AES Block Cipher Modes

1. Introduction 2

2. Programs 3

Key Generation 3

2.1.1. Run 3

2.1.2. Input 3

2.1.3. Output 3

Encryption 3

2.1.4. Run 3

2.1.5. Input 3

2.1.6. Output 4

Decryption 4

2.1.7. Run 4

2.1.8. Input 4

2.1.9. Output 4

Test 4

3. Extra Credit 4

4. Software Libraries 5

Cryptopp - http://www.cryptopp.com/ 5

Timer.cpp - http://www.songho.ca/misc/timer/timer.html 5

5. Test Procedures 5

6. Performance Analysis 6

Overview 6

6.1. Detailed Effect of Key Size on Performance for Each Mode 6

6.1.1. ECB 6

6.1.2. CBC 7

6.1.3. OFB 8

6.1.4. CTR 9

6.1.5. CFB 10

6.2. Overall Effect of Key Size on Performance 11

6.2.1. Encryption 11

6.2.2. Decryption 12

6.3. Overall Effect of File Size on Performance 12

6.3.1. Encryption 12

6.3.2. Decryption 15

7. Conclusions 17

# Introduction

This document presents a view into and an analysis of the block cipher modes for the block cipher AES. AES is the Advanced Encryption Standard with a fixed block size of 128 bits. We will be analyzing the various block cipher modes for AES. The modes to be analyzed are ECB, CBC, OFB, Counter, and CFB. Each of these modes have different levels of security and efficiency. The primary focus of the program presented is to analyze the efficiency of each block cipher mode and compare them with each other. The comparisons will be made encrypting messages of varying lengths and with keys of varying lengths.

**ECB**

Electronic codebook mode is a mode that breaks down the plaintext into a series of blocks and then passes the plaintext block into an encryption algorithm with another input of a private key. Each block of plaintext is treated in this manner without any block relying on the previous block. Because of this, ECB is a deterministic mode and does not provide indistinguishability. This mode is generally not used as it is not a very secure method but is included in the analysis as it is a valid mode and is needed for completeness.

**CBC**

Cipher Block Chaining mode breaks the plaintext down into block of plaintext. For the first block, the plaintext is XORed with an IV (initialization vector - generally a random value) and the output form this is sent to the encryption algorithm along with the key. For the remaining blocks, the output ciphertext from the previous encryption is then XORed with the plaintext block for the input into the encryption algorithm along with the key. Since each block is linked to the previous block, the name chaining is used. This mode does provide indistinguishability with a chosen message attack.

**OFB**

Output feedback mode makes a stream of keys that are XORed with the plaintext to create the ciphertext. The plaintext message is broken down into blocks. On the first block, an IV and the key are input into the encryption algorithm and the output is XORed with the plaintext block to generate the ciphertext. The output is also sent as the input of the next run with the key. The generation of the key stream can be ran independently of the XORing with the plaintext and so can be precomputed to allow for parallelization of the XOR process when the plaintext is available. This mode is also secure with the sense of indistinguishability for chosen message attack.

**CTR**

Counter mode creates a key stream similar to that of OFB. This mode will take an IV as the first input along with the key to the stream and then each additional run will increment the initial IV as a counter does. The output of each round is XORed with the plaintext block to generate the ciphertext. As with OFB, CTR is secure in the sense of indistinguishability for chosen message attack.

**CFB**

Cipher Feedback mode combines the ideas from CBC and OFB. The first round will take an IV and key as the input into the encryption algorithm and the output will be XORed with the plaintext block to generate the ciphertext. For each additional round, the ciphertext from the previous round will be used as the input along with the key into the encryption algorithm of that round. The plaintext block is then XORed with the output from that to generate the ciphertext. As this is closely related with CBC and OFB, this mode should be secure in the sense of indistinguishability for chosen message attacks as long as the encryption algorithm is a pseudorandom permutation.

# Programs

The program used for analyzing these modes is written in C++ with three programs: keygen, encrypt, decrypt.

## Key Generation

The program keygen generates the private key that will be used for encryption and decryption. This key is generated with an random generator and will generate multiple key lengths.

### Run

To run keygen, at the command line type:

#>./keygen [-hv] [-s key\_size\_in\_bytes (16, 24, 32) default=16]

If no key size is defined then the default of 16 will be used. Valid values for key size are 16, 24, and 32 which correspond to key sizes of 128 bit, 192 bit, and 256 bit respectively.

### Input

The input is just the size of the key desired. Either 16, 24, or 32.

### Output

The output will be a text file called *key\_<keysize>.txt* which will hold the key in hex format.

## Encryption

The program encrypt will encrypt plaintext into ciphertext with AES using the modes of operation that were discussed in the introduction. Each mode is timed to allow for analyzation of the running of each mode.

### Run

To run encrypt, at the command line type:

#>./encrypt [-v] [-k key\_file=key.txt] [-s key\_size (16, 24, 32)] [-m mode (CBC, OFB, CFB, ECB, CTR)] [-l loop\_count] [-p plaintext\_file] [-c ciphertext\_file]

### Input

The input to the encrypt program is the key file and the plaintext file along with the options for mode of operation.

### Output

The program outputs a file with the ciphertext based on the mode of operation. Also, timing information is output for analyzation of the modes of operation.

## Decryption

The decrypt program will take the ciphertext and decrypt it into plaintext using AES and a mode of operation as discussed in the introduction. The program times each mode of operation so that the mode can be analyzed.

### Run

To run encrypt, at the command line type:

#> decrypt [-bv] [-k key\_file] [-s key\_size (16, 24, 32)] [-m mode (CBC, OFB, CFB, ECB, CTR)] [-l loop\_count] [-p plaintext\_file] [-c ciphertext\_file]

### Input

The input to the program decrypt is the key file, the ciphertext file and the mode of operation.

### Output

The program decrypt outputs a plaintext file based on the ciphertext and the mode of operation used. Also, timing information is output for analysis of the modes of operation.

## Test

Test is a script that verifies correct operation of the encrypt/decrypt program pair. This script will run encrypt using all block cipher modes against a 100K test file, save the encrypted files, and then decrypt each file into separate plaintext files. Finally it performs a binary comparison from the original test file and each of the decrypted plaintext files and outputs any differences discovered. A successful result is when no differences are detected between the original and resulting files.

To run test, type:

#> ./test

# Extra Credit

Our program implements the following enhancements over the final project requirements and which are eligible for extra credit.

* Multiple key lengths
* An additional block cipher mode Cipher Feedback (CFB) not required in the project assignment
* A “performance analysis” mode that allows the user to run the encrypt and decrypt programs with the following features:
  + Perform the encrypt/decrypt cycle a specified number of iterations
  + Output performance data to the command line in a delimited format which can be easily imported into a spreadsheet
  + Enable all block cipher modes with each creating a unique plaintext or ciphertext file (e.g. plaintextcbc.txt instead of the default plaintext.txt)

# Software Libraries

As the program is a C++ program, it was chosen to use the public library cryptopp for implementing the various modes of operation for AES.

## Cryptopp - <http://www.cryptopp.com/>

Cryptopp was used as it is an open source library well supported by the user community. It’s clever implementation of “pipelining” from “Source” to “Sink” while applying BufferedStreamTransforms makes the application of various transformations (Hexencode, Hexdecode, encrypt, decrypt) efficient to code. Additionally the API handles many of the detailed cipher technical requirements such as padding when necessary almost automatic.

## Timer.cpp - <http://www.songho.ca/misc/timer/timer.html>

An open source timer class was utilized to perform the performance measurements. Compared to other packages this code was easy to use and maintained an accuracy of 1 millisecond. Compared to system clock() calls this is an accuracy improvement of 15x.

# Test Procedures

The encrypt and decrypt programs used timing mechanisms in order to provide information on the running of each program. The analysis is based on the key size, the size of the plaintext, and the time it took to encrypt and decrypt the files for each mode of operation. Average run times over a series of 100 runs was used as the basis for the comparison. From this data Throughput was computed in the form of KB/s by dividing the input file size by the average runtime. By comparing throughput instead of average run times we are able to aggregate the data across the different modes of operation. For example using average times only it is not possible to compare a modes of operation across various file types as the average time is directly dependent on the size of the file. Using throughput however the attribute is no longer dependent on the file size.

Test files were created for the following sizes: 16 bytes, 32 bytes, 5 Megabytes, 10 Megabytes, and 256 Megabytes. The test results for the 16 and 32 byte files are not included in this report as their performance data fluctuated dramatically due to the allocation/deallocation of memory as the program loaded the crypto library into RAM.

Test files were generated using random characters to ensure that the data realistically represented actual binary files. The following command was used to generate each test file:

dd if=/dev/random of=testfile\_10M bs=10240 count=1

Tests were not performed using a dedicated test system and therefore the results are subject to influence by system and user processes running in the back- and foreground. For the most part test runs were started in the evening and ran through the night to minimize the impact of user processes however batch processes and maintenance tasks such as backups often run at night when the system idle processes are at their highest.

# Performance Analysis

## Overview

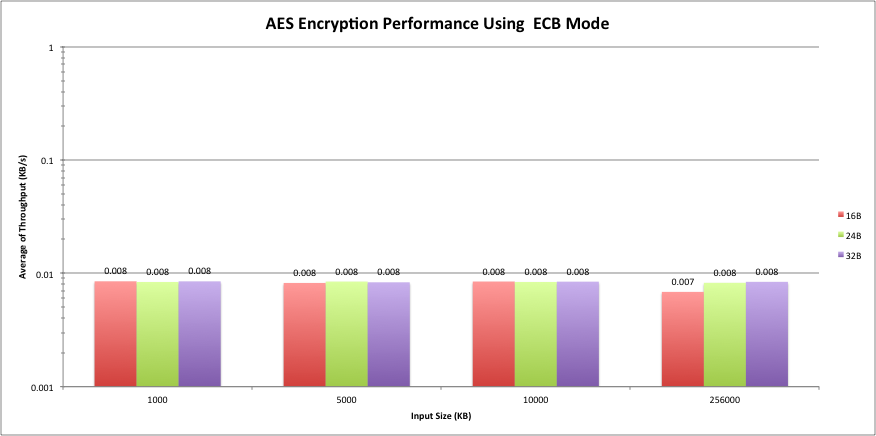
The analysis for the performance of each mode of operation is based on the comparison of each mode individually and then each mode comparatively. First we look at how each mode of operation handles the different key sizes with the different text file sizes. Then we show how each mode of operation compares with each other based on key size and file size. The goal is to see how the size of the key can affect the performance and how the size of the file can affect the performance.

### Detailed Effect of Key Size on Performance for Each Mode

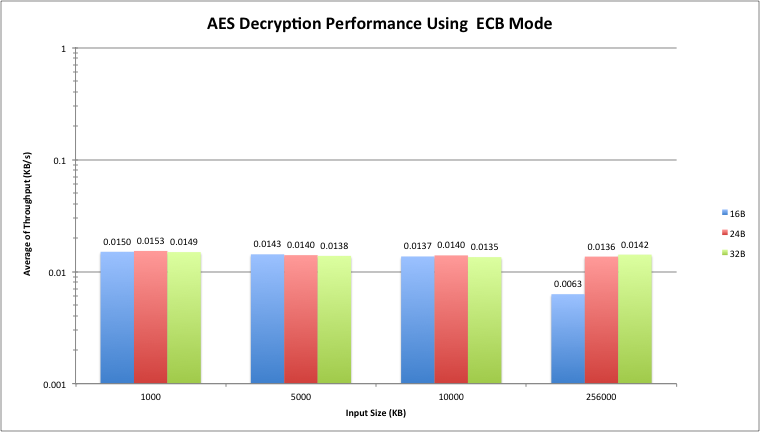
### ECB

This graph shows throughput performance based on key size for ECB mode of operation:

Encryption:



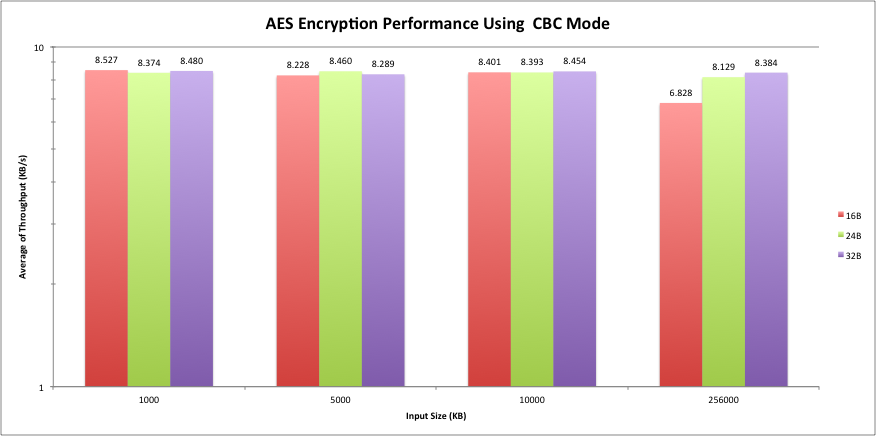
Decryption:



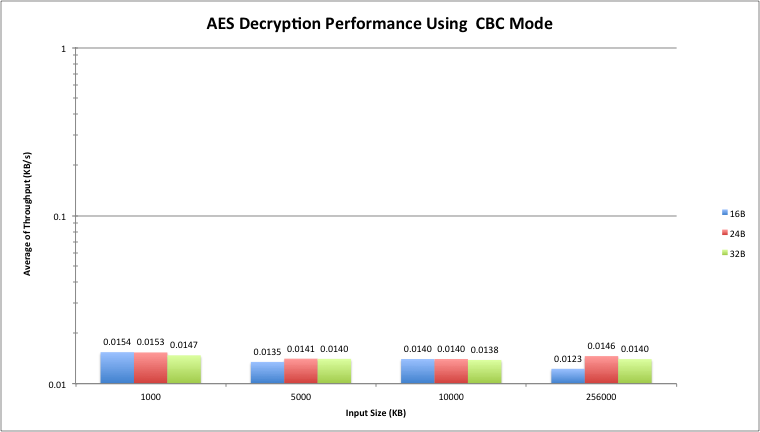
### CBC

This graph shows throughput performance based on key size CBC mode of operation:

Encryption:



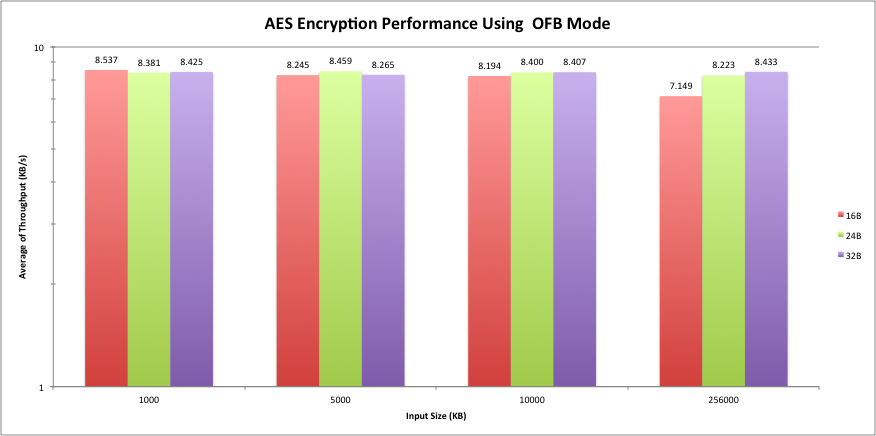
Decryption:



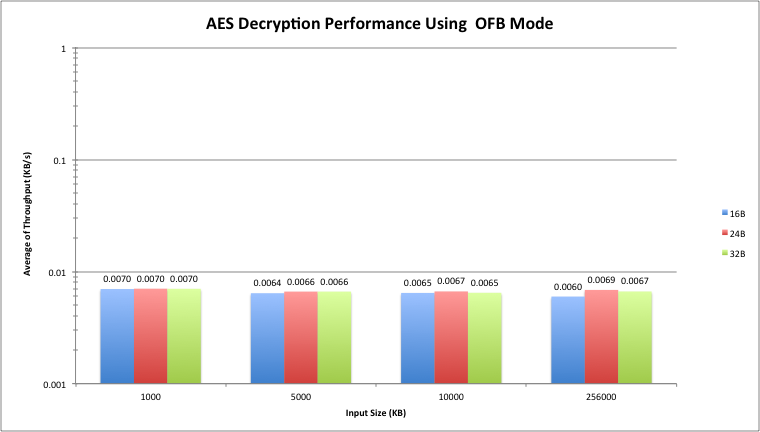
### OFB

This graph shows throughput performance based on key size OFB mode of operation:

Encryption:



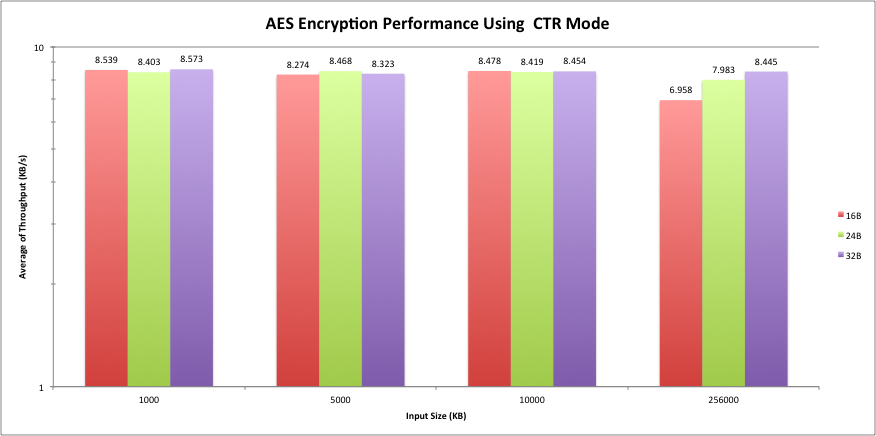
Decryption:



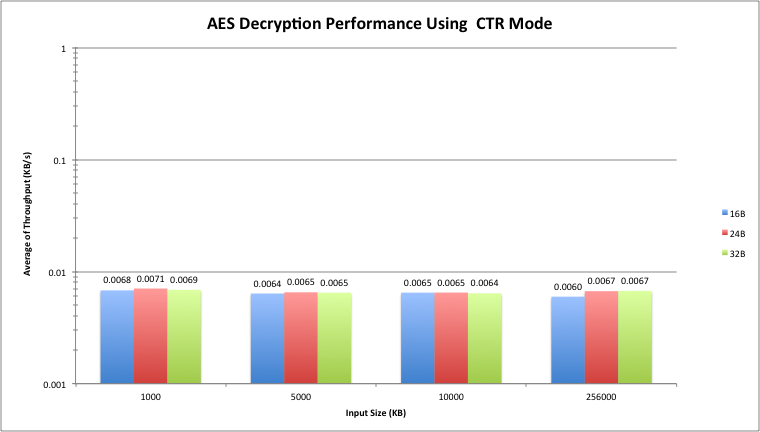
### CTR

This graph shows throughput performance based on key size CTR mode of operation:

Encryption:



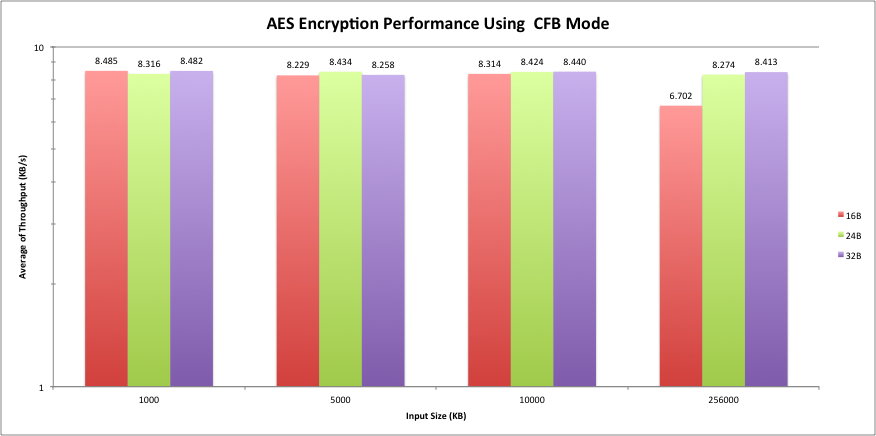
Decryption:



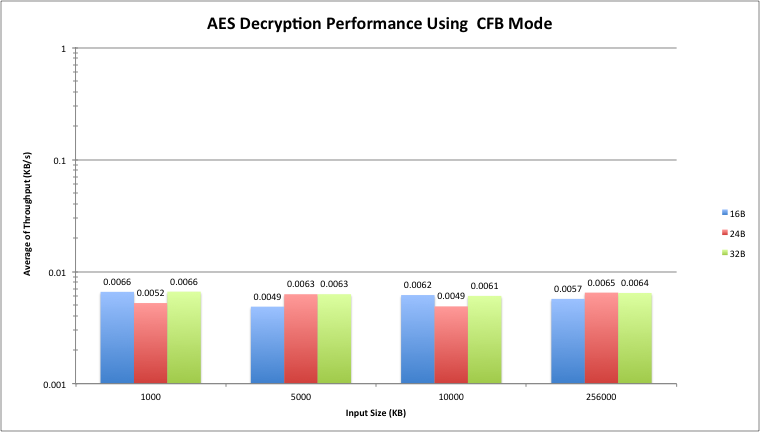
### CFB

This graph shows throughput performance based on key size CFB mode of operation:

Encryption:



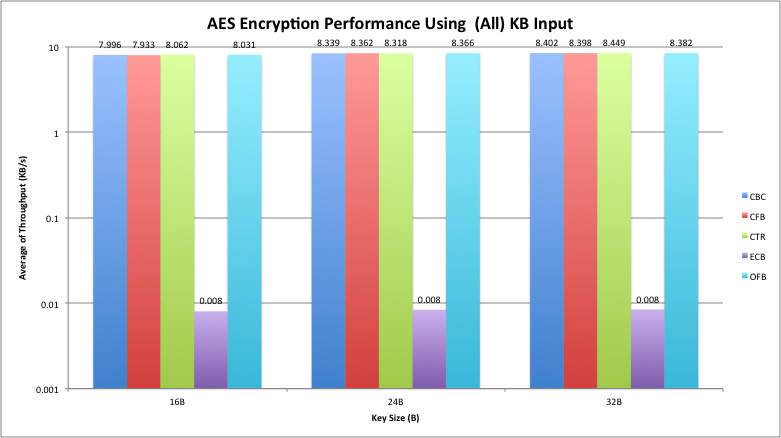
Decryption:



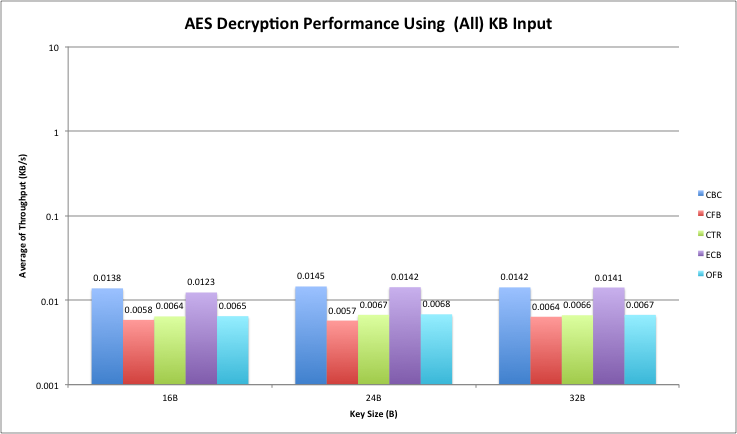
### Overall Effect of Key Size on Performance

Finally we look at the data across all test file sizes and compare the effect of different key sizes on the encryption and decryption throughput.

### Encryption



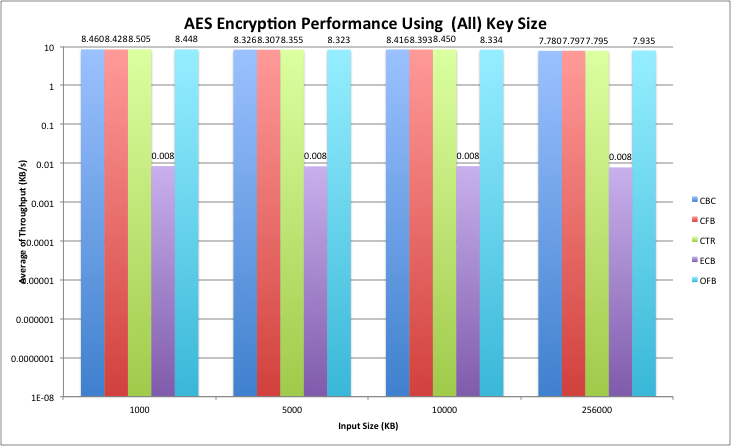
### Decryption



### Overall Effect of File Size on Performance

### Encryption

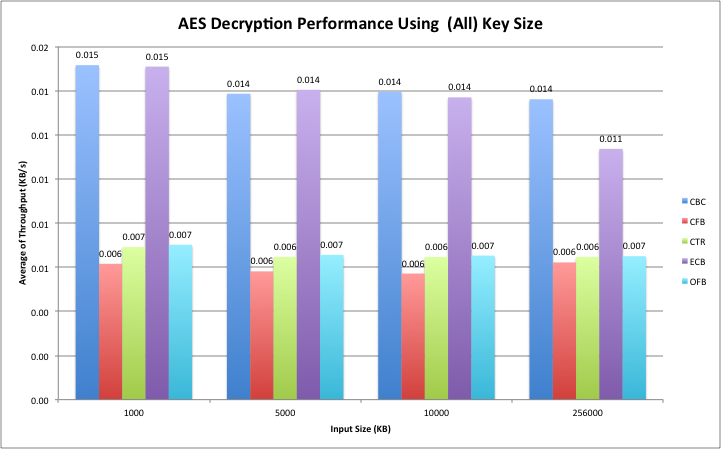
This section will show a comparison of the modes of operation with each other for encryption. A table will also show the min, max and average values for throughput for each mode.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Values |  |  |  |
| Row Labels | Min Time (ms) | Max Time (ms) | Average Time (ms) | Average of Throughput (KB/s) |
| CBC |  |  |  |  |
| 16B |  |  |  |  |
| 1000 | 0.113983 | 0.161035 | 0.11727494 | 8.526970894 |
| 5000 | 0.582458 | 0.70158 | 0.60771479 | 8.227543713 |
| 10000 | 1.15328 | 1.64988 | 1.1904 | 8.400537634 |
| 256000 | 33.159 | 41.7762 | 37.49458 | 6.82765349 |
| 24B |  |  |  |  |
| 1000 | 0.11464 | 0.153276 | 0.11941117 | 8.374425944 |
| 5000 | 0.581197 | 0.652863 | 0.59102666 | 8.459855263 |
| 10000 | 1.162 | 1.35043 | 1.1914591 | 8.393070312 |
| 256000 | 30.5214 | 33.3994 | 31.49103 | 8.129299042 |
| 32B |  |  |  |  |
| 1000 | 0.114906 | 0.15098 | 0.11793 | 8.479606546 |
| 5000 | 0.58796 | 0.772426 | 0.60317784 | 8.289429201 |
| 10000 | 1.15606 | 1.40618 | 1.1829124 | 8.453711365 |
| 256000 | 30.0476 | 31.2405 | 30.53484 | 8.383865774 |
| CFB |  |  |  |  |
| 16B |  |  |  |  |
| 1000 | 0.114568 | 0.158217 | 0.11785674 | 8.484877488 |
| 5000 | 0.587129 | 0.694268 | 0.60762368 | 8.228777391 |
| 10000 | 1.15983 | 2.14634 | 1.2027247 | 8.314454671 |
| 256000 | 33.6031 | 40.6218 | 38.19552 | 6.702356716 |
| 24B |  |  |  |  |
| 1000 | 0.115059 | 0.164382 | 0.12024837 | 8.316121042 |
| 5000 | 0.581399 | 0.674013 | 0.59286342 | 8.433645645 |
| 10000 | 1.16251 | 1.34371 | 1.1871028 | 8.423870283 |
| 256000 | 30.1804 | 33.5862 | 30.93884 | 8.274389085 |
| 32B |  |  |  |  |
| 1000 | 0.115645 | 0.150651 | 0.11790259 | 8.481577886 |
| 5000 | 0.591531 | 0.780781 | 0.60550594 | 8.257557308 |
| 10000 | 1.16086 | 1.38445 | 1.1848856 | 8.439633328 |
| 256000 | 30.1078 | 31.1549 | 30.42927 | 8.412952397 |
| CTR |  |  |  |  |
| 16B |  |  |  |  |
| 1000 | 0.114057 | 0.140622 | 0.11710759 | 8.539156173 |
| 5000 | 0.582324 | 0.712031 | 0.60431409 | 8.27384316 |
| 10000 | 1.15118 | 1.38016 | 1.1794972 | 8.478188842 |
| 256000 | 34.4609 | 42.0123 | 36.79368 | 6.957716651 |
| 24B |  |  |  |  |
| 1000 | 0.114585 | 0.158452 | 0.11899982 | 8.403374056 |
| 5000 | 0.579421 | 0.699999 | 0.59049072 | 8.46753358 |
| 10000 | 1.16288 | 1.31675 | 1.1878141 | 8.418825808 |
| 256000 | 30.3313 | 39.9751 | 32.06876 | 7.982846858 |
| 32B |  |  |  |  |
| 1000 | 0.114708 | 0.144878 | 0.11664321 | 8.573152265 |
| 5000 | 0.586555 | 0.78129 | 0.60078034 | 8.322509355 |
| 10000 | 1.15491 | 1.29677 | 1.1828337 | 8.454273834 |
| 256000 | 30.0205 | 30.8876 | 30.31387 | 8.444979147 |
| ECB |  |  |  |  |
| 16B |  |  |  |  |
| 1000 | 113.931 | 150.254 | 117.66337 | 0.008498822 |
| 5000 | 113.931 | 150.254 | 608.40352 | 0.00821823 |
| 10000 | 1149.2 | 1678.15 | 1182.0321 | 0.008460007 |
| 256000 | 32920.7 | 41554 | 37350.99 | 0.006853901 |
| 24B |  |  |  |  |
| 1000 | 114.81 | 147.434 | 119.3309 | 0.008380059 |
| 5000 | 580.424 | 677.193 | 591.50602 | 0.008452999 |
| 10000 | 1161.12 | 1454.54 | 1193.1301 | 0.008381316 |
| 256000 | 30269.3 | 31891.6 | 31066.19 | 0.00824047 |
| 32B |  |  |  |  |
| 1000 | 115.19 | 149.911 | 117.81823 | 0.008487651 |
| 5000 | 115.19 | 149.911 | 600.4612 | 0.008326933 |
| 10000 | 1153.4 | 1385.54 | 1185.3488 | 0.008436335 |
| 256000 | 29912.3 | 32149.7 | 30459.56 | 0.008404586 |
| OFB |  |  |  |  |
| 16B |  |  |  |  |
| 1000 | 0.114009 | 0.146226 | 0.11713888 | 8.536875203 |
| 5000 | 0.584783 | 0.708566 | 0.60642534 | 8.245038045 |
| 10000 | 1.16193 | 3.12236 | 1.2204305 | 8.193829964 |
| 256000 | 32.4285 | 37.516 | 35.80771 | 7.149298294 |
| 24B |  |  |  |  |
| 1000 | 0.115183 | 0.138544 | 0.11931408 | 8.381240504 |
| 5000 | 0.580627 | 0.661953 | 0.59110143 | 8.458785153 |
| 10000 | 1.16529 | 1.41491 | 1.1904867 | 8.399925845 |
| 256000 | 30.3939 | 32.5265 | 31.13221 | 8.22299477 |
| 32B |  |  |  |  |
| 1000 | 0.115715 | 0.149821 | 0.1187003 | 8.424578539 |
| 5000 | 0.590415 | 0.728783 | 0.60496234 | 8.264977288 |
| 10000 | 1.16763 | 1.29964 | 1.1894183 | 8.407471114 |
| 256000 | 30.0827 | 30.8844 | 30.35763 | 8.432805855 |

### Decryption

This section will show a comparison of the modes of operation with each other for decryption averaged across all key sizes. A table will also show the min, max and average values for throughput for each mode.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Values |  |  |  |
| Row Labels | Min Time (ms) | Max Time (ms) | Average Time (ms) | Average of Throughput (KB/s) |
| CBC |  |  |  |  |
| 16B |  |  |  |  |
| 1000 | 63.257 | 90.796 | 64.94904 | 0.015396686 |
| 5000 | 316.416 | 452.415 | 371.24275 | 0.013468276 |
| 10000 | 626.506 | 961.841 | 713.52114 | 0.014015002 |
| 256000 | 17765.2 | 24444.7 | 20887.55 | 0.012256105 |
| 24B |  |  |  |  |
| 1000 | 63.917 | 92.831 | 65.21395 | 0.015334142 |
| 5000 | 326.174 | 538.863 | 355.06464 | 0.014081943 |
| 10000 | 642.07 | 910.121 | 713.37619 | 0.014017849 |
| 256000 | 16964.2 | 18130.5 | 17549.99 | 0.014586903 |
| 32B |  |  |  |  |
| 1000 | 65.937 | 102.129 | 67.82167 | 0.01474455 |
| 5000 | 326.926 | 469.158 | 356.05536 | 0.01404276 |
| 10000 | 648.073 | 1083.34 | 722.88204 | 0.013833516 |
| 256000 | 17562.2 | 19662.7 | 18287.77 | 0.013998426 |
| CFB |  |  |  |  |
| 16B |  |  |  |  |
| 1000 | 144.74 | 228.22 | 151.53071 | 0.006599322 |
| 5000 | 939.601 | 1218.31 | 1027.24114 | 0.004867406 |
| 10000 | 1453.66 | 1872.27 | 1618.3942 | 0.006178964 |
| 256000 | 39332.1 | 51986.9 | 44826.65 | 0.005710889 |
| 24B |  |  |  |  |
| 1000 | 186.771 | 217.685 | 190.72769 | 0.005243077 |
| 5000 | 730.336 | 1150.85 | 795.59423 | 0.006284611 |
| 10000 | 1873.78 | 2434.65 | 2043.044 | 0.004894657 |
| 256000 | 38132.5 | 43966.7 | 39415.91 | 0.006494839 |
| 32B |  |  |  |  |
| 1000 | 147.586 | 184.193 | 150.87722 | 0.006627906 |
| 5000 | 735.66 | 1070.53 | 795.1363 | 0.00628823 |
| 10000 | 1483 | 2125.83 | 1647.4159 | 0.006070113 |
| 256000 | 38498.3 | 42343.2 | 39702.75 | 0.006447916 |
| CTR |  |  |  |  |
| 16B |  |  |  |  |
| 1000 | 144.336 | 173.049 | 147.03017 | 0.006801325 |
| 5000 | 716.917 | 957.846 | 784.01287 | 0.006377446 |
| 10000 | 1389.71 | 1834.94 | 1540.9601 | 0.006489461 |
| 256000 | 37510.4 | 51647.6 | 42821.19 | 0.005978349 |
| 24B |  |  |  |  |
| 1000 | 139.572 | 164.385 | 141.55201 | 0.007064541 |
| 5000 | 698.52 | 966.987 | 764.77649 | 0.006537858 |
| 10000 | 1396.6 | 1809.69 | 1538.193 | 0.006501135 |
| 256000 | 36331.8 | 41324.8 | 38251.66 | 0.00669252 |
| 32B |  |  |  |  |
| 1000 | 141.429 | 224.486 | 145.48395 | 0.00687361 |
| 5000 | 707.04 | 1093.83 | 770.82878 | 0.006486525 |
| 10000 | 1405.86 | 2162.38 | 1561.4413 | 0.006404339 |
| 256000 | 36630.1 | 39906.9 | 38000.57 | 0.006736741 |
| ECB |  |  |  |  |
| 16B |  |  |  |  |
| 1000 | 63.61 | 99.171 | 66.49903 | 0.015037813 |
| 5000 | 63.61 | 99.171 | 350.32823 | 0.01427233 |
| 10000 | 623.679 | 921.032 | 732.25806 | 0.013656388 |
| 256000 | 18018.8 | 212627 | 40691.43 | 0.006291251 |
| 24B |  |  |  |  |
| 1000 | 63.583 | 91.541 | 65.36764 | 0.015298089 |
| 5000 | 320.313 | 469.678 | 356.3565 | 0.014030893 |
| 10000 | 641.78 | 874.651 | 716.42056 | 0.013958282 |
| 256000 | 17908.7 | 21713.3 | 18800.35 | 0.013616768 |
| 32B |  |  |  |  |
| 1000 | 64.852 | 113.285 | 66.93177 | 0.014940588 |
| 5000 | 64.852 | 113.285 | 361.7554 | 0.013821494 |
| 10000 | 658.14 | 973.351 | 741.82154 | 0.013480331 |
| 256000 | 17056.7 | 19476.5 | 18046.44 | 0.014185623 |
| OFB |  |  |  |  |
| 16B |  |  |  |  |
| 1000 | 138.886 | 211.757 | 142.99573 | 0.006993216 |
| 5000 | 705.912 | 972.088 | 779.07748 | 0.006417847 |
| 10000 | 1363.03 | 1814.28 | 1549.4659 | 0.006453837 |
| 256000 | 37101.8 | 49596.6 | 42739.47 | 0.005989779 |
| 24B |  |  |  |  |
| 1000 | 138.877 | 175.601 | 142.01324 | 0.007041597 |
| 5000 | 687.136 | 1381.15 | 753.7661 | 0.006633357 |
| 10000 | 1376.31 | 1733.52 | 1502.4118 | 0.006655965 |
| 256000 | 35978.1 | 39519.4 | 37297.38 | 0.006863753 |
| 32B |  |  |  |  |
| 1000 | 139.316 | 174.239 | 142.38094 | 0.007023412 |
| 5000 | 694.511 | 1072.66 | 753.73852 | 0.0066336 |
| 10000 | 1395.64 | 2014.8 | 1544.6084 | 0.006474133 |
| 256000 | 36217.4 | 43328.2 | 38486.26 | 0.006651725 |

# Conclusions