**AES and 3DES Ciphers Using CBC and OFB Modes,**

**Implemented and Timed in C++**

**Introduction:**

This report will compare the relative efficiency of the AES and 3DES ciphers in both the CBC and OFB modes of operation, both by cipher and mode of operation. It will begin with a description of relevant features of the implementation of the two ciphers and two modes (written by the author for this project). There will then be a section describing the timing tests and results, and a section of disassembler-based analysis of the results.

**Implementation details:**

The complete source code for the implementations and testing, as well as testing data, is available at <https://github.com/georgehodgkins/ECE5397project>.

All the code used for this report, including the testing code, is written in C++ using only STL components. Internally, both the AES and DES implementations use C++ primitives to represent data and keys (i.e. the DES key and message are represented as uint\_64t, while the AES message and keys are represented as arrays of uint32\_t). This implementation is designed to favor computation time at the cost of memory usage and code size, so wherever possible cipher operations are implemented as hardcoded array lookups. The AES function is also written as a single template that allows the user to select any of the AES key sizes (it will not accept non-AES key sizes). Neither cipher function makes any use of dynamic memory.

The ciphers interface with the block mode functions using the STL bitset template, and messages are passed into the block mode functions using vector containers of bitsets. The block mode functions are entirely independent of the cipher used, taking block size, key size, and cipher encryption/decryption function pointer as template arguments. At the block mode level, bitwise operations are done using the overloads built into the bitset class. The initial vector for the block modes is generated using the STL random\_device method, which uses system-dependent hardware sources of entropy.

**Testing procedure:**

The source code used in the testing process is included in the repository linked above as ‘timing\_frontend.cpp’. The testing consisted of loading in a 3.764 KB data file (with varying contents) and successively encrypting and then decrypting the file’s contents repeatedly. The encryption and decryption operations were timed separately (from the point at which the mode of operation function was called to the point at which that function returned to the caller). For each test case, 8192 iterations of encryption and subsequent decryption were timed and recorded; this raw data is available in the ‘test\_data’ directory of the repository linked above (all times are in microseconds). Timing was done using the STL chrono library’s steady\_clock. The code was compiled for x86 with GCC version 6.3, using the highest level of optimization and the C++11 standard. For each of the eight tests (CBC or OFB with 3DES, AES-128, AES-192, or AES-256), the code was recompiled with the correct template arguments and output file name before execution. All manipulation, analysis, and visualization of the data after its collection was done using MATLAB.

**Results:**

A summary table of results is presented below, giving mean encryption and decryption times, the standard deviation of that mean, and the corresponding rate, for each cipher using each mode of operation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CBC | | OFB | |
|  | Encrypting | Decrypting | Encrypting | Decrypting |
| 3DES | 18.979 ms  (σ = 6.397)  [198.32 Kbps] | 18.978  (σ = 6.416)  [198.33 Kbps] | 18.970  (σ = 6.383)  [198.42 Kbps] | 19.230  (σ = 6.532)  [195.74 Kbps] |
| AES-128 | 8.248 ms  (σ = 1.967)  [456.35 Kbps] | 8.039 ms  (σ = 1.954)  [468.22 Kbps] | 7.614 ms  (σ = .733)  [494.35 Kbps] | 7.575 ms  (σ = .648)  [496.90 Kbps] |
| AES-192 | 9.885 ms  (σ = 2.079)  [380.78 Kbps] | 9.677 ms  (σ = 2.086)  [388.96 Kbps] | 9.166 ms  (σ = .691)  [410.65 Kbps] | 9.133 ms  (σ = .704)  [412.13 Kbps] |
| AES-256 | 11.523 ms  (σ = 2.277)  [326.65 Kbps] | 11.249 ms  (σ = 2.263)  [334.61 Kbps] | 10.670 ms  (σ = .826)  [352.76 Kbps] | 10.635 ms  (σ = .836)  [353.93 Kbps] |

Presented below are plots, comparing encryption and decryption times for 3DES and AES-256, using both modes of operation. Each point on the plots represents an average of a cluster of 32 iterations of encryption and decryption, to increase readability.

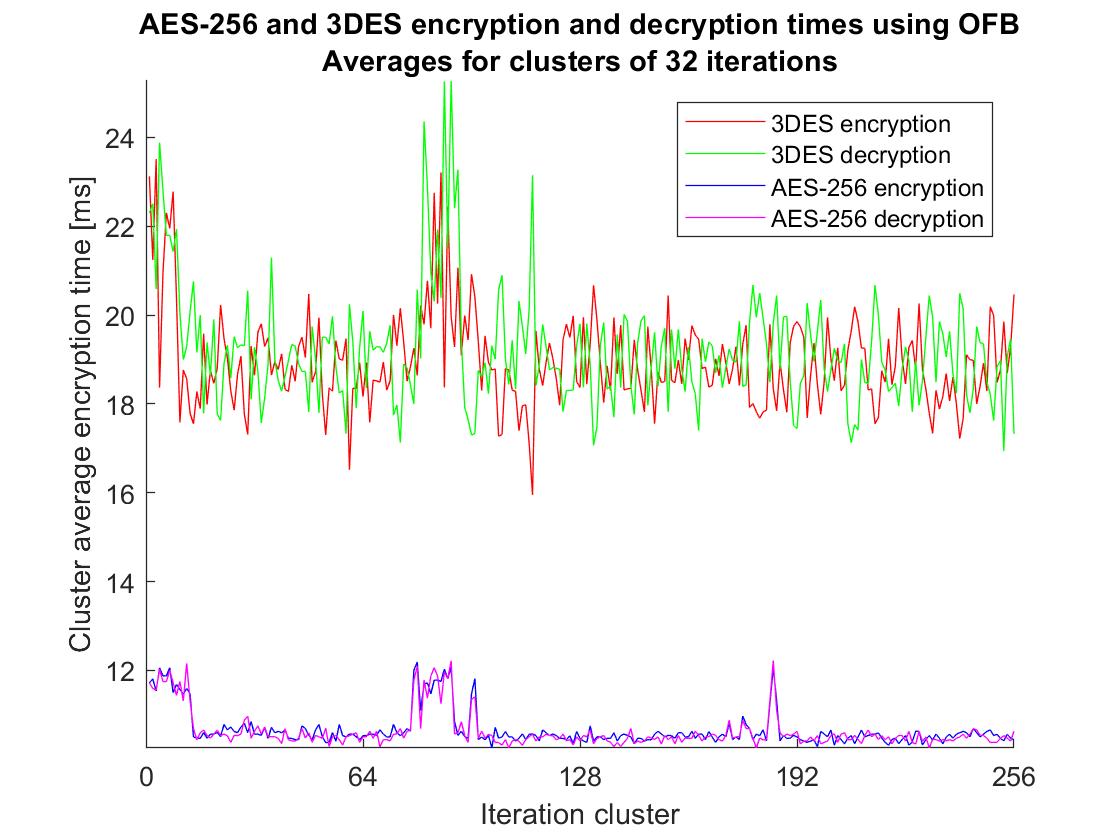


Figure 1: Comparison of the AES-256 and 3DES ciphers using the OFB mode of operation. Note that the tests were not done synchronously, so the large spikes in running time between the 64th and 128th clusters for both ciphers are purely coincidental (and do not appear for AES-192 or AES-128).

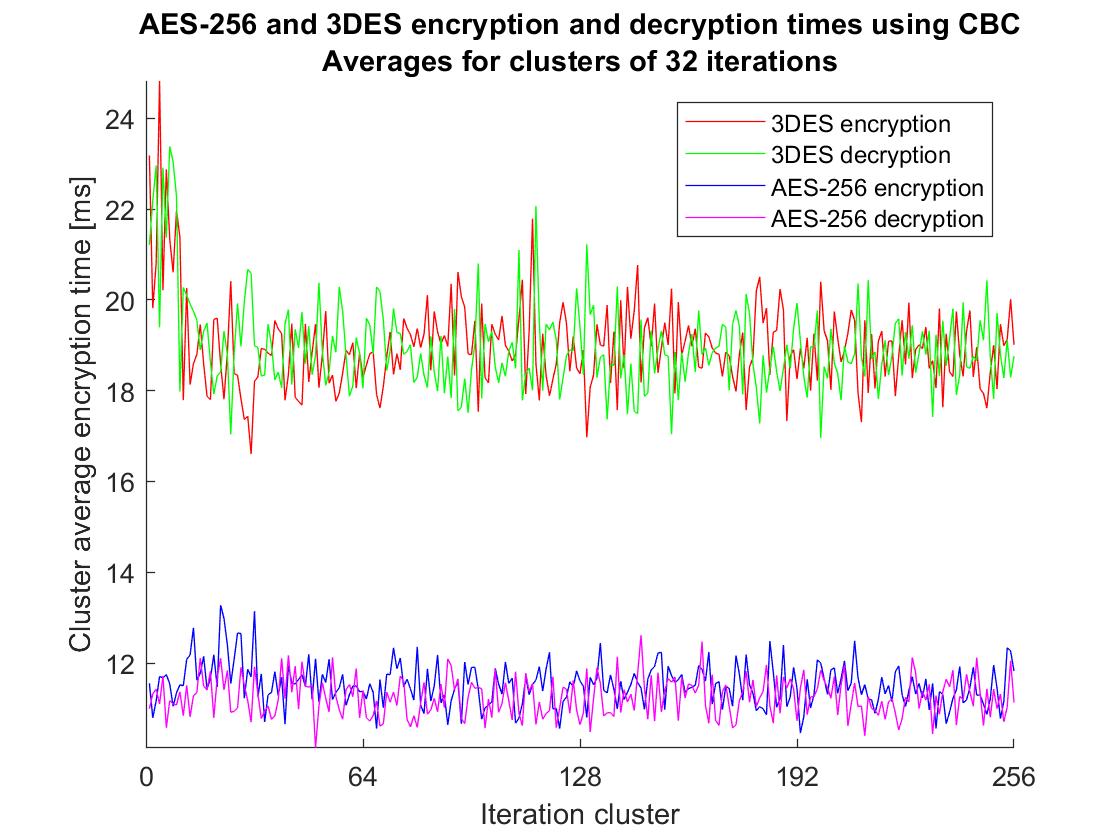


Figure 2: Comparison of the AES-256 and 3DES ciphers using the CBC mode of operation.

**Analysis:**

My implementation of the AES cipher performs significantly faster than my implementation of the 3DES cipher, which is typical of implementations of these ciphers on general-purpose hardware [1]. Disassembler-based analysis (using IDA Pro) of the two implementations gives a total of approximately 300,000 instructions for one 3DES block encryption, and approximately 240,000 instructions for one AES-256 block encryption (note also that the AES block size is twice that of DES, so the 3DES encryption has to be run twice to encrypt the same amount of data as AES-256). A detailed breakdown of the analysis is presented below in table form.

The instruction counts given are necessarily approximate. Where the execution of instructions depended on a bit being set or not, this bit was treated as a random binary variable with 50% probability of being true. “Entrance/exit overhead” refer to instructions allocating locals, initializing loop counters, and similar; in some places, this also includes other miscellaneous instructions in the code that do not fit neatly into any other category. Note that since DES encryption and decryption are very similar programmatically, differing only in the placement of operands and the order in which round keys are used, the same instruction count is used for both operations.

|  |  |
| --- | --- |
| **AES-256** | |
| Entrance overhead | *40* |
| Bitset to int conversion | *5448* |
| Round key generation | *111,963* |
| Initial keyAdd | *272* |
| Encryption round: | *216,048* (14 rounds) |
| Entrance overhead | *23* |
| subBytes | *224* |
| shiftRows | *252* |
| mixCols | *14,656* |
| addRoundKey | *272* |
| Exit overhead | *6* |
| **Total:** | **15,417** |
| Last encryption round  (no MixCols) | *761* |
| Int to bitset conversion | *5721* |
| Exit overhead | *50* |
| **Total** | **240,042** |

|  |  |
| --- | --- |
| **3DES** | |
| Entrance overhead | *11* |
| Subkey generation | *20,288* |
| Nested encryption/decryption | *279,114* (3 rounds) |
| Entrance overhead | *20* |
| Round key generation | *50,556* |
| Bitset to int conversion | *2724* |
| Encryption round | *37,008* (16 rounds) |
| Entrance overhead | *39* |
| E-box | *1092* |
| Key addition | *102* |
| S-box | *399* |
| P-box | *655* |
| Exit overhead | *26* |
| **Total** | **2297** |
| Int to bitset conversion | *2947* |
| Exit overhead | *6* |
| **Total** | **93,038** |
| Exit overhead | *124* |
| **Total** | **299,537** |

**References:**

[1] A. Al-Tamimi, “Performance Analysis of Data Encryption Algorithms,” *cse.wustl.edu*, 2016. [Online]. Available: <https://www.cse.wustl.edu/~jain/cse567-06/ftp/encryption_perf>. [Accessed: May 2, 2019].