Intel AES-NI

•••

Ram Longman and Prathibha Rama

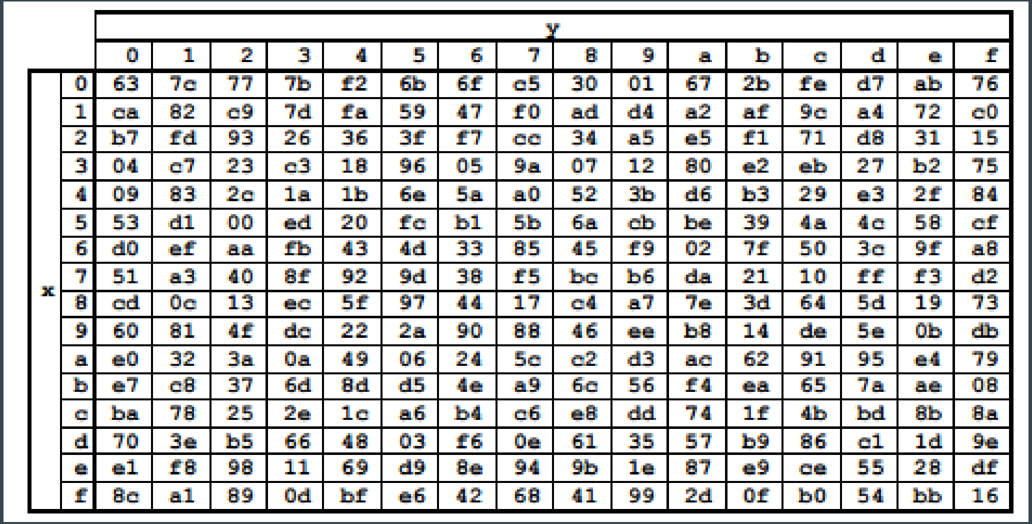
Agenda

Advance Encryption Standard Background

- Encryption Standard adopted by the U.S. government starting in 2001
- Widely used to protect network traffic, personal data, and corporate IT infrastructure
- Symmetric block cipher that encrypts/decrypts data through several rounds
- Four basic building blocks of the AES algorithm include: SubBytes, ShiftRows, MixColumns, and AddRoundKey

SubBytes

- Non-linear byte substitution
- SubBytes works by operating independently on each byte of the Plaintext using a substitution table or an S-Box (non-linear)

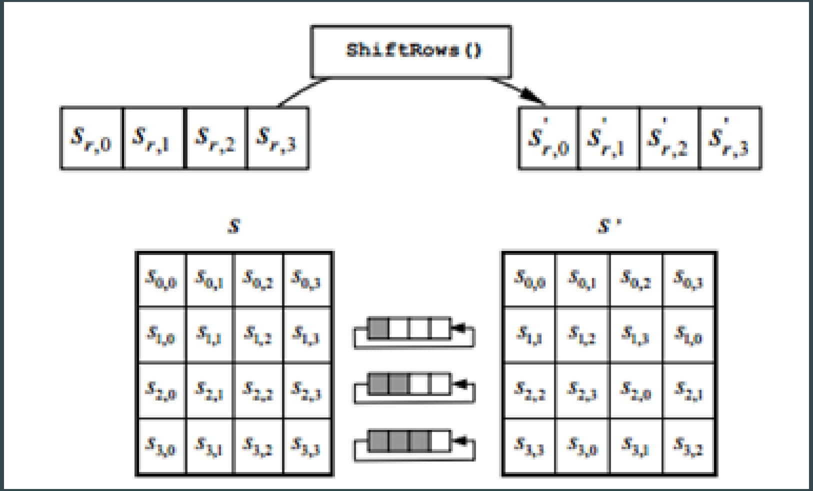


		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
x	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

ShiftRows

- ShiftRows works by cyclically shifting the bytes in the rows of the Plaintext
  - first row is left alone, second row is shifted to the left by one space, third row is shifted to the left by two spaces, fourth row is shifted to the left by three spaces

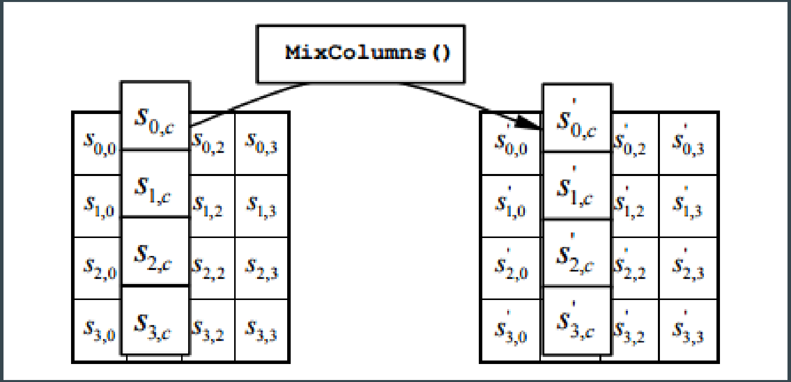


ShiftRows()
$s_{r,0}$ $s_{r,1}$ $s_{r,2}$ $s_{r,3}$
$s'_{r,0}$ $s'_{r,1}$ $s'_{r,2}$ $s'_{r,3}$
s
s'
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,1}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$

MixColumns

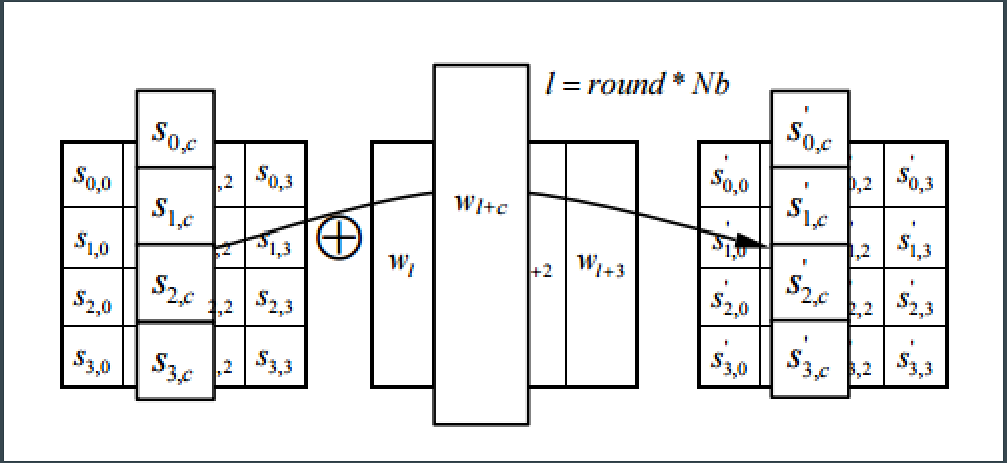
- MixColumns works by operating on the Plaintext column by column. In this case, each column is treated as a four-term polynomial over GF(2^8)

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < Nb.$$

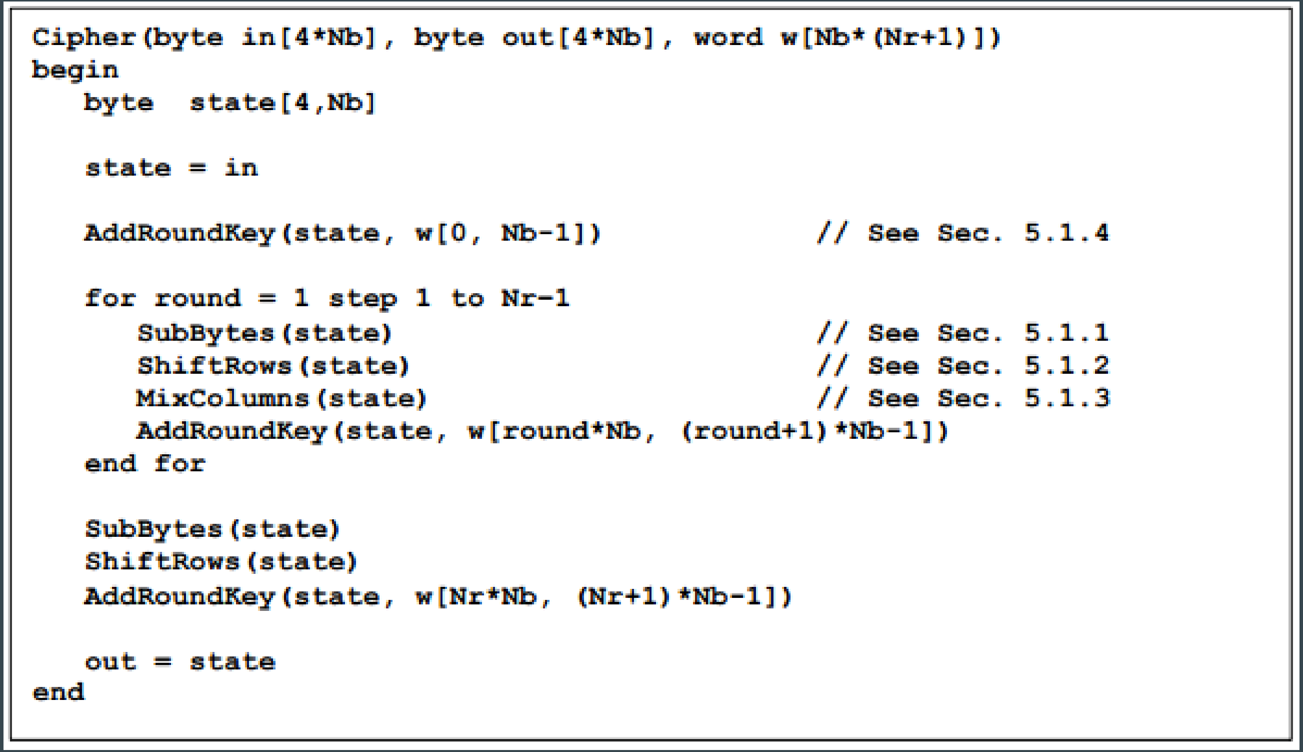

MixColumns()
$s_{0,c}$
$s_{1,c}$
$s_{2,c}$
$s_{3,c}$
$s_{0,0}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,c}$
$s'_{1,c}$
$s'_{2,c}$
$s'_{3,c}$
$s'_{0,0}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,2}$ $s'_{3,3}$

AddRoundKey

- AddRoundKey works by performing a bitwise XOR operation between the Plaintext and a Round Key



Algorithm Encryption

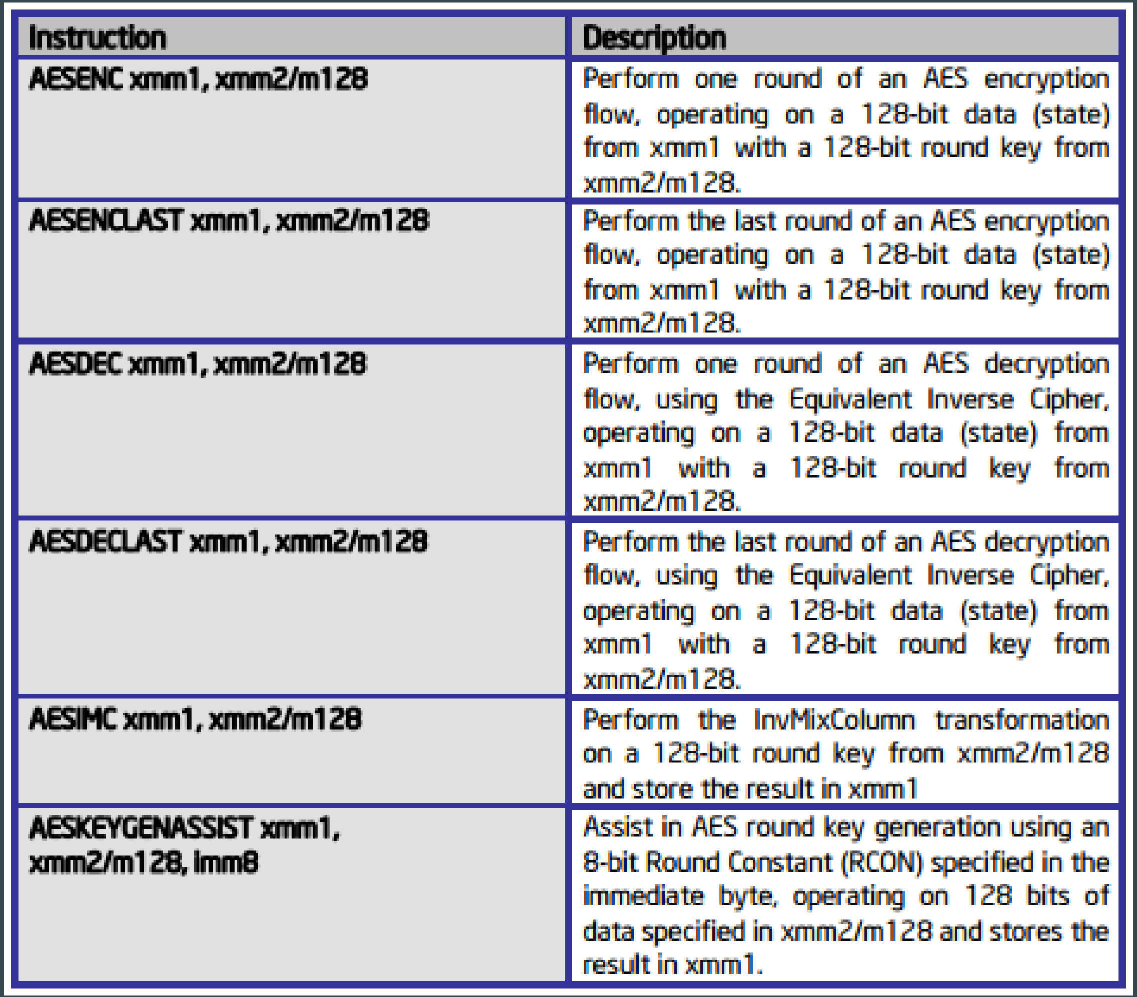


```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])             // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                         // See Sec. 5.1.1
      ShiftRows(state)                        // See Sec. 5.1.2
      MixColumns(state)                       // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Intel AES-NI Overview

- Available beginning with 2010 Intel Core processor family 2010 Intel microarchitecture codename Westmere
- Designed to implement complex and performance intensive steps of AES using hardware
- The use of hardware accelerates the performance of an implementation of AES by 3 to 10x over a completely software implementation
- Flexibility to support all usages of AES: standard key lengths, standard modes of operation, etc.
- Six Instructions:
  - Four Instructions- accelerating encryption/decryption
  - Two Instructions- round key generation

AES-NI Instructions



Instruction	Description
AESENC xmm1, xmm2/m128	Perform one round of an AES encryption flow, operating on a 128-bit data (state) from xmm1 with a 128-bit round key from xmm2/m128.
AESENCLAST xmm1, xmm2/m128	Perform the last round of an AES encryption flow, operating on a 128-bit data (state) from xmm1 with a 128-bit round key from xmm2/m128.
AESDEC xmm1, xmm2/m128	Perform one round of an AES decryption flow, using the Equivalent Inverse Cipher, operating on a 128-bit data (state) from xmm1 with a 128-bit round key from xmm2/m128.
AESDECLAST xmm1, xmm2/m128	Perform the last round of an AES decryption flow, using the Equivalent Inverse Cipher, operating on a 128-bit data (state) from xmm1 with a 128-bit round key from xmm2/m128.
AESIMC xmm1, xmm2/m128	Perform the InvMixColumn transformation on a 128-bit round key from xmm2/m128 and store the result in xmm1
AESKEYGENASSIST xmm1, xmm2/m128, imm8	Assist in AES round key generation using an 8-bit Round Constant (RCON) specified in the immediate byte, operating on 128 bits of data specified in xmm2/m128 and stores the result in xmm1.

Security Benefits

- Runs data-independant time: eliminates major timing attacks
- Performs encryption and decryption completely in hardware without the need for software look up tables: eliminates major cache-based attacks i.e. side-channel attacks
- Reduced code size: reduces risk of inadvertent introduction to security flaws i.e. side channel leaks

Performance Benefits

- AES encryption provides a significant speedup to encryption and decryption of bulk data
- At an algorithm level AES-NI provides significant speedup of AES
- For non-Parallel modes of AES operation (CBC encrypt), AES-NI can provide 2-3 fold gain in performance over a completely software application
- For parallel modes of AES operation (CBC-decrypt, CTR), AES-NI can provide 10x improvement over a completely software application
- Performance speedup would still be significant in scenarios where pipelined operation is impossible

Relative Cost of Key Expansion

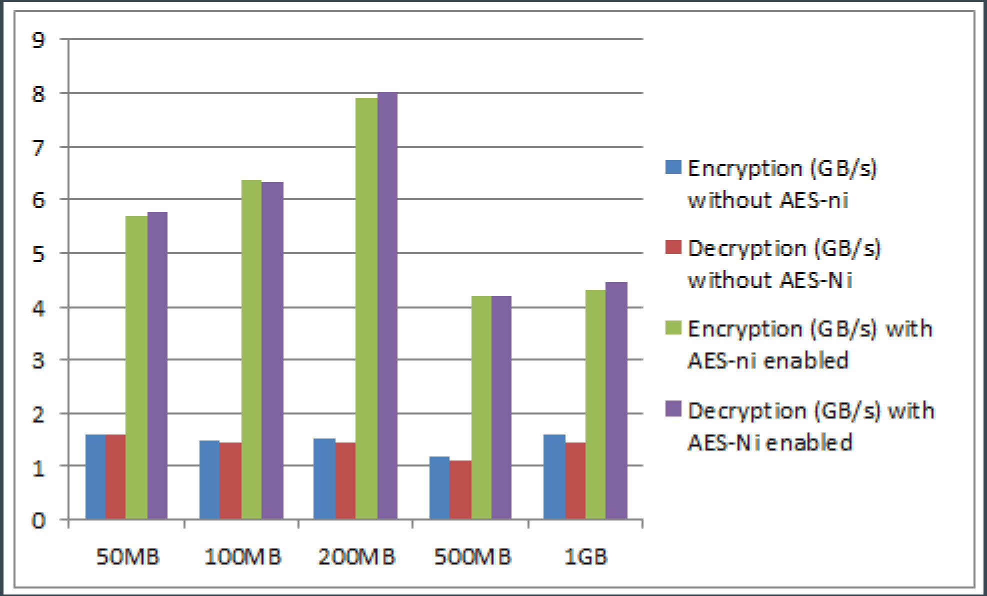
- AES architecture optimized for security and performance in applications where many block encryptions are performed with the same key (disk, network encryption)
- AESKEYGENASSIT and AESIMC facilitate key expansion without lookup tables
- AESKEYGENASSIT and AESIMC are faster than software only key expansion
- Unrolling key expansion code, improves performance

Optimize AES for better performance in parallel mode

- Significant performance enhancement can be achieved by reordering computations
- Helps take advantage of parallelism in parallel modes of operation: ECB, CTR, and CBC-decrypt
- Hardware that supports four AES round instructions is pipelined- allows independent AES instructions to be dispatched every 1-2 CPU clock cycles
- As a results AES throughput can be significantly enhanced for parallel modes of operation

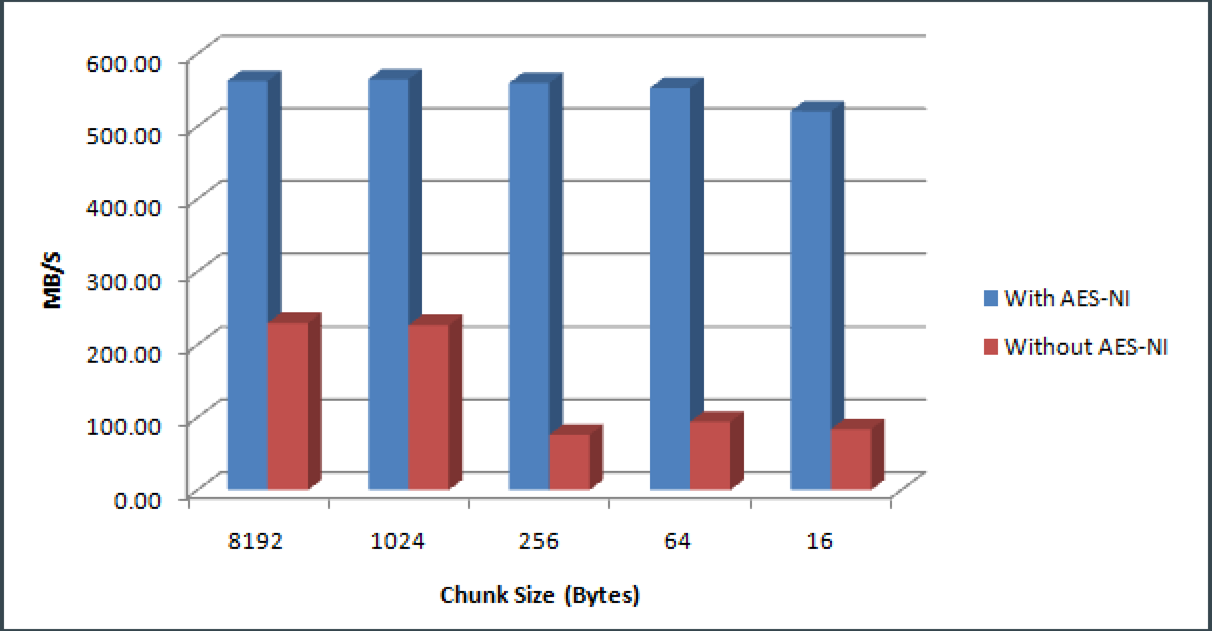
Performance Improvement General

- Test setup with AES-NI enabled provides 3 to 7 times more disk speed when encrypting and decrypting files with AES



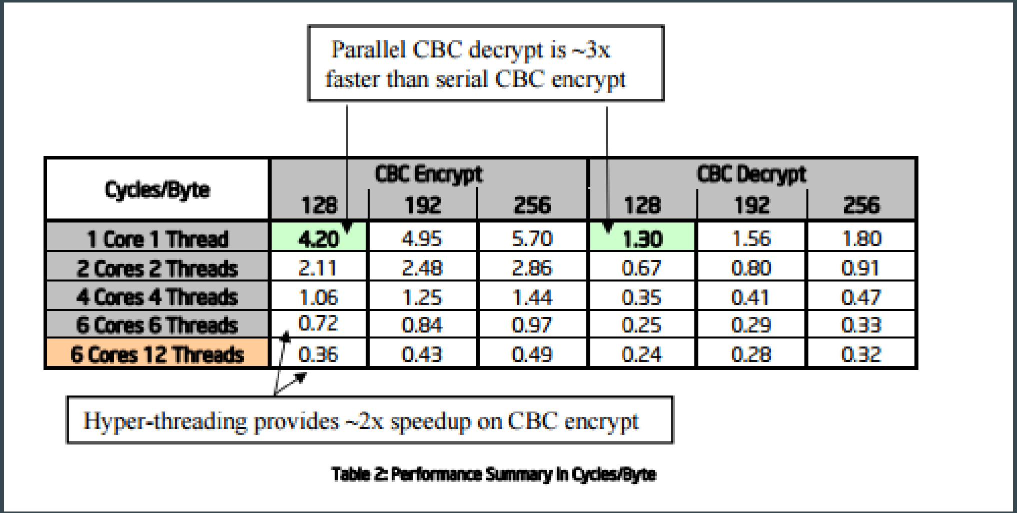
Performance Improvement i5 Processor

- Test setup using OpenSSl on Ubuntu Server 11.04 with i5-2300 processor



Performance Improvement i7 Processor

- Performance done in CBC mode measured on the Intel Core i7 Processor Extreme Edition. I7-980x
- Done for variable number of cores and threads
- Ues highly optimized implementations of AES



Parallel CBC decrypt is ~3x faster than serial CBC encrypt

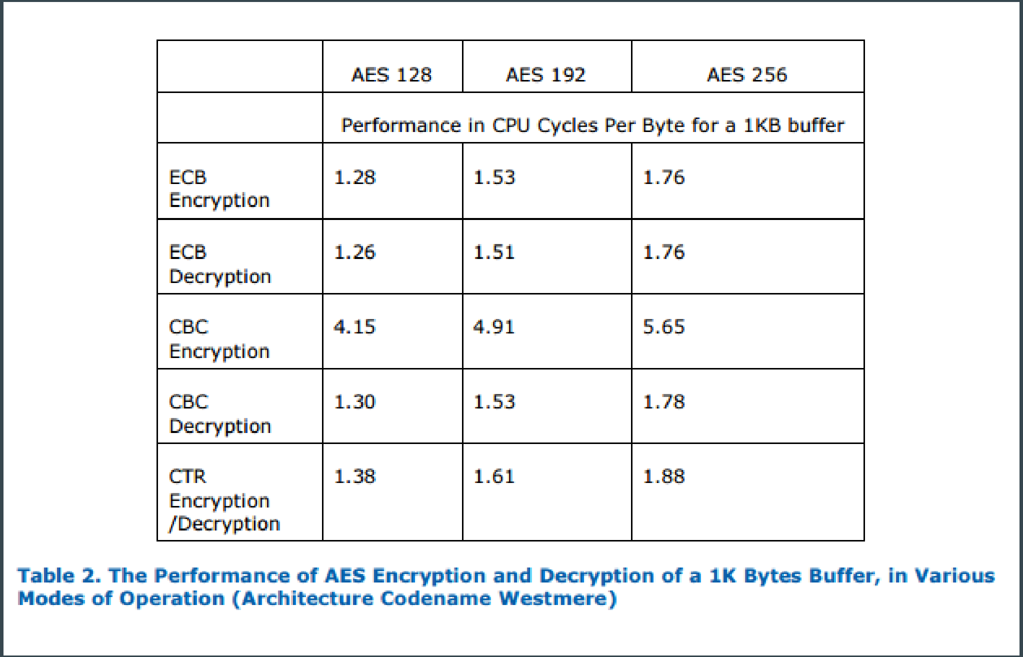
Cycles/Byte	CBC Encrypt			CBC Decrypt		
	128	192	256	128	192	256
1 Core 1 Thread	4.20	4.95	5.70	1.30	1.56	1.80
2 Cores 2 Threads	2.11	2.48	2.86	0.67	0.80	0.91
4 Cores 4 Threads	1.06	1.25	1.44	0.35	0.41	0.47
6 Cores 6 Threads	0.72	0.84	0.97	0.25	0.29	0.33
6 Cores 12 Threads	0.36	0.43	0.49	0.24	0.28	0.32

Hyper-threading provides ~2x speedup on CBC encrypt

Table 2: Performance Summary In Cycles/Byte

Performance Improvement Westmere

- Experiments were carried out on a processor based on Intel Architecture code name Westmere



	AES 128	AES 192	AES 256
	Performance in CPU Cycles Per Byte for a 1KB buffer		
ECB Encryption	1.28	1.53	1.76
ECB Decryption	1.26	1.51	1.76
CBC Encryption	4.15	4.91	5.65
CBC Decryption	1.30	1.53	1.78
CTR Encryption /Decryption	1.38	1.61	1.88

Table 2. The Performance of AES Encryption and Decryption of a 1K Bytes Buffer, in Various Modes of Operation (Architecture Codename Westmere)

Conclusion

- Takes advantage of AES algorithm by combining the four instructions involved in one round of encryption
- Uses hardware to accelerate the execution of AES algorithms
- Improves performance by taking advantage of key expansion and parallel modes of operation
- Improves security as the new instructions help address side channel attacks on AES

References

https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni

https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set

https://software.intel.com/sites/default/files/article/165683/aes-wp-2012-09-22-v01.pdf

https://dirteam.com/sander/2012/12/14/five-must-have-hardware-components-to-get-the-most-out-of-windows-8-and-windows-server-2012/