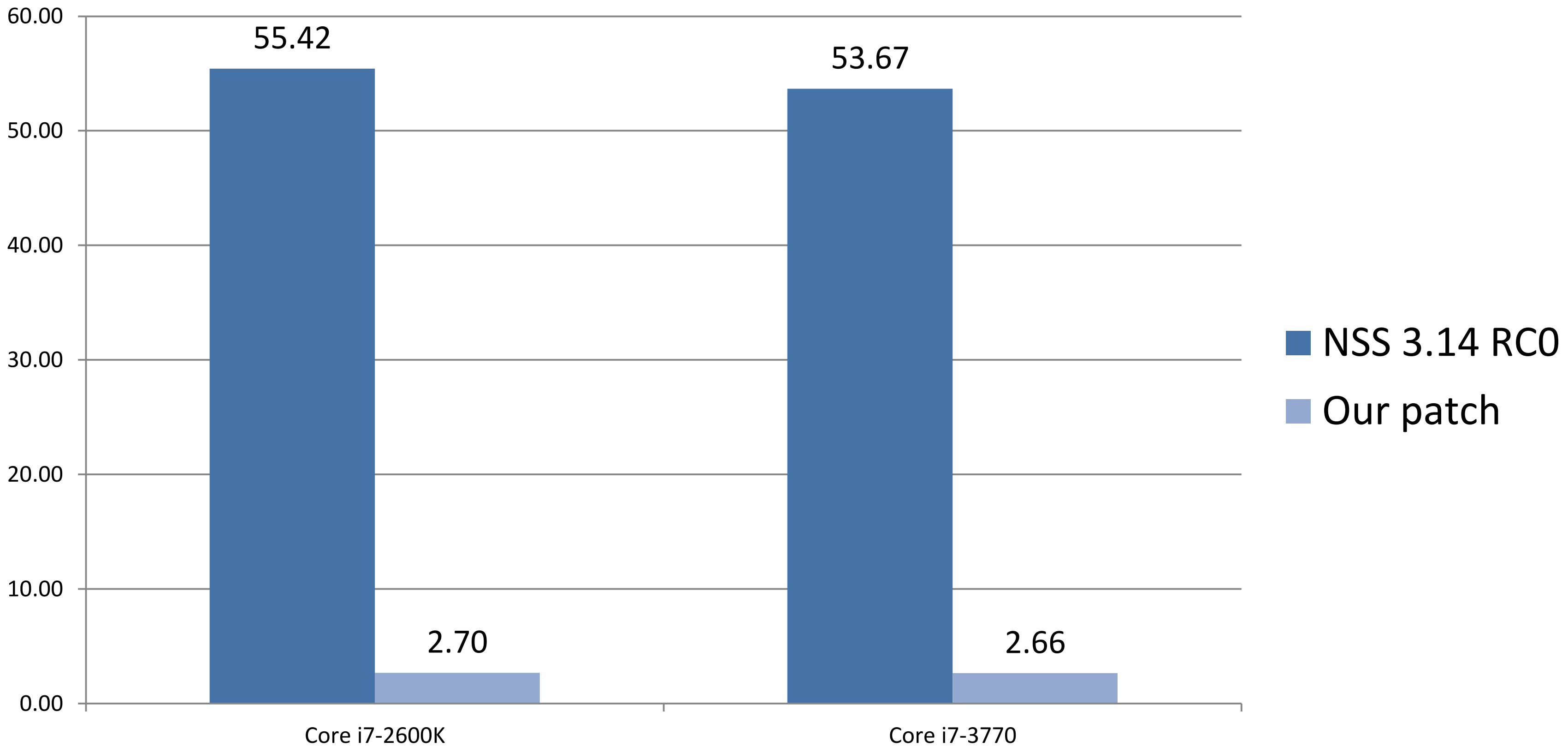
* E. Käsper, P. Schwabe, Faster and Timing-Attack Resistant AES-GCM, http://homes.esat.kuleuven.be/~ekasper/papers/fast_aes_slides.pdf

Some breakdown

- AES-GCM:
  - 4KB message: 2.53 C/B
  - 16KB message: 2.47 C/B
- Breakdown
  - CTR performance for 16KB: 0.79 C/B
  - The cost of the GHASH is ~1.68 C/B
    - ~68% of the computations
  - The performance of standalone GHASH is 1.75 C/B
    - The delta is the gain from interleaving GHASH with CTR.
- Notes: the MAC computations are still significant
  - Limited by the current performance of PCLMULQDQ
  - Ultimate goal: achieve AES-GCM at the performance of CTR+ ε

The NSS patch (2012)

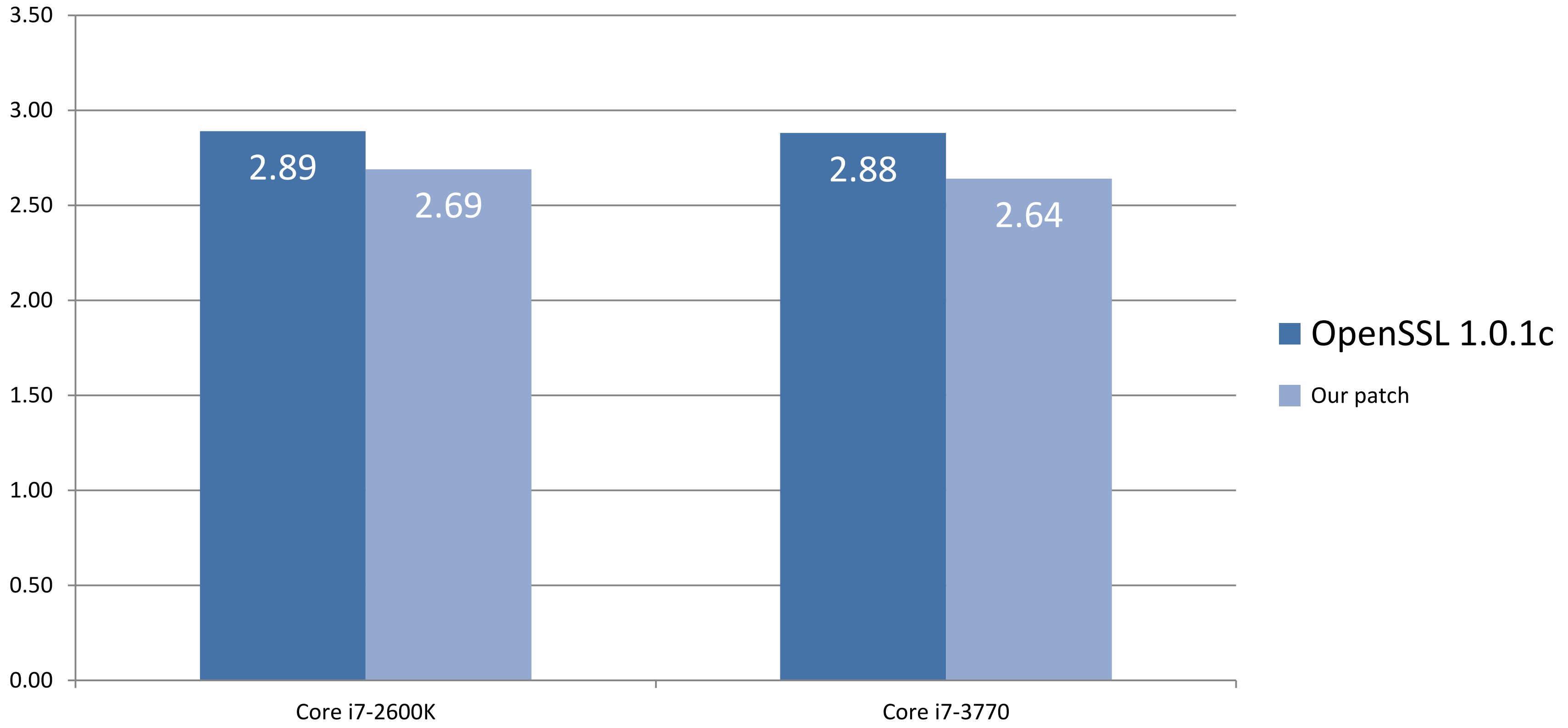
The performance of NSS AES GCM Encryption on 8KB buffer in CPU cycles per Byte, Intel® Core™ i7-2600K and Intel® Core™ i7-3770 Processors, Lower is better



Ready to boost performance on the coming processors generation (4th Generation Core)

The OpenSSL patch (2012)

The performance of OpenSSL AES GCM Encryption on 8KB buffer in CPU cycles per Byte, Intel® Core™ i7-2600K and Intel® Core™ i7-3770 Processors, Lower is better

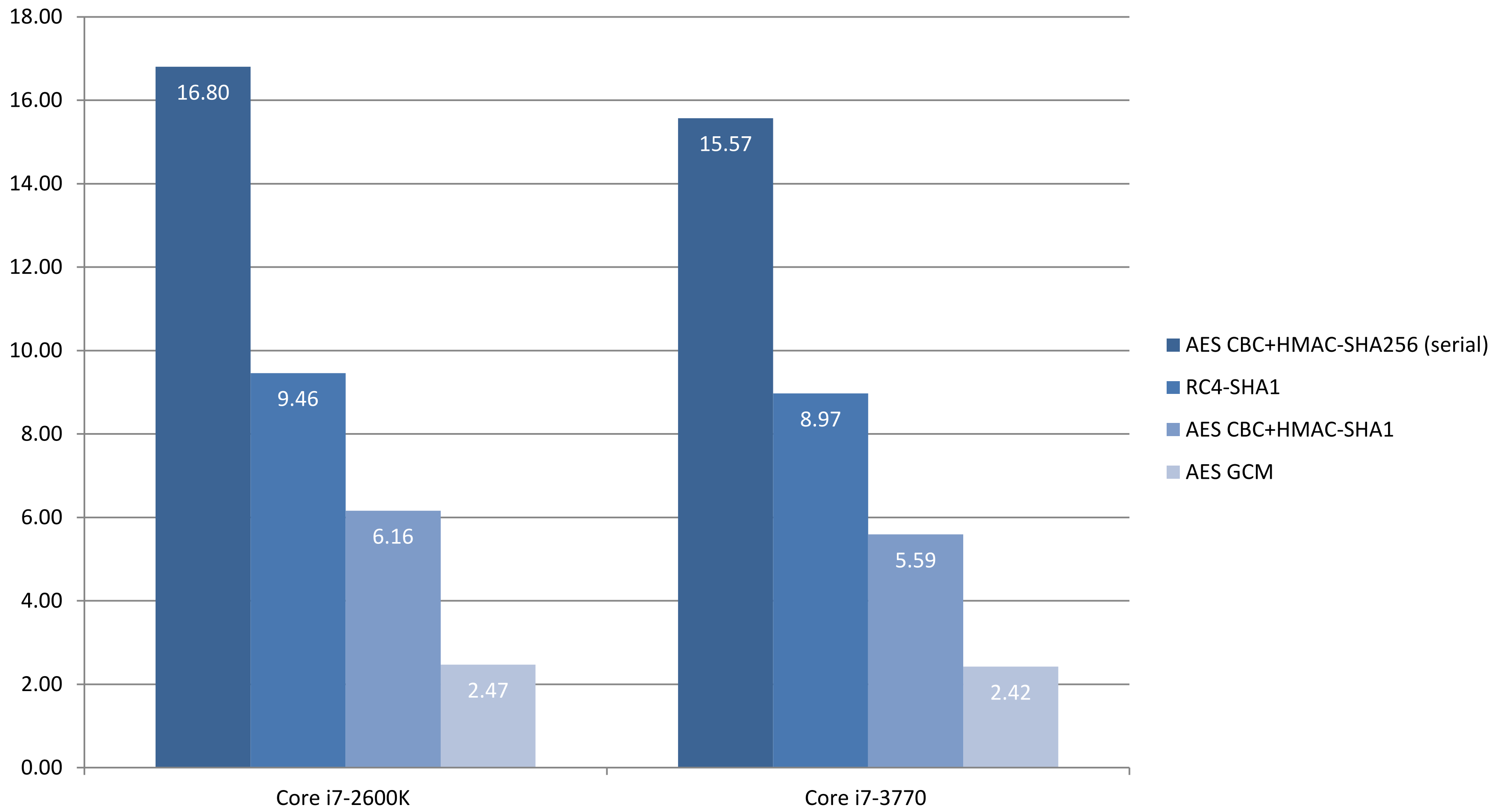


Ready to boost performance on the coming processors generation (4th Generation Core)

What does it give?

AES-GCM vs. other (NIST standard) Authenticated Encryption

The performance of NSS AES GCM Encryption on 32KB buffer in CPU cycles per Byte, Intel® Core™ i7-2600K and Intel® Core™ i7-3770 Processors, Lower is better



Summary

- AES-GCM is the best performing Authenticated Encryption combination among the NIST standard options (esp. compared to using HMAC SHA-1)
  - SE on x86-64
  - + Performance keeps improving across CPU generations
  - Just wait for the coming “4th Generation Core” (2013)
- We try to actively help the eco-system move to the more efficient AE
- With some luck, we might see significant deployment already in 2013
  - Optimized algorithms & implementations released as patches for Open Source
  - Thanks to Google/Mozilla/RedHat colleagues
    - Review and commit to NSS; add TLS1.2; enable Firefox / Chrome support
- The ultimate goal: achieve AES-GCM at the performance of CTR+ ε
- All the codes and papers are publicly available (see reference)

References

References

AES-GCM (The algorithms and methods that underlie the AES-GCM patches codes are detailed in references [1-4])

1. S. Gueron, Michael E. Kounavis: Intel® Carry-Less Multiplication Instruction and its Usage for Computing the GCM Mode (Rev. 2.01) http://software.intel.com/sites/default/files/article/165685/clmul-wp-rev-2.01-2012-09-21.pdf
2. S. Gueron, M. E. Kounavis: Efficient Implementation of the Galois Counter Mode Using a Carry-less Multiplier and a Fast Reduction Algorithm. Information Processing Letters 110: 549û553 (2010).
3. S. Gueron: AES Performance on the 2nd Generation Intel Core Processor Family (to be posted) (2012).
4. S. Gueron: Fast GHASH computations for speeding up AES-GCM (to be published soon) (2012).

AES-NI

5. S. Gueron. Intel Advanced Encryption Standard (AES) Instructions Set, Rev 3.01. Intel Software Network. http://software.intel.com/sites/default/files/article/165683/aes-wp-2012-09-22-v01.pdf
6. S. Gueron. Intel's New AES Instructions for Enhanced Performance and Security. Fast Software Encryption, 16th International Workshop (FSE 2009), Lecture Notes in Computer Science: 5665, p. 51-66 (2009).

OpenSSL patch:

- S. Gueron, V. Krasnov, "[PATCH] Efficient implementation of AES-GCM, using Intel's AES-NI, PCLMULQDQ instruction, and the Advanced Vector Extension (AVX). http://rt.openssl.org/Ticket/Display.html?id=2900&user=guest&pass=guest (2012)

NSS patch:

- S. Gueron, V. Krasnov, "Efficient AES-GCM implementation that uses Intel's AES and PCLMULQDQ instructions (AES-NI), and the Advanced Vector Extension (AVX) architecture. For the NSS library", Attachment 673021 Details for Bug 373108, [PATCH] https://bugzilla.mozilla.org/show_bug.cgi?id=805604#c0 (2012)