Ryan Barron

Cryptology

Dr. Zhang

Discovering AES-NI Instructions

May 15, 2020

Introduction

In the modern computing world, data security is a large concern, especially in the government sector, but also in the business world. Having data security means that patents will not be stolen, surprising business tactics are not revealed, company salary information is not released, and state secrets are kept secret. Due to these requirements by the industry, the federal government needed a standard for encryption, and so the National Institute of Standards and Technology defined the Advanced Encryption Standard (AES) to be one of the standards in FIPS197 for symmetric encryption. Since the implementation of AES was in software, the Intel Corporation decided it would improve the operation of the algorithm by creating their instance of AES in their processors. This created the Advanced Encryption Standard – New Instructions. The new instructions have benefits in performance, security, reduced instruction count, and the ability to change with altered definitions of AES.

To begin my project, I read Intel's New AES Instructions for Enhanced Performance and Security by Shay Gueron. He was part of Intel Corporation's Mobility group and the University of Haifa at the time of writing. Gueron describes the governmental and general interest in the algorithm, and how intel is seeking to improve several aspects of AES runtime and provides coding examples in Assembly and C++. The instructions run in assembly because the instructions run in the processor, which only uses assembly, rather than in code. He explains the design is significant for intel because incorporating the instructions take silicon space on the processor, which is expensive to produce, and must be maintained for backward compatibility with newer processors in the future. Further, he explains the instructions are singular for a specific part of the AES algorithm, such that they can be redesigned in the future for alternate algorithms. This is important to the need for backward compatibility, which is not as much of an issue if the system will not become legacy since it will have the ability to adapt to the needs of newer designs. (Gueron, 2009)

Advanced Encryption Standard

After reading through Shay Gueron's paper, I thought it would be helpful to state the operation of the AES algorithm so that when the AES-NI instruction made improvements, the changes would be understood. So, to encrypt using AES, the user must have a key to the standardized lengths. These include 128 bits, 192 bits, or 256 bits. The key will not be used directly to encrypt the data, because it could be discovered easily. Therefore, the AES standard calls for the expansion of the input key into a round key. If the input key is 128 bits long, the key expansion reads it as 4 words, which is 16 bytes, as every word is composed of 32 bits. The output is a 44 element long 32-bit-word array, which is also 1404 bits. To create this array, the original key placed at the first four words of the output array, and then the fourth word is created by XORing it with the word in the position four back, at index 0, and the previous word, at index 3, run through a special function that rotates the bytes, the calls a substitutes byte function, which is a lookup table that is run on all 4 bytes in the word, which uses the first 4 bits as the row, and the last 4 bits as the column to change the contents of the byte to the value at the destination. Then the bytes are XORed with a round constant. This special function is only run when the word in the expanded key is a multiple of 4. Otherwise, the word is XORed with the previous word, index - 1, and the word in the 4^{th} back position, index - 4. The output is in 44 words because every round of the AES encryption takes 4 of the words to use as a round key, which forms 128 bits. (Therithal info and Chennai, 2018-2020). For 128 bits there are 11 round keys formed, but the first one is for the initialization stage, as the 128-bit standard calls for 10 rounds. The 192-bit standard calls for 12 rounds, and the 256-bit standard calls for 14 rounds. Similarly, the round keys will be expanded in their appropriate sizes for the larger key sizes as well. In an AES round, there is a series of steps that occur, which are substitute bytes, shift rows, mix columns, and then XOR in the round key. The substitute byte works in the same way as the special function. The shift rows treat the 128 blocks as an array of 16 bytes. The first row is left-shifted 0, the second row left-shifted 1, the third row is left-shifted 2 and the last row is left-shifted three (Finjan Team, 2017). For example, the following array, [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15], becomes [0,1,2,3,5,6,7,4,10,11,8,9,15,12,13,14].

Advanced Encryption Standard – New Instructions

The intel corporation realized that AES is a standard that is gaining in usage from many people, businesses, and governments. While the standard is cryptographically sound, there are still computations to be made, and keys to guard. The computations still cause systems to take time to calculate, which reduces the run time of a program or system. Similarly, the AES uses lookup tables for keys, which have side-channel attacks. Finally, the programmers still must write the code to run, which takes production time. The culmination of these factors caused the inspiration for AES-NI.

There are 6 instructions designated as the AES-Ni instructions. They are 3 groups of 2 instructions, by function. The first two are for key generation and key management: AESKEYGENASSIST and AESIMC. The first, AESKEYGENASSIST is used for generating the round keys. The second, AESIMC, is for converting the encryption round keys to a form usable for decryption using the Equivalent Inverse Cipher.

Next in the pairs, instructions 3 and 4, is the pair dealing with the encryption of the data. The two instructions are AESENC and AESENCLAST. The first, AESENC is to encrypt a single round of encryption and combines four steps of AES into one instruction. The four steps are shift rows, substitute bytes, mix columns, and XOR the round key. The second instruction, AESENCLAST, is that same, but without mix columns, because it's not necessary on the last round.

The final pair of instructions, 5 and 6, are AESDEC and AESDECLAST. The first, AESDEC, has a similar structure to AESENC, except that it is a single round of decryption. The instruction still combines 4 steps AES, which are the inversion of shift rows, the inversion of substitute bytes, the inversion of mix columns, and XORing the round key. The second and final instruction is AESDECLAST, which is similar to AESENCLAST, in that it has a similar function as the main round instruction without the inversion of the mix columns since the encryption didn't use it on the last instruction.

The performance increase is due to all the execution occurring inside the processor rather than executing parts and then searching tables for the next part of the AES to execute. This means there is no

bussing information or lookup times top add. Similarly, by combining multiple steps into one instruction, the amount of code that must be compiled and run is significantly shorter. These factors build up to speed up. The speedups Intel reports are that using AES-NI could be 2-3 times faster for non-parallel modes such as CBC, and as much as 10 times faster for parallel modes like CTR. These speedups are for exchanging a software implementation of the algorithms with a hardware implementation. Additionally, intel states the instructions achieve ~1.3 cycles per byte on a single-core Intel Core i7 for AES-128 in parallel modes.

In addition to the speedups, there is increased security. By moving the operation of AES into an operation that is exclusively in the hardware, there is no opportunity for a side-channel attack that is from manipulating the tables to reveal information about the AES keys. Specifically, there are 2 attacks: Evict & Time and Prime & Probe. The first attack, Evict & Time, works by removing data from the cache containing the lookup table, waiting for the encryption to run, then measure the time needed to run the encryption. This reveals information about the key because the index into the table is based on the key, and the further into the table the AES algorithm has to go, the larger the value for the aspect of the key relating to the lookup is. The second attack, Prime & Probe, is when the adversary fills the cache table with its data, waits for the encryption to run, then the adversary checks what is left inside the cache. Since the cache index is based on the key, information about the key is revealed to the adversary. Using AES-NI prevents these attacks, increasing the security of the AES algorithm.

Finally, the design by intel is such that it excludes microbranching. In architecture, branching is the source of large prediction calculations, where miscalculated branches are very expensive in terms of lost cycles. The instructions could have combined the four encryption and decryption instructions into single instruction with an option to specify which function to use. This would have caused micro branches, but intel intentionally dispersed the functions into multiple instructions to make the implementation as fast as possible.

Testing the Difference – Pycryptodome

Since the AES-NI instruction provides speedups in addition to reducing code and increasing security, it's possible to examine run times. To test this, I decided to use Pycryptodome with two algorithms that can be used with the new instructions: Cipher Block Chaining (CBC) and Counter Block Chaining (CTR).

With CBC, I tested a 25,204 KB file with more than 55 iterations. The tests were conducted at different load factors on my CPU, as I was attending lectures while running the tests. However, the tests were run in sequence so that the same loads on the CPU were run with AES-NI and without it. The average run time with AES-NI was 0.738 seconds, and the time without was .0946 seconds. This gave a speedup (old-time/new time) of 1.284 times.

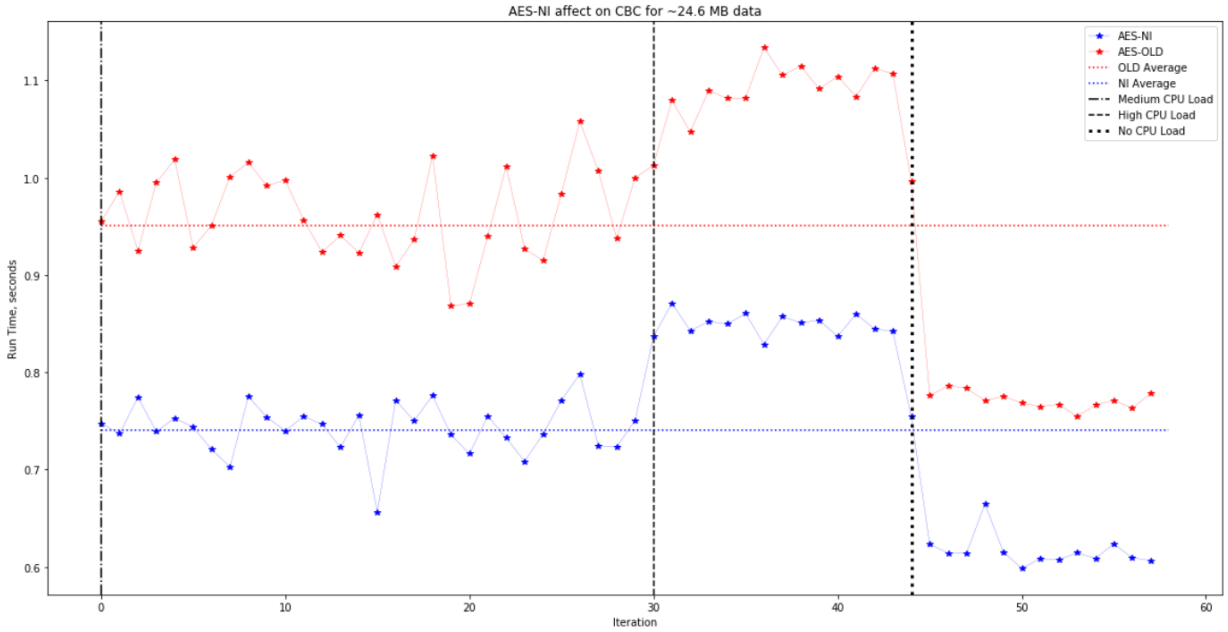

Figure 1. AES-NI in Pycryptodome with CBC, May 6, 2020

Similarly, with CTR, the file size was 554,486 KB and was run with more than 40 iterations, but with a constant (relative to CBC) CPU load, with no other major programs running in the background.

With AES-NI, the average run-time was 11.478 seconds, and without the new instructions, the run time was 15.425 seconds. This gave a speedup of 1.344 times.

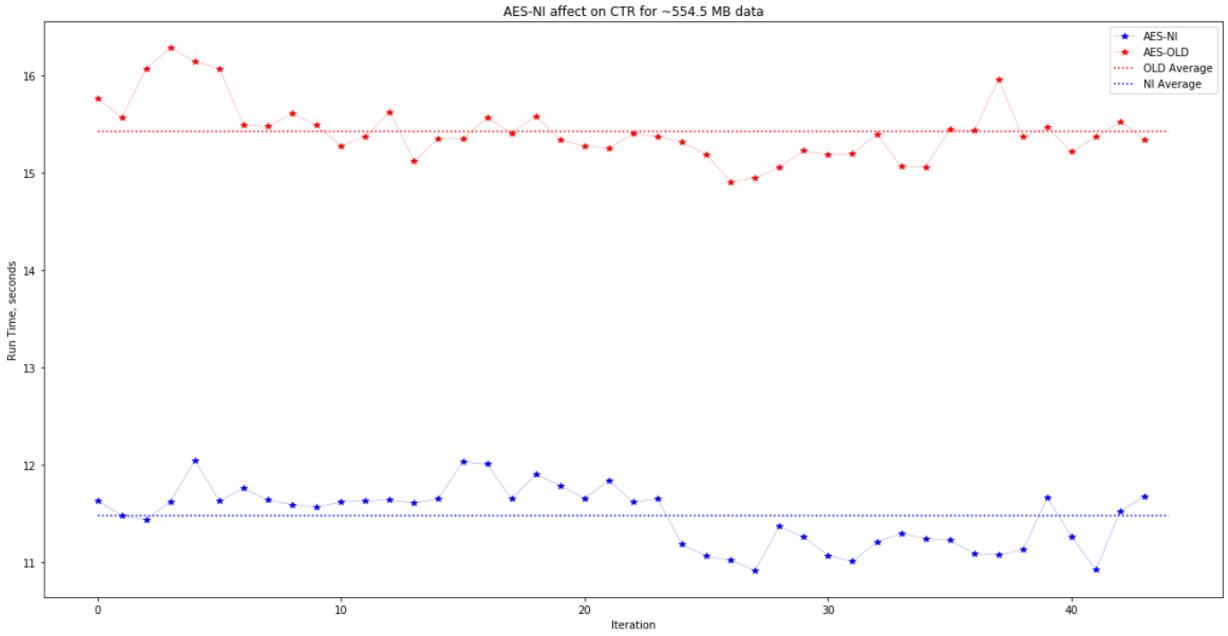

Figure 2. AES-NI on Pycryptodome with CTR. May 6, 2020.

To produce the codes, I consulted the Pycryptodome documentation, which has the same line of code for creating the encrypting object. For CBC, the line to activate AES-NI is AES.new(key, AES.MODE_CBC, use_aesni = True), and for CTR, the code is the same, other than the mode argument becomes AES.MODE_CTR. The graphs were generated using Matplotlib, and the data was stored in text files between iterations.

Intel states that the speedups that could be gained from using. The numbers I found with Pycryptodome were not quite as much as the numbers stated by intel which means much of the encryption process in python runs through intermediate code before utilizing the hardware's execution of the new instructions.

Conclusion

If I were to continue researching Pycryptodome, it would be interesting to implement more algorithms and compare the times between other algorithm implementations in the same code language. Additionally, I would like to implement the AES-NI instructions in C++ with all the same algorithms and files to compare the speed differences in handling the new instructions. Similarly, creating an assembly implementation and comparing it to the other languages would also be interesting. After this implementation, and I still had the resources to research this topic, I would like to see if there were any altered or extended algorithms that could be generated for increased security, such as an algorithm that could take a larger key than 256 by combining the AES-NI instructions and AES standard to maintain the ability to encrypt information in a world that continues to make decrypting easier for adversaries.

As far as AES-Ni itself goes, it is a practical addition that intel made to the data security industry and study that they intend to support for many new processors. The increased performance, security, simplicity, and reliability are significant benefits to Intel users and will have a wide variety of applications.

Bibliography

Rott, Jeffrey Keith. " Intel® Advanced Encryption Standard Instructions (AES-NI)." Intel Corporation, February 02 , 2012. https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni

Gueron, Shay. "Intel's New AES Instructions for Enhanced Performance and Security." 2009.

Pycryptodome. "Classic Modes of Operation for Symmetric Block Ciphers" Read the Docs. https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html?highlight=cbc#cbc-mode

Therithal info, Chennai. "AES Key Expansion". BrainKart, 2018-2020. https://www.brainkart.com/article/AES-Key-Expansion_8410/

Finjan Team. "Shift Row Transformation and Other Examples of Advanced Encryption". Finjan Blog, June 26, 2017. https://blog.finjan.com/shift-row-transformation-and-other-examples-of-advanced-encryption/

Taylor, Patrick. "Symmetric Block". University of Missouri. https://taylor.git-pages.mst.edu/index_files/Security/Content/10a-SymmetricBlock.html