High Performance Storage Encryption on Intel® Architecture Processors

Vinodh Gopal
Erdinc Ozturk
Wajdi Feghali
Jim Guilford
Gil Wolrich
Kirk Yap
Martin Dixon
IA Architects

Intel Corporation

White Paper
August, 2010
Executive Summary

There is a critical need for securing data-at-rest in enterprise storage devices at very high data rates. The IEEE has drafted the P1619 standard to support encryption of data-at-rest for block-based devices. This paper describes the performance characteristics of an optimized implementation of storage encryption, benefiting from the AES-NI set of instructions on Intel® processors based on the 32-nm microarchitecture.

This paper describes the performance characteristics of an optimized implementation of Storage encryption as defined in the IEEE P1619 Standard. In terms of throughput, we are able to perform AES128 Encryption in the XTS mode at the aggregate rate of ~18 Gigabits/sec for large buffers, on a single core of an Intel® Core™ i5 650 processor, with Intel® Hyper-Threading Technology.

For enterprise storage, more of the market is rapidly moving from unencrypted to fully encrypted storage over time. Storage appliances need to support line-rates of the order of ~50 Gigabits/second and will eventually require encryption at these rates, assuming all data is stored encrypted. We see that Intel® processors supporting the AES-NI set of instructions can comfortably satisfy the peak performance requirement with just three cores.

# Contents

Overview ................................................................................................................................. 4
Encryption of Data-at-rest ........................................................................................................ 4
  Background of Encryption .................................................................................................... 4
  The AES-XTS Encryption procedure ..................................................................................... 5
  Storage Encryption Requirements ......................................................................................... 5
Our Implementation of AES-XTS ............................................................................................ 6
Performance ............................................................................................................................ 6
  Methodology .......................................................................................................................... 6
  Results .................................................................................................................................... 7
Conclusion ............................................................................................................................... 9
References .............................................................................................................................. 10
Overview

The need for storage encryption is rapidly increasing with incidents of data loss containing sensitive personal or financial information related to medical records, credit-card/social-security numbers etc. Regulations are being put in place that mandate encryption for all sensitive information. Other regulations on companies are increasing, requiring them to preserve all records of transactions securely for extended periods of time, with strict penalties for violations. There is thus a critical need for securing data-at-rest in storage devices at very high data rates.

The IEEE has drafted the P1619 standard [1] to support encryption of data-at-rest for block-based devices. Encryption is performed by the AES block cipher in a mode of operation called XTS. AES-XTS performance can be significantly increased by using the AES set of instructions in Intel® processors based on the 32-nm micro-architecture. For simplicity we have described the encryption in this paper, however the decryption approach and performance are almost identical.

Encryption of Data-at-rest

Background of Encryption

The AES algorithm is a 128-bit block cipher used to encrypt data. It is defined with three key sizes of 128, 192 and 256 bits and can be applied with different modes of operation. The AES algorithm applies a number of rounds of processing on the data block, depending on the key size – 10, 12, 14 rounds for the key-sizes of 128, 192 and 256 bits respectively.

Data-in-flight is encrypted in the context of protocols such as IPsec or SSL/TLS. In these protocols, AES128 (128 bit keys) is the preferred option and modes such as Counter and CBC (Cipher-Block-Chaining) are used. These modes are amenable to very fast implementations using the AES set of instructions in Intel® processors, as described in [2].

Data-at-rest needs to be encrypted using a special mode such as XTS. In this paper, we show that the XTS mode can be implemented very efficiently as well, with performance close to the other fast modes such as Counter mode. Applications such as TrueCrypt [4] that perform full-disk-encryption using XTS, have seen a large speedup on their encryption after using the AES set of instructions in Intel® processors.
XTS (Xor-Encrypt-Xor Tweaked codebook with Ciphertext Stealing) mode refers to a narrow-block encryption scheme working on a block cipher size of 128 bits. It is based on a family of tweakable block cipher constructions, such that the same plaintext results in a different ciphertext, depending on the tweaks. The tweak is derived from a combination of a private key and a public value such as location of the data block. The goal of these modes is to get efficient performance for random access to data blocks, with security strengths comparable to modes such as CBC, but without requiring storage of Initialization Vectors which would cause storage expansion.

**The AES-XTS Encryption procedure**

The AES encryption is performed using the XTS mode of operation with 128 or 256 bit key-pairs as described in [1]. The XTS mode supports encryption of data buffers whose lengths are non-multiples of the AES cipher block size of 128 bits. The final partial block (< 128 bits) is processed in conjunction with the previous one using a method called ciphertext-stealing (CTS) [1].

A sector consists of a number of 128-bit blocks and an optional partial block. The sector has a logical address that is used to modify (tweak) the encryption process for the sector. XTS uses a pair of keys for encryption:

- a key to encrypt the sector address to generate tweak values
- a key to encrypt the data

For a single 128-bit data block P in a sector, the encryption can be described as \( C = T \oplus \text{AES-ENCRYPT}_{k1}(T \oplus P) \), where \( T = \text{AES-ENCRYPT}_{k2}(i) \ast \alpha^j \).

In this case, \( i \) is the address for the sector, \( j \) represents the position of the 128-bit block within the sector, \( \alpha \) is a primitive element of the Galois field \( \text{GF}(2^{128}) \) and the multiplication is defined in the same field as described in [1].

**Storage Encryption Requirements**

In addition to the requirements of the P1619 Standard, the storage encryption usages typically have some properties such as:

1. Requiring high-speed encryption/decryption to match wire-speeds of interfaces. Enterprise storage interfaces could require ~50 Gigabits/sec data rates. Encryption should not become a bottleneck in these systems.
2. Having additional DIF fields for data integrity with the actual data, that also need to be encrypted. Thus, for a data block size of 512 bytes, we also need to support encryption of 520 and 528 bytes to account for any added DIF fields.
3. Typical range of sector sizes varying between 512-8208 bytes.
4. Ability to perform encryption/decryption with key expansion or using pre-expanded keys.
Our Implementation of AES-XTS

We developed highly optimized implementations of the AES-XTS functions for 128 and 256 bit keys, with and without key expansion, for encryption and decryption, that would work efficiently on buffers of all sizes.

The main loop of the implementation processes four blocks in parallel, performing the AES encryption of these blocks while generating the tweak values for the next set of four blocks. The parallel processing of the AES operations is similar to that described in parallel modes such as counter mode in [2]. In the case where we require key expansion, we also stitch the initial key expansion code for the pair of keys with the encryption of the sector address i that is used to generate the initial tweak. The concept of function stitching has been described in detail in [3]. Here, we apply the same concept within the function, for distinct pieces such as key expansion or tweak generation, that are stitched with the processing of the AES encryptions.

Performance

The performance results provided in this section were measured on an Intel® Core™ i5 650 processor at a frequency of 3.20 GHz, supporting Intel® AES-NI. The tests were run with Intel® Turbo Boost Technology off, and represent the performance with and without Intel® Hyper-Threading Technology (Intel® HT Technology) on a single core. We present the results with both warm and cold data to reflect a variety of usage models.

Methodology

When a test for warm data is called, it is first run numerous times to warm up the cache.

When we require the test to work on cold data, we allocate a very large region of memory (> 256 MB) using the system call malloc, and initialize it to zero. We create an array of random pointer offsets into this region, such that no address is re-used. Since the region of memory is significantly larger than the size of the last-level cache, and the pointers we access in subsequent calls to the function appear random, we get cold cache behavior with TLB Misses as well.

The timing is measured using the rdtsc() function which returns the processor time stamp counter (TSC). The TSC is the number of clock cycles since the last reset. The ‘TSC_initial’ is the TSC recorded before the function is called. Then, the function is called for the specified number of times for data buffers of a given size. After the runs are complete, the rdtsc() is called again to record the new cycle count ‘TSC_final’. The effective cycle count for
the called routine is computed using:

\[ \text{# of cycles} = \frac{(TSC_{\text{final}} - TSC_{\text{initial}})}{\text{(number of iterations)}}. \]

When two such identical threads are run simultaneously, the number of cycles measured for the function represents the cycles consumed in the core for two data buffers (from each thread) and we calculate the cycles/byte accordingly. Thus, if each thread was computing XTS on buffers of size 1KB, then the net cycles/byte = \( \text{# of cycles} / 2 \times 1\text{KB} \)

**Note:** Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, Go to: [http://www.intel.com/performance/resources/benchmark_limitations.htm](http://www.intel.com/performance/resources/benchmark_limitations.htm)

**Results**

When we compared the performance of encryption and decryption, we found that they were within 1% on the average. Therefore, we only discuss the encryption results for simplicity.

**Figure 1: Performance (Cycles/byte) of AES128 XTS Encryption with Key Generation for different Buffer Sizes**
The figures show the performance in Cycles/byte as a function of the buffer size in bytes. We show the performance associated with generating the expanded set of round-keys for each buffer. In some usages, there may be a small set of keys that are re-used for many such blocks and remain mostly static; in such cases, we could get better performance by using pre-computed keys. We should note that the cost of key generation (100-140 cycles) is not very high, using some of the optimized methods in [2]. The cost of key generation does not depend on the size of the data buffer, and thus becomes a very small relative overhead for the larger sizes.

The shape of the graphs in Figure 1 and Figure 2 are very similar as expected. The data buffer sizes were chosen to be:

- 512, 520, 528,
- 1024, 1032, 1040,
- 2048, 2056, 2064,
- 4096, 4104, 4112,
- 8192, 8200, 8208

For each power-of-2 buffer size, we tested the performance on a buffer that had an additional 8 or 16 bytes to handle the cases of supporting data with DIF fields. In general, these three points tend to be very close to each other in performance, indicating that the incremental cost of processing an additional AES block with CTS is very low. In the figures, these points appear
clustered and can only be distinguished in the case of the smallest sizes 512, 520, 528 where overheads have the maximum effect.

The graphs start to flatten quickly at sizes beyond 4KB, approaching values in the range of **1.4-1.6 cycles/byte** for AES128 XTS for the different scenarios, for large buffers. Note that with Intel® HT Technology, the large buffer performance is better than **1.48 cycles/byte**.

The corresponding range of values tends to **1.9-2.1 cycles/byte** for AES256 XTS. Note that with Intel® HT Technology, the large buffer performance is better than **1.95 cycles/byte**.

Note that the ratio of large buffer performance for 128 and 256 bit keys, is **1.9/1.4, ~1.4X** – the same as the ratio of the number of rounds defined for these key-sizes.

In terms of throughput, a Single core can process AES128 XTS Encryption at the aggregate rate of ~ **18 Gigabits/sec** for large buffers.

**Figure 3: Detailed Performance (Cycles) of XTS Encryption for a 512-Byte Buffer**

<table>
<thead>
<tr>
<th>Key-size</th>
<th>Key Generation</th>
<th>Warm Data</th>
<th>Cold Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ST Cycles</td>
<td>HT Cycles</td>
</tr>
<tr>
<td>128</td>
<td>Key Generation</td>
<td>938</td>
<td>824</td>
</tr>
<tr>
<td>128</td>
<td>Precomputed Keys</td>
<td>837</td>
<td>756</td>
</tr>
<tr>
<td>256</td>
<td>Key Generation</td>
<td>1258</td>
<td>1093</td>
</tr>
<tr>
<td>256</td>
<td>Precomputed Keys</td>
<td>1116</td>
<td>1009</td>
</tr>
</tbody>
</table>

Figure 3 shows detailed performance for the smallest size of 512 bytes. We have shown the performance in absolute cycles to make it easier to see the differences caused by various factors. As an example, for 128-bit keys, the difference in Single-thread (ST) performance between the two rows for key generation and pre-computed keys is ~100 cycles. For 256-bit keys, we can see that the difference in ST performance for key generation is ~140 cycles. We can also see the difference between the columns for ST performance, which indicates that the cost of cold data compared to warm data is ~1000 cycles. Note that the difference between columns for Intel® HT Technology is much better, and the cost of cold data compared to warm data is ~500 cycles. Overall, for the case of cold data, Intel® HT Technology brings a 1.4X speedup over single-thread performance.

**Conclusion**

The paper describes the performance of the best-known implementation of AES-XTS on Intel® processors, using the AES set of instructions in Intel® processors based on the 32-nm micro-architecture. By applying function stitching techniques, we are able to perform AES128-XTS at ~ **1.4**
**Cycles/byte** on large buffers on a single core with Intel® Hyper-Threading Technology. In terms of throughput, a **single core** can process AES128 XTS Encryption at the aggregate rate of ~**18 Gigabits/sec** for large buffers. This performance is close to what we can achieve on some of the popular modes such as Counter mode.

Enterprise Storage appliances need to support line-rates of the order of ~50 Gigabits/second and will eventually require encryption at these rates, assuming all data is stored encrypted. **We see that Intel® processors supporting the AES-NI set of instructions can comfortably satisfy the peak performance requirement with just 3 cores.**

**References**


**Authors**

Vinodh Gopal, Jim Guilford, Erdinc Ozturk, Gil Wolrich, Wajdi Feghali, Kirk Yap and Martin Dixon are IA Architects with the IAG Group at Intel Corporation.

**Acronyms**

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA</td>
<td>Intel® Architecture</td>
</tr>
<tr>
<td>ILP</td>
<td>Instruction level parallelism</td>
</tr>
<tr>
<td>SIMD</td>
<td>Single Instruction Multiple Data</td>
</tr>
<tr>
<td>SSE</td>
<td>Streaming SIMD Extensions</td>
</tr>
</tbody>
</table>