Third NIST Workshop on Block Cipher Modes of Operation 2023

October 3-4, 2023

Constructions based on the AES round and polynomial multiplication that are efficient on modern processor architectures

Shay Gueron

University of Haifa, Israel and **Meta, USA**

shay.gueron@gmail.com

Agenda

- AES-NI + PCLMULQDQ + GF-NI
  - Current and future performance
  - The most important lesson: latency versus throughput
  - Ingredients in multiple usages
  - Can make BBB constructions affordable
  - Lots of innovation to happen
- Call for NIST actions

The AES-NI: AES New Instructions (now already old)

AES New Instructions (AES-NI)

- Introduced into the x86-64 Instructions Set in 2009
- Westmere (2009), Sandy bridge (2010)…

Ubiquitous practically on all 64-bit architectures

- Fully pipelined (throughput 1)
- Decreasing latency across generations
- 8 → 7 → 4 → 3 cycles

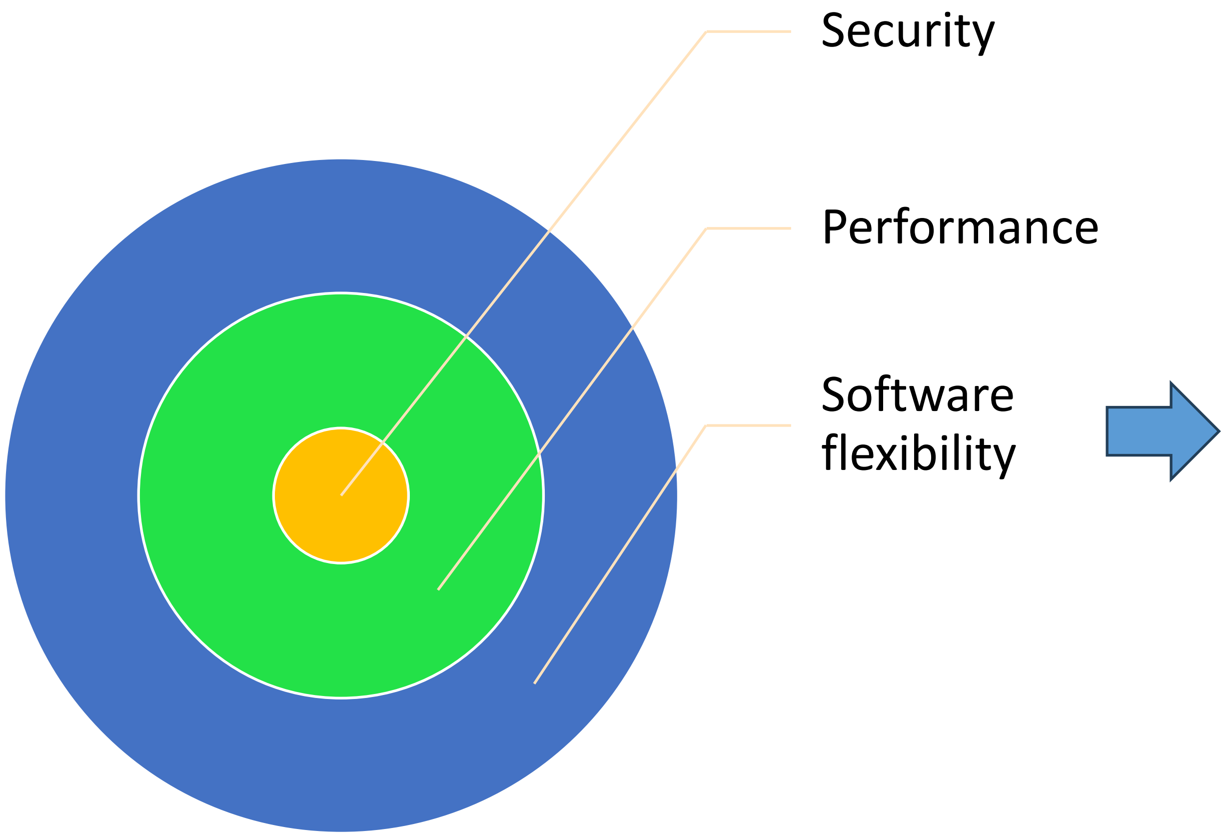
Recently added

- Vectorized form
- 4 x throughput compared to a single AES-NI unit

Recently added

- GF-NI
- GF(2^8) support
- GF(2^8) operations: multiply, inverse affine, affine-inverse

The AES-NI: AES New Instructions: planned goals



- AES encryption and decryption
- All key lengths (128/192/256)
- Key expansion on-the-fly and offline
- All modes of operation

- Unplanned byproduct
  - Extract AES transformation individually

- PCLMULQDQ: 63 degree polynomial multiplication
  - Speed up AES-GCM

- But actually, as it turns out, there is much more

AES Encryption flow

Altogether 40/48/56 steps

- Tmp = AddRoundKey (Data, Round_Key_Encrypt [0])
- For round = 1-9 (or 1-11) (or 1-13)
- Tmp = ShiftRows (Tmp)
- Tmp = SubBytes (Tmp)
- Tmp = MixColumns (Tmp)
- Tmp = AddRoundKey (Tmp, Round_Key_Encrypt [round])
- end loop
- Tmp = ShiftRows (Tmp)
- Tmp = SubBytes (Tmp)
- Tmp = AddRoundKey (Tmp, Round_Key_Encrypt [10 or 12 or 14])
- Result = Tmp

AESENC (S, RK)

AESENCLAST (S, RK)

AES New Instructions (AES-NI)

AES encryption

- AESENC
- AESENCLAST

AES decryption

- AESDEC
- AESEDCLAST

Key expansion

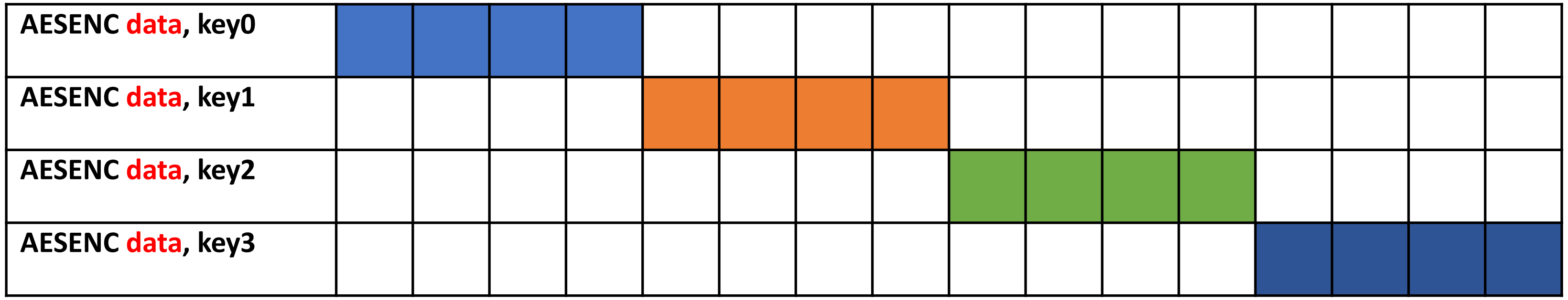
- AESKEYGENASSIT X
- AESIMC

Polynomials in $Z_2[x]$

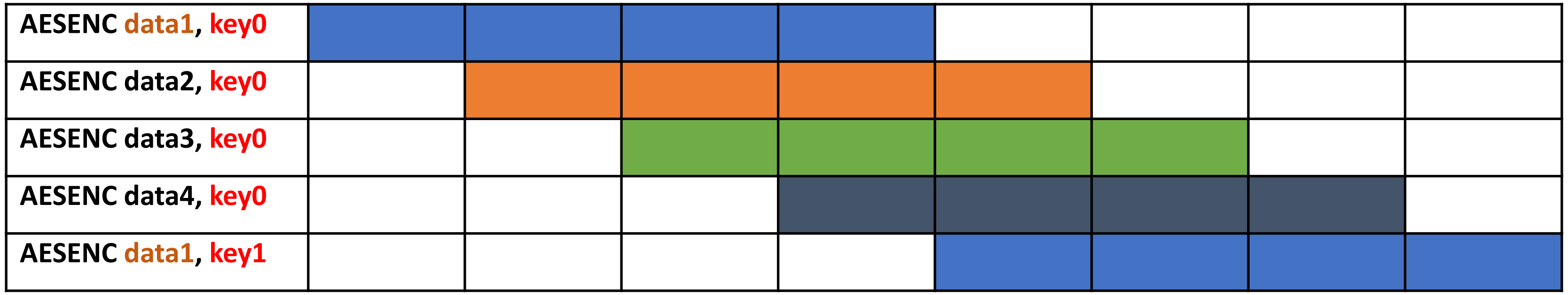
- PCLMULQDQ

AES-NI: throughput vs. latency (for 4 cycles latency)

A most important observation

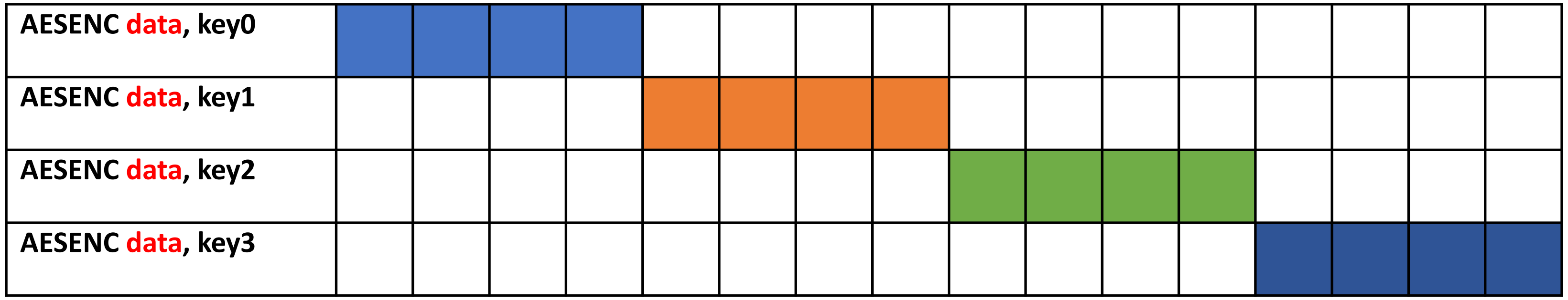


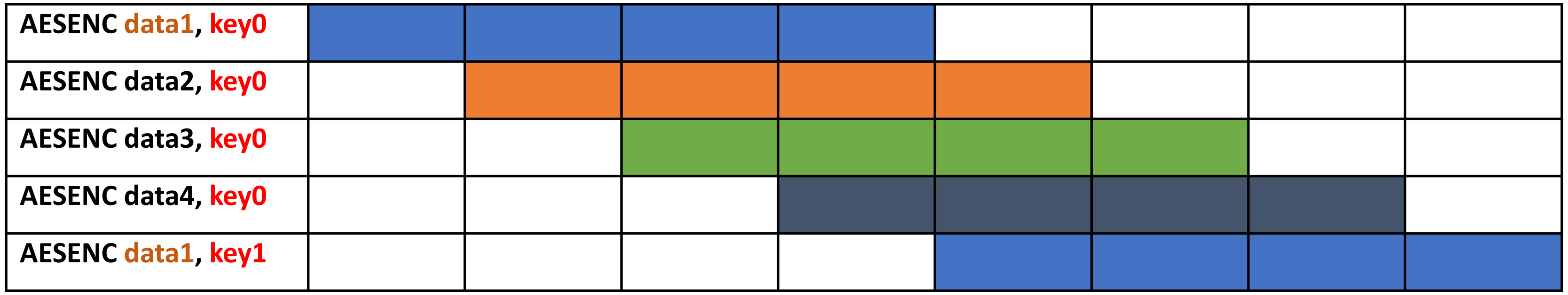
Serial performance:
Latency: 1+10*L
Per 16B block
L = 8 → 5.06 C/B
L = 4 → 2.56 C/B



Gueron (2010)

AES-NI: throughput vs. latency (for 4 cycles latency)





Pipelined Performance:
L = 4
B/2 + 10*B+L for 16B

Asymptotically: 1 fully pipeline AES unit can support throughput of 10/16 = 0.625 C/B

≈0.625 C/B is an achievable performance target

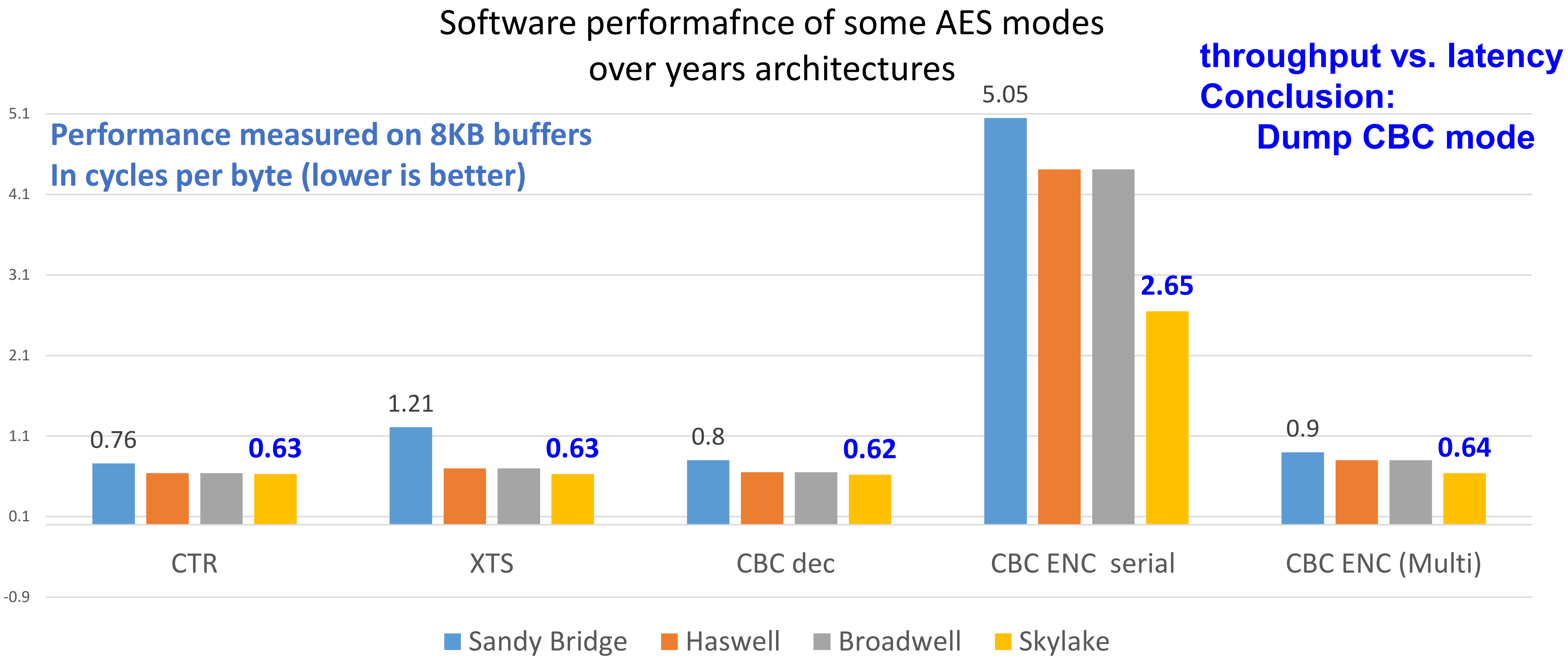
AES encryption flow using AES-NI

```
__m128i AES128_encrypt (const __m128i EKS [11], const __m128i* pt) {
__m128i ct;
          ct = _mm_xor_si128(*pt, EKS[0]);
          ct = _mm_aesenc_si128(ct, EKS[1]);
          ct = _mm_aesenc_si128(ct, EKS[2]);
          ct = _mm_aesenc_si128(ct, EKS[3]);
          ct = _mm_aesenc_si128(ct, EKS[4]);
          ct = _mm_aesenc_si128(ct, EKS[5]);
          ct = _mm_aesenc_si128(ct, EKS[6]);
          ct = _mm_aesenc_si128(ct, EKS[7]);
          ct = _mm_aesenc_si128(ct, EKS[8]);
          ct = _mm_aesenc_si128(ct, EKS[9]);
          ct = _mm_aesenclast_si128(ct, EKS[10]);
     return ct;}
```

AES encryption flow using AES-NI – 4-way pipelining

```
tmp1 = _mm_xor_si128(tmp1,((__m128i*)key)[0]);
tmp2 = _mm_xor_si128(tmp2,((__m128i*)key)[0]);
tmp3 = _mm_xor_si128(tmp3,((__m128i*)key)[0]);
tmp4 = _mm_xor_si128(tmp4,((__m128i*)key)[0]);
    for(j=1; j <10; j++) {
        tmp1 = _mm_aaesenc_si128 (tmp1,((__m128i*)key)[j]);
        tmp2 = _mm_aesenc_si128 (tmp2,((__m128i*)key)[j]);
        tmp3 = _mm_aesenc_si128 (tmp3,((__m128i*)key)[j]);
        tmp4 = _mm_aesenc_si128 (tmp4,((__m128i*)key)[j]);
        };
    tmp1 = _mm_aesenclast_si128 (tmp1,((__m128i*)key)[10]);
    tmp2 = _mm_aesenclast_si128 (tmp2,((__m128i*)key)[10]);
    tmp3 = _mm_aesenclast_si128 (tmp3,((__m128i*)key)[10]);
    tmp4 = _mm_aesenclast_si128 (tmp4,((__m128i*)key)[10]);
```

Software performance of some AES modes (over the years)



AES-GCM optimizations (1)

Combining PCLMULQDQ and AES-NI

- **PCLMULQDQ** **64 x 64 → 128 (carry-less)**
  - Using it for AES-GCM (GHASH) -- $GF(2^{128})[x]$ 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)
- **GHASH does not operate on $GF(2^{128})$ computations "as expected"**
  - Bits inside the bytes are reversed
  - Description in AES-GCM NIST spec SP800-38D is through a bit-level algorithm
  - Current "**modulo** $x^{128} + x^7 + x^2 + x + 1$" description obscures the math and hides the optimizations
  - Equivalent (mathematical) formulation of the field operation is:
    - $A \times B \times x^{-127}$ mod $x^{128} + x^{127}+x^{126}+x^{121} + 1$
    - Better written as $A \times (B \times x) \times x^{-128}$ mod $x^{128} + x^{127}+x^{126}+x^{121} + 1$

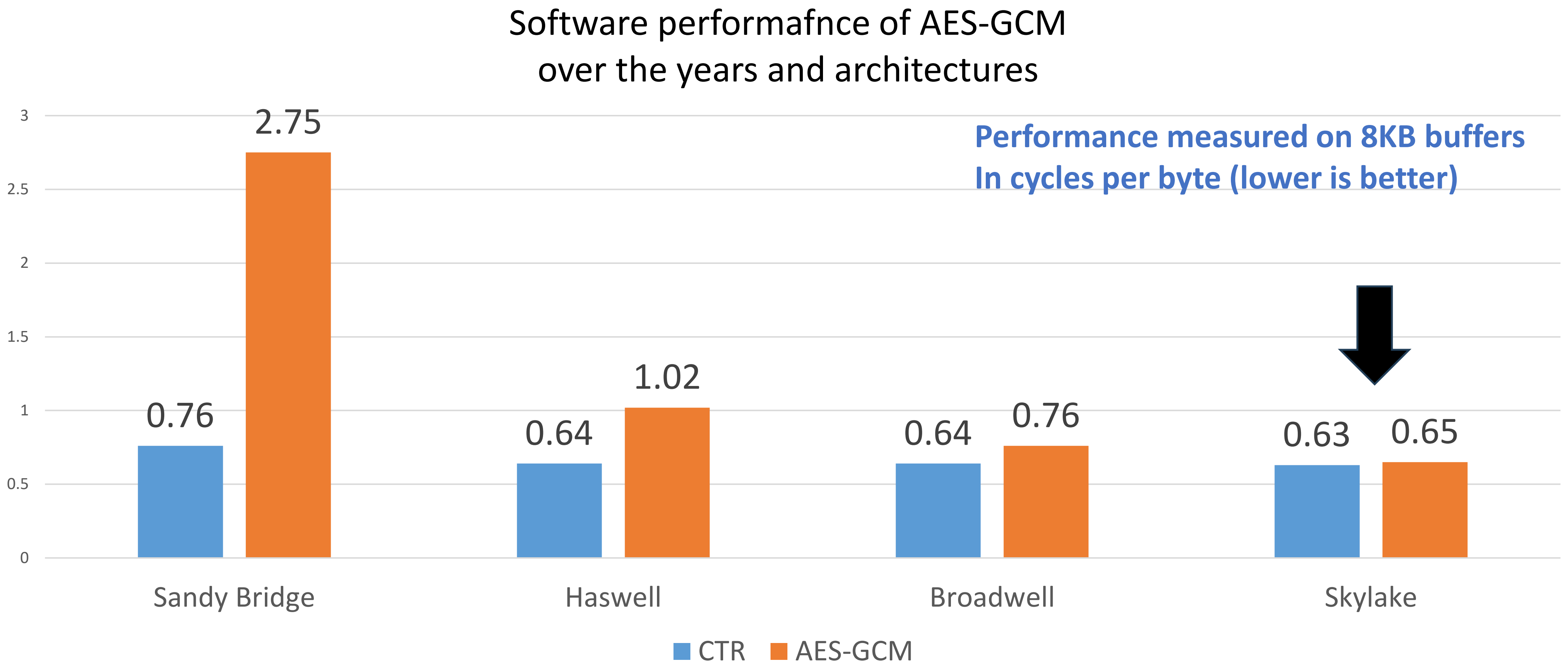
Gueron (2013, 2023)
Gueron, Kounavis (2008, 2010)

AES-GCM optimizations (2)

Combining PCLMULQDQ and AES-NI

- **Improve PCLMULQDQ**: latency and throughput
- **Aggregated reduction** instead of a Horner form (iterative computation)
  - Defer reduction to once every “N” blocks
- **Interleaving** AES-NI optimized CTR and GHASH
  - Better pipelining ➔ better performance
- **Microarchitecture**: AESENC and PCLMULQDQ on separate ports

Software performance of AES-GCM (over the years)



Isolating the AES transformations

Isolating ShiftRows

```
PSHUFB xmm0, 0x0b06010c07020d08030e09040f0a0500
```

Isolating InvShiftRows

```
PSHUFB xmm0, 0x0306090c0f0205080
```

Isolating MixColumns

```
AESDECLAST xmm0, 0x0000000000000000
AESENC xmm0, 0x00000000000000000000
```

Isolating InvMixColumns

```
AESENCLAST xmm0, 0x000000
AESDEC xmm0, 0x000
```

Isolating SubBytes

```
PSHUFB xmm0, 0x0306090c0f0205080b0e0
AESENCLAST xmm0, 0x0000000000000000
```

Isolating InvSubBytes

```
PSHUFB xmm0, 0x0b06010c07020d08030e0
AESDECLAST xmm0, 0x0000000000000000
```

AESDECLAST xmm0, 0
Tmp:= Inverse Shift Rows (State);
Tmp:= Inverse Sub Bytes (Tmp);
xmm0:= Tmp xor 0 = xmm0
AESENC xmm0, 0
Round Key:= 0
Tmp:= Shift Rows (Tmp);
Tmp:= Sub Bytes (Tmp);
Tmp:= Mix Columns (Tmp);

PSHUFB xmm0 xmm0,
0x0306090c0f0205080b0e0104070a0d00

AESENClast xmm0, 0
Round Key:= 0
Tmp:= Shift Rows (Tmp);
Tmp:= Sub Bytes (Tmp);
xmm0:= Tmp xor 0

Garbled circuits (Multiparty computations)

- Compare:

  Gueron, Lindell, Nof, Pinkas (2018)

  - $AES_{K_1}\left(AES_{K_2}(K_3)\right)$

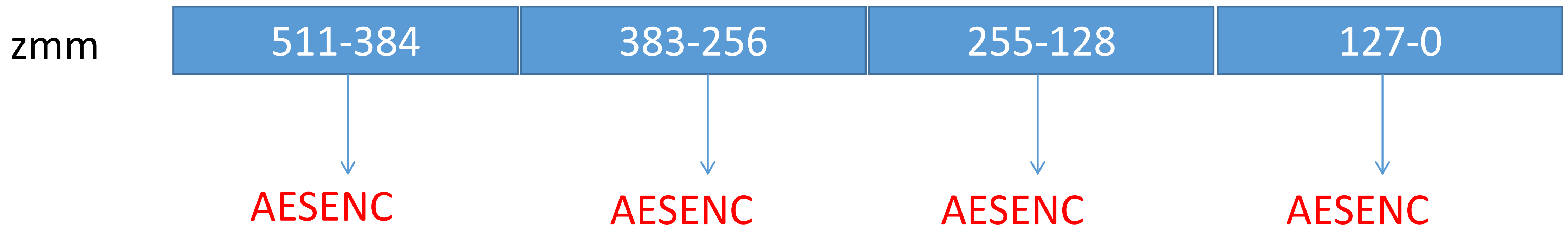
  to

  - $AES_{K_1}(g) \oplus AES_{K_2}(g) \oplus K_3$
  - The former cannot be pipelined, whereas the latter can
- Number of operations:
  - AND gate garbling: 4 keys, 8 encryptions
  - AND/XOR gate evaluation: 2 keys, 2 encryptions
  - XOR gate garbling: 4 keys, 4 encryptions
- Pipeline multi-key-scheduling with one encryption
  - Replace AESKEYGENASSIST with the AESENCLAST + PSHUFB (shuffle)

Vector AES and Vector PCLMULQDQ
(AVX2) AVX512 architecture

AESENC, AESENCLAST, AESDEC, AESDECLAST, PCLMULQDQ

operating on 4 independent SIMD elements (blocks of 128 bits)

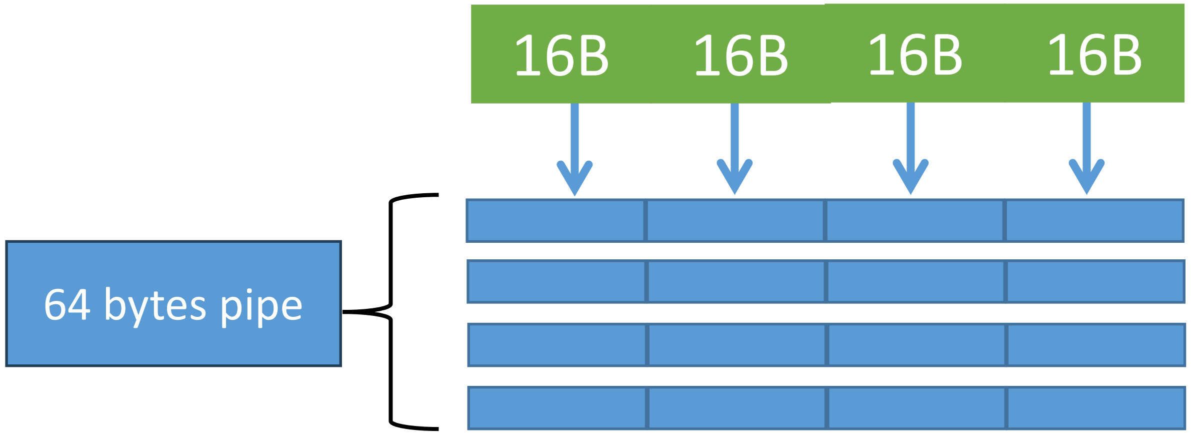


Quadruple the throughput of the AVX version

Drucker, Gueron, Krasnov (2018a, 2018b, 2019)

Vectorization of the instructions

- Vectorized (4x) AES; Vectorized (4x) PCLMULQDQ
  - Vectorized (32x) GF instructions ("GF-NI")
- **Filling the pipeline is crucial** for enjoying the throughput benefits
  - Needs a sufficient number of streams or sufficient number of independent sub-buffers



- AVX512 AND AVX2

- Get ready for AES-GCM at ~ 0.17 C/B
- AES-XTS at ~ 0.165 C/B
- AES-GCM-SIV enc at ~ 0.24 C/B
- AES-GCM-SIV dec at ~0.17 C/B

Rijndael-256 (256-bit block size) – do them with AES-NI

- 256-bit block; 256-bit key; 14 rounds

Round = 2 × blends + 2 × shuffles + 2 × AESENCs

```
vpblendvb %xmm2, %xmm1, %xmm5, %xmm3
vpblendvb %xmm1, %xmm2, %xmm5, %xmm4
pshufb    %xmm8, %xmm3
pshufb    %xmm8, %xmm4
aesenc    %xmm6, %xmm3
aesenc    %xmm7, %xmm4
```

- Theoretical: 1.75 C/B (accounting for only AESENCs; but shuffles/blends conflict on ports)
- Achieved for CTR mode: 2.23 C/B
- **Using Vector-AES-NI 256-bit block cipher for 0.87 C/B**

Gueron (201)
Drucker, Gueron (2022)

International-NI

- Different ciphers use different representations of $GF(2^8)$ for Sbox
  - AES Sbox uses: $x^8 + x^4 + x^3 + x + 1$
  - ZUC-256 Sbox uses: $x^8 + x^7 + x^3 + x^1 + 1$
  - SM4 uses: $x^8 + x^7 + x^6 + x^5 + x^4 + x^2 + 1$
  - Camellia use: $x^8 + x^6 + x^5 + x^3 + 1$
- All representations of $GF(2^8)$ are isomorphic
  - F_X is another representation of $GF(2^8)$ then:
    - there is (fixed) matrix 8× 8 K and isomorphism

$$\phi : \mathbb{F}_{\mathrm{X}} \longrightarrow \mathbb{F}_{\mathrm{AES}}$$
$$x \longmapsto Kx$$

- Matrix · vector flow:

```
xmm1 = _mm_srli_epi64(in, 4);
xmm1 = _mm_and_si128(xmm1, and_mask);
xmm2 = _mm_and_si128(in, and_mask);
xmm1 = _mm_shuffle_epi8(mask2, xmm1);
xmm2 = _mm_shuffle_epi8(mask1, xmm2);
xmm1 = _mm_xor_si128(xmm1, xmm2);
```

- A composition of affine transformations is an affine transformation

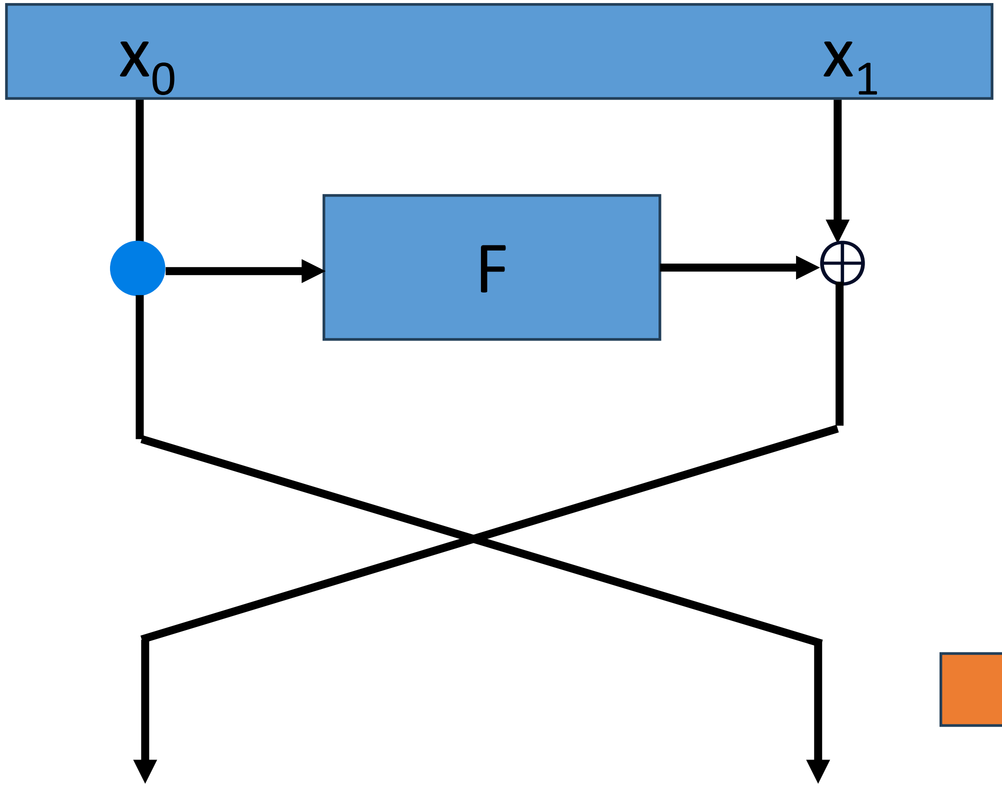
GF-NI: $A \cdot x + b$; $A \cdot x^{-1} + b$, $u \times v$ in GF (2^8)

- GF-NI: $A \cdot x + b$; $A \cdot x^{-1} + b$, $u \times v$ in GF (2^8) / $x^8 + x^4 + x^3 + x + 1$
- GF-NI: a cryptographer's paradise

Hardware ties to a specific polynomial?

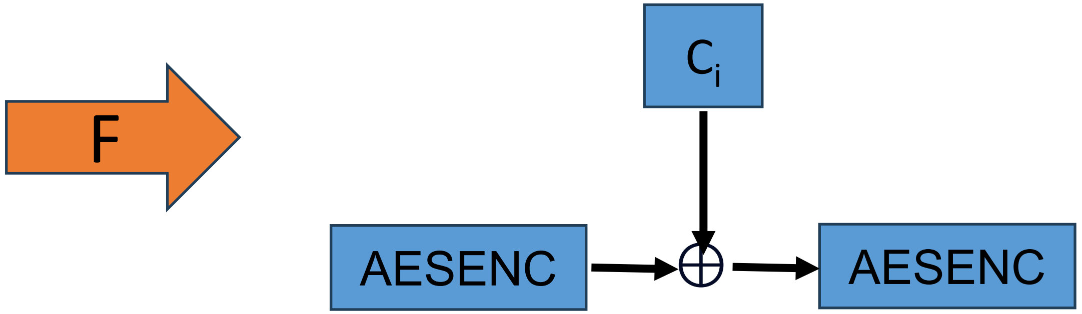
- Affine transformation $A \cdot x + b$
  - Any field representation to any othe representations – and back
  - **But also:** any bit-permutation inside (64-) byte
    - Bit reverse, nibble swap,
  - Parity, partial parity
  - Select any number of bits
  - and more…
- GF-NI: $A \cdot x^{-1} + b$,
  - Make your own (GF(2^8)-based) Sboxs , together with AFFINE
- GF-NI: $u \times v$
  - Error correction codes

Simpira permutation (b=2; 256 bits)



- (key-less) Cryptographic permutation
- **15** Feistel rounds
  - **2** serialized-AESENC per round
  - Total: **30**
- Latency (on 4 cycles): 120 cycles / 32 bytes
- Throughput: (4 pipe): 30/32 = **0.94** C/B
- **With vectorization**: **0.24** C/B

R(y, z, i) = AESENC (AESENC (y, C(i)), z)



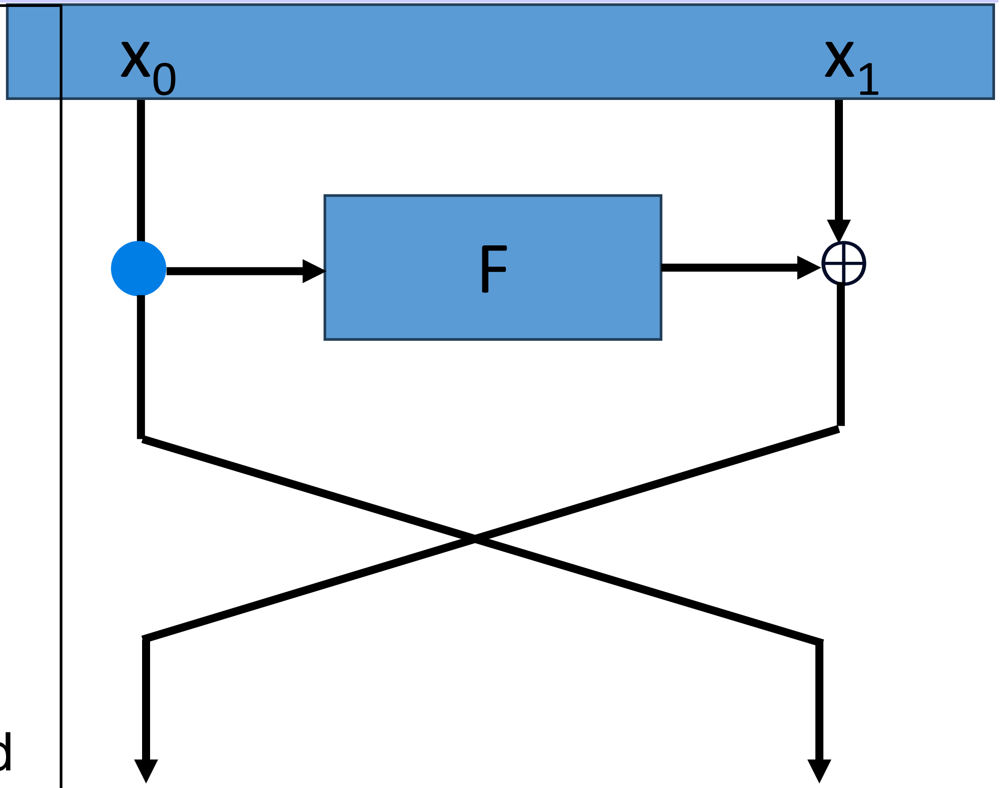
C_i = (0x00^i^0x02, 0x10^i^0x02,0x20^i^0x02, 0x30^i^0x02)

Gueron, Mouha (2016, 2017)

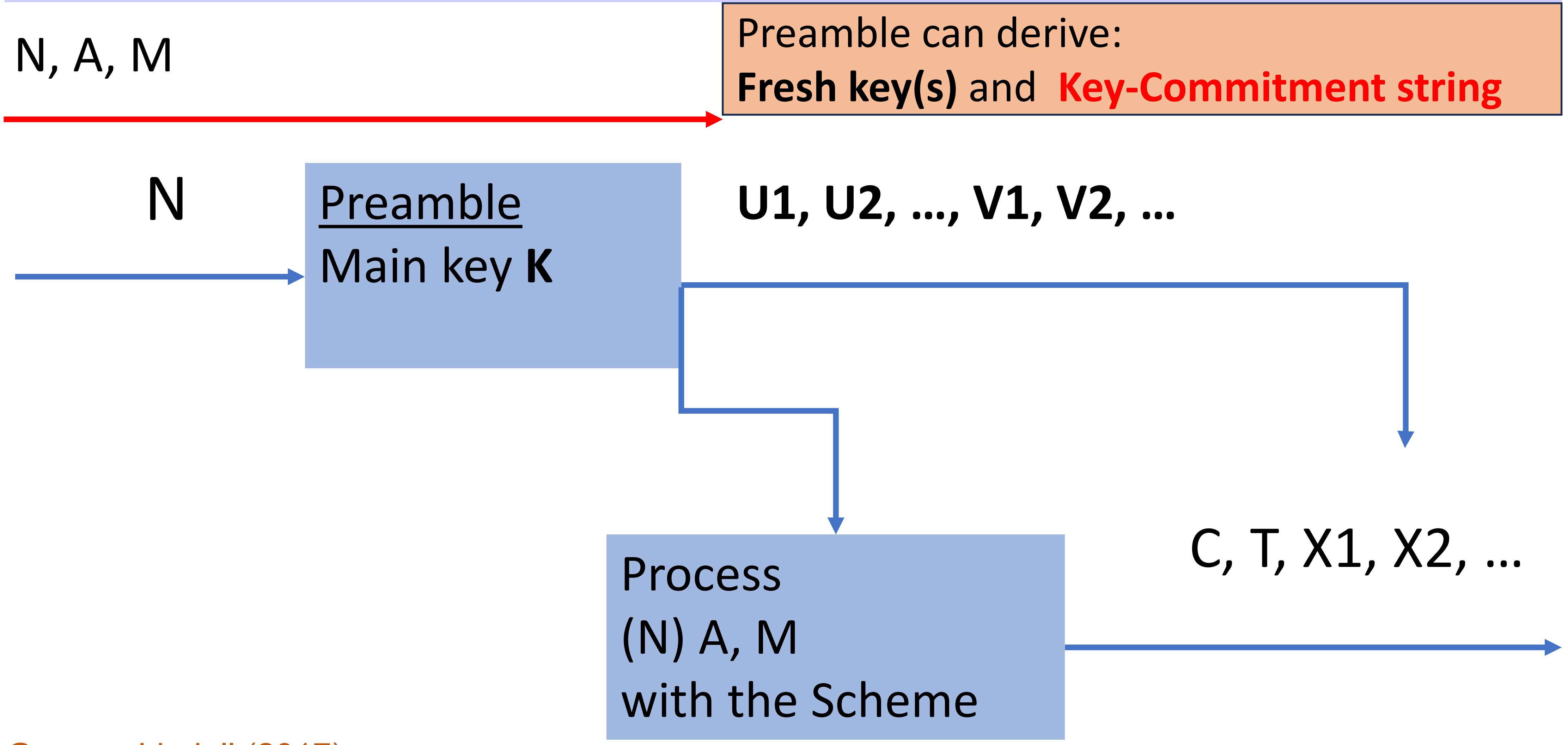
Simpira permutation (b=2; 256 bits)

- A cryptographic permutation: building block for
  - Wide-block encryption
  - Hashing
  - Short-input hashing
  - Authenticated encryption
- Even-Mansour construction
  - Simpira $(X \oplus K) \oplus K$
  - **Efficient*** 256-bit block cipher with 128-bit multi-key security.
  - Security beyond 2^{64} data blocks
  - Block cipher with no key schedule overhead
- Tweakable EM: Simpira $(P \oplus K \cdot T) \oplus K \cdot T$

* At < 1 C/B (< 0.25 C/B with vectorization)



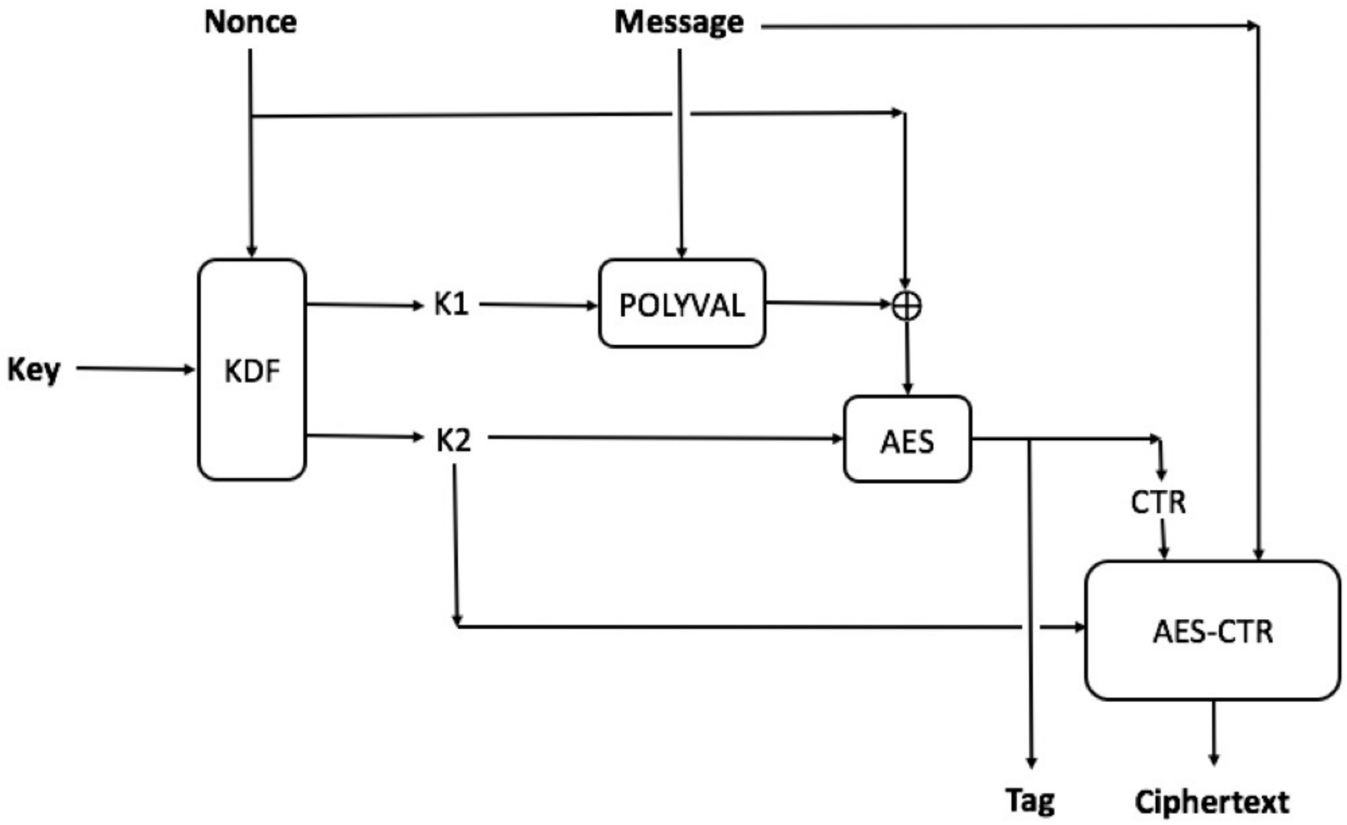
AEAD with a preamble



Gueron, Lindell (2017)

AES-GCM-SIV (a preamble example)

- Nonce misuse resistant AEAD
- Good security bounds
- Efficient (< 1 C/B)
  - ~4x with vectorization
- New (improved) universal hash
- Extended key lifetime
  - $2^{64} \times 2^{13}$ blocks with adv. 2^{-32}



Gueron, Lindell (2017)
Gueron, Langley, Lindell (2019) RFC8542

Derive key AES-GCM - extend key lifetime to “forever”
avoid problem with 96-bit random nonces

- **Derive-Key-AES-GCM** (K, N, A, M)
  - K’ = Derive (K, N) |N| = 126
  - (C, Tag) = AES-GCM (K’, 0^{96}, A, M)
- Derive (K, N): a permutation-based PRF
- Extended key lifetime
- Improved security bounds

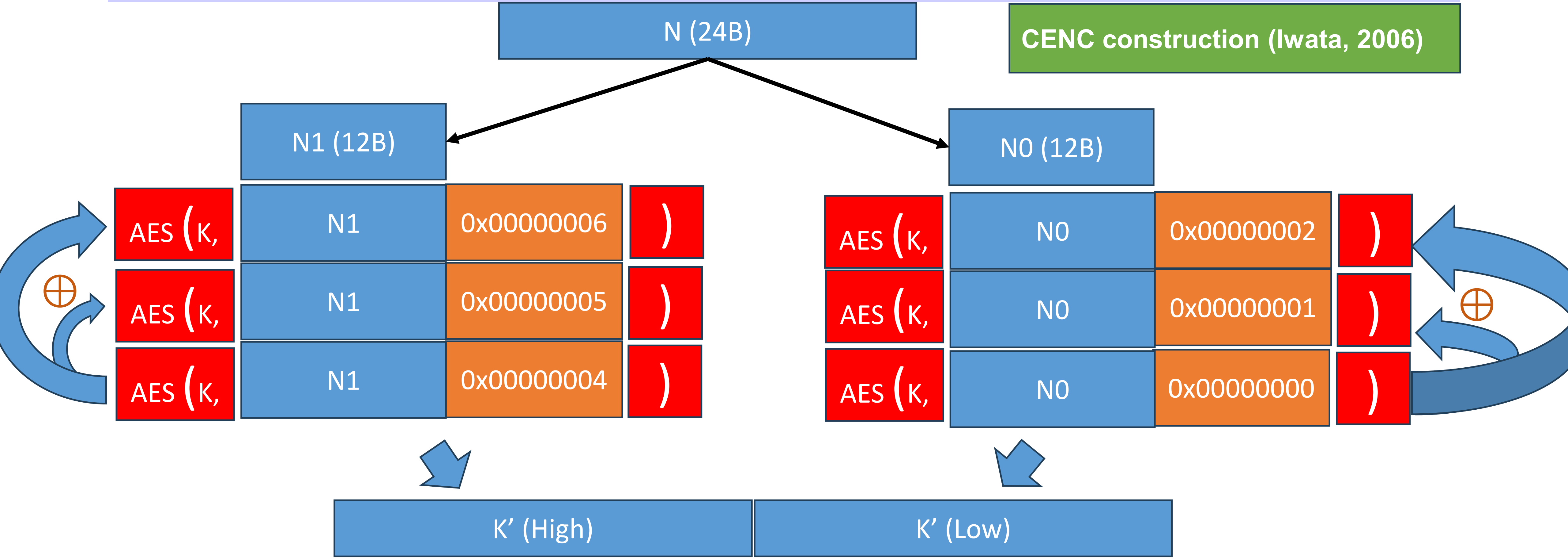
Derive:
- An efficient (hopefully) BBB PRF construction
- Used for producing a relatively small number of pseudorandom blocks.
- Minimalism → use AES as a building block.

192-bit random nonce do not collide
256-bit keys
Do not collide…

- **Double-Nonce-Derive-Key-AES-GCM** (K, N, A, M)
  - K’ = Derive (K, N) |N| = 192
  - (C, Tag) = AES256-GCM (K’, 0^{96}, A, M)
- Derive (K, N): a permutation-based PRF
- Extended key lifetime & better security bounds; 2^{64} blocks

Gueron, Lindell (2017)
Gueron (2022)

Double-Nonce-Derive: (K, N) → K'
from only permutations



Double-Nonce-Derive-Key-AES-GCM is on path to soon be used at Meta for production systems

Innovative-creative use of AES-NI / PCLMULQDQ

- Samples from many designs that use the AES round as a building block
  - Reduced-round (4 and 6 rounds) AES as a component in AEZ [HKR15] and LmD [BDMN16]
  - Apparent impact on the CAESAR competition: almost all the authenticated cipher winning proposals (e.g., AEGIS, OCB [KR21], Deoxys-II) pipeline AES or AES elements
  - Haraka [KLMR16]: AESENC-inspired designs for short-input keyed hash
- PCLMULQDQ
  - CRCs and fast error detection
  - High degree polynomial mult / inverse (some post-quantum KEM proposals e.g. BIKE)
  - POLYVAL: a universal family of hash functions
    - Faster than GHASH on Little-Endian architectures
- GF-NI
  - A cryptographer's paradise

Summary and call for action

- AES-NI + PCLMULQDQ + GF-NI
  - Current and future performance
  - Ingredients in multiple usages
  - Can make BBB constructions affordable (XOR two permutations?)
  - (Cook your own dinner from these ingredients)
- Call for NIST actions
  1. Change GHASH formulation in AES-GCM NIST spec SP800-38D to
     1. $A \times (B \times x) \times x^{-128} \bmod x^{128} + x^{127} + x^{126} + x^{121} + 1$
  2. Standardize AES-GCM-SIV
  3. Standardize “preamble” as an acceptable extension-mode for standard modes
     1. Double-Nonce-Derive-Key-AES-GCM (K, N, A, M)
  4. Standardize 256-bit block ciphers and/or a cryptographic permutation
     1. Rijndael 256
     2. Simpira (b=2) + Even-Mansour + tweak for a tweakable (authenticated) cipher

Thank you

References (1)

- [BBGR09] Benadjila R., Billet O., Gueron S., Robshaw M.J.B. (2009) The Intel AES Instructions Set and the SHA-3 Candidates. In: *Matsui M. (eds) Advances in Cryptology – ASIACRYPT 2009. ASIACRYPT 2009*. Lecture Notes in Computer Science, vol 5912. Springer, Berlin, Heidelberg.
- [BOS11] Bos J.W., Özen O., Stam M. (2011) Efficient Hashing Using the AES Instruction Set. In: *Preneel B., Takagi T. (eds) Cryptographic Hardware and Embedded Systems – CHES 2011. CHES 2011. Lecture Notes in Computer Science*, vol 6917. Springer, Berlin, Heidelberg.
- [BDMN16] Bossuet, L., Datta, N., Mancillas-López, C., Nandi, M. (2016, November). ELmD: A Pipelineable Authenticated Encryption and Its Hardware Implementation. In *IEEE Transactions on Computers*, vol. 65, no. 11, pp. 3318-3331.
- [DGK18a] Drucker, N., Gueron, S., & Krasnov, V. (2018, June). Fast multiplication of binary polynomials with the forthcoming vectorized VPCLMULQDQ instruction. In *2018 IEEE 25th Symposium on Computer Arithmetic* (ARITH) (pp. 115-119). IEEE.
- [DGK18b] Drucker, N., Gueron, S., & Krasnov, V. (2018, June). The comeback of Reed Solomon codes. In *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)* (pp. 125-129). IEEE.
- [DGK19] Drucker, N., Gueron, S., & Krasnov, V. (2019). Making AES great again: the forthcoming vectorized AES instruction. In *16th International Conference on Information Technology-New Generations (ITNG 2019)* (pp. 37-41). Springer, Cham.
- [DG22] Drucker, N., & Gueron, S. (2022). Software Optimization of Rijndael for Modern x86-64 Platforms. In *ITNG 2022 19th International Conference on Information Technology-New Generations* (pp. 147-153). Springer, Cham.
- [Gue09] Gueron, S. (2009, February). Intel's new AES instructions for enhanced performance and security. In *International Workshop on Fast Software Encryption* (pp. 51-66). Springer, Berlin, Heidelberg.

References (2)

- [Gue10] Gueron, S. (2010). Intel advanced encryption standard (AES) instructions set. *Intel White Paper, Rev*, *3*, 1-94 https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf
- [Gue13] Gueron, S. (January, 2016). AES-GCM for Efficient Authenticated Encryption – Ending the Reign of HMAC-SHA-1?. *In Workshop on Real-World Cryptography (Real World Crypto)* https://crypto.stanford.edu/RealWorldCrypto/ (2013).
- [Gue22] Gueron, S. (2022). Counter Mode for Long Messages and a Long Nonce. *In: Dolev, S., Katz, J., Meisels, A. (eds) Cyber Security, Cryptology, and Machine Learning. CSCML 2022.* Lecture Notes in Computer Science, vol 13301. Springer, Cham.
- [Gue23] Gueron, S. (2023). A New Interpretation for the GHASH Authenticator of AES-GCM. *In: Cyber Security, Cryptology, and Machine Learning.* CSCML 2023, 424–438. Lecture Notes in Computer Science, vol 13914. Springer, Cham.
- [GK08a] Gueron, S., & Kounavis, M. (2008). Carry-less multiplication and its usage for computing the GCM mode. *White Paper, Intel Corporation*. https://www.intel.ph/content/dam/www/public/us/en/documents/white-papers/carry-less-multiplication-instruction-in-gcm-mode-paper.pdf
- [GK10] Gueron, S., & Kounavis, M. (2010). Efficient implementation of the Galois Counter Mode using a carry-less multiplier and a fast reduction algorithm. *Information Processing Letters*, *110*(14-15), 549-553.
- [GL17] Gueron, S., & Lindell, Y. (2017, October). Better bounds for block cipher modes of operation via nonce-based key derivation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1019-1036).
- [GLNP18] Gueron, S., Lindell, Y., Nof, A., & Pinkas, B. (2018). Fast garbling of circuits under standard assumptions. *Journal of Cryptology*, *31*(3), 798-844.

References (3)

- [GLL19] Gueron, S., Langley, A., & Lindell, Y. (2019). AES-GCM-SIV: Nonce misuse-resistant authenticated encryption. RFC 8452. https://datatracker.ietf.org/doc/html/rfc8452
- [GM16b] Gueron, S., & Mouha, N. (2016, December). Simpira v2: A family of efficient permutations using the AES round function. In *International Conference on the Theory and Application of Cryptology and Information Security* (pp. 95-125). Springer, Berlin, Heidelberg.
- [GM17] Gueron, S., & Mouha, N. (2017). SPHINCS-Simpira: Fast Stateless Hash-based Signatures with Post-quantum Security. In *Cryptology ePrint Archive, Report* 2017/645 https://ia.cr/2017/645
- [HKR15] Hoang V.T., Krovetz T., Rogaway P. (2015). Robust Authenticated-Encryption AEZ and the Problem That It Solves. In: *Oswald E., Fischlin M. (eds) Advances in Cryptology -- EUROCRYPT 2015. EUROCRYPT 2015*. Lecture Notes in Computer Science, vol 9056. Springer, Berlin, Heidelberg.
- [KR21] Krovetz, T., Rogaway, P. The Design and Evolution of OCB. *J Cryptol* 34, 36 (2021). https://doi.org/10.1007/s00145-021-09399-8
- [KLMR16] Kölbl, S., Lauridsen, M. M., Mendel, F., & Rechberger, C. (2016). Haraka v2–efficient short-input hashing for post-quantum applications. *IACR Transactions on Symmetric Cryptology*, 1–29.