Studying the Performance of Ishmael

By: John Shultz

Contents

[Abstract 3](#_Toc52294726)

[Introduction 3](#_Toc52294727)

[Methodology 3](#_Toc52294728)

[Results 6](#_Toc52294729)

[Discussion 7](#_Toc52294730)

[Conclusion 10](#_Toc52294731)

# Abstract

The goal of this report was to determine how using different wordlists would impact the performance of Ishmael. It involved encrypting five test files several times, using different wordlists either intentionally constructed or downloaded as mixed text to test their performance. Two factors of performance were measured, the overall speed of encryption and the size of the resulting file. The fastest and most storage efficient list amongst the ones tested was a list using many unique words each under five characters in length. In addition, some potential changes to program’s control flow could further these improvements.

# Introduction

The goal of this study is to analyze the overall performance of the Ishmael encryption scheme that I have created. Ishmael is an encryption scheme that uses large wordlists as the key. These wordlists can commonly take the form of books, but any significantly large list of words would be possible to use. This is done first by encoding a file into base64 text, and then creating a translation table in memory that maps multiple words in the list to each character in the base64 text. I intend to analyze how constructing this word list can affect both the size of the encrypted files, as well as the speed of the encryption process.

# Methodology

To determine whether the contents of the file being encrypted has any bearing on how long the encryption process takes, I will be encrypting an example of a few common file types. These files will be a .jpg, a .mp3, a .mp4, a .txt, and a .pdf file. I have selected this arrangement of formats because they are all commonly found on an average users system, could contain information that would be sensitive enough to encrypt, and together represent a collection of several different types of file (image, audio, video, raw text, and a formatted document). These files will each be encrypted five times per each wordlist, with the time it takes to encrypt in seconds and the resulting size of the encrypted file in kilobytes noted each time. To perform this process, I will start the program, attempt to encrypt the file five times using the given wordlist without restarting the program, and close the program before moving on to the next wordlist. The total amount of time to encrypt and decrypt each file will focus solely on the time it takes to generate the wordlist and to encrypt the file, ignoring any time spent by the user writing data or the program writing files to disk as these factors will be highly variable. The results from these tests will then be averaged and compared against each other. I will provide a copy of all the wordlists that I will be using in this study, as well as providing a summary of each wordlist below in figure one.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Wordlist Name | Total Number of Words | Total Number of Unique Words | Description of List | Purpose of Wordlist |
| mobydick.txt | 208,458 | 20,050 | The full text of Moby Dick from Project Gutenberg | Control. This was the first wordlist used with Ishmael. |
| aliceinwonderland.txt | 26,440 | 2,832 | The full text of Alice’s Adventures in Wonderland from Project Gutenberg | Experiment to test the performance of a smaller book against the control |
| warandpeace.txt | 562,487 | 20,821 | The full text of War and Peace from Project Gutenberg | Experiment to test the performance of a larger book against the control |
| common.txt | 10,000 | 10,000 | The 10,000 most common words in order of frequency as determined by n-gram frequency analysis of the Google’s Trillion Word Corpus, performed by Josh Kaufman. | Experiment to test the performance of an averagely sized list of only unique words |
| maxwords.txt | 466,550 | 466,550 | A list of 466,550 words assembled by github user dwyl. | Experiment to test the performance of an extraordinarily large list of words. |
| minwords.txt | 65 | 65 | 65 words randomly selected from the previous maxwords.txt list. | Experiment to test the performance of the smallest possible list of words |
| smallwords.txt | 16,600 | 16,600 | All words from the previous maxwords list containing fewer than four characters. | Experiment to test the performance of a list made entirely of small words. |

*Fig. 1. Descriptions of all wordlists used.*

# Results

To review all the data collected as a part of this test, please view the attached spreadsheet “Analysis Data.xlsx”. The excel workbook contains five sheets, one for each file type being tested. Each sheet documents all five trials run for each of the word lists, including the overall time it took the program to run, the final encrypted filesize, and by how much the filesize increased expressed as a percentage. Averages are also provided.

# Discussion

The results from my testing have proved to be interesting. Firstly, as can be seen in figure 2, for the most part the percentage of file size inflation post encryption remained consistent for most file types. This is to be expected, as we are simply encoding each character in a base64 representation of the file. However, for all wordlists there was less inflation for the encoding of .txt files. This may be because textual data most easily translates into shorter base64 strings, or simply that since the text file was significantly smaller than the others there was less to encrypt. While for larger files there may be a few hundred bytes of difference between attempts, overall, the final size of the file could be predicted, and displayed to the user before the encryption process begins. Another conclusion that can be drawn from figure 2 is the impact of wordlist construction on overall final encrypted file size. By far the best performing words list in this area was “smallwords.txt”, which was constructed to have no words larger than 4 characters. For this reason, the final resulting file size stays rather small. This shows us that if we are optimizing for size a good wordlist is one constructed as many small words as possible. A word list beginning with all combinations of single characters, then all combinations of two characters, then all combinations of three characters, etc. would give the best performance. Even with these constraints, it is still likely that the resulting encrypted file would be multiple times larger than the original.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Wordlist | .jpg | .mp3 | .mp4 | .txt | .pdf |
| MOBYDICK.TXT | 1066% | 1069% | 1068% | 1041% | 1066% |
| ALICEINWONDERLAND.TXT | 880% | 885% | 881% | 868% | 884% |
| WARANDPEACE.TXT | 1073% | 1075% | 1072% | 1045% | 1074% |
| COMMON.TXT | 908% | 911% | 910% | 889% | 909% |
| MAXWORDS.TXT | 1281% | 1281% | 1292% | 1255% | 1278% |
| MINWORDS.TXT | 1257% | 1265% | 1285% | 1231% | 1261% |
| SMALLWORDS.TXT | 517% | 518% | 518% | 511% | 517% |

*Fig 2. Average File Size Inflation per Filetype for all Wordlists*

Figure 2 also demonstrates that the file sizes do not increase exponentially as the files get larger. If this were the case, we would expect the overall inflation rate to be much higher for the .mp4 and .mp3 files than for the .jpg and .pdf files which were significantly smaller. This is not observed for any of our wordlists, with only slight increases in overall inflation rate observed in two of the wordlists.

Chart, line chart

Description automatically generated

*Fig. 3. A line chart plotting total encryption time in seconds for each test file for all wordlists*

Chart, line chart

Description automatically generated  
*Fig. 4. A line chart plotting the file size of the original files and the average encrypted size for all wordlists*

Another interesting point of discussion can be found in the comparison of figure 3 and figure 4. These graphs plot file sizes in kilobytes against the compute time in seconds for each of the wordlists, with the file sizes of the original files included for reference. As is unsurprising the shape of these graphs is similar. As the size of the file to be encrypted increases, so too does the time it takes to encrypt it. However, these are not the only factors that appear to affect average time to encrypt. As can be shown in Figure 5, there appears to be a correlation between having a low number of unique words and having a low compute time. While this is a factor, the words within the lists do themselves play a factor as well, as smallwords.txt consistently outperformed the other lists across the board. This suggests that crafting a list that will compute efficiently is a matter of balancing the number of unique words in the list against the overall size of each word within it.

  
*Fig. 5. Each wordlist sorted by average compute time in ascending order*

Finally, it should be noted that when encrypting test.mp4, the largest file in my sample three wordlists had memory issues occur. The lists in question were mobydick.txt, warandpeace.txt, and maxwords.txt. This issue is likely due to the current programs structure. Each time a user wishes to encrypt a file, they are required to provide the wordlist again, and the entire process of creating the translation table is repeated. While I had assumed that the old tables would be released from memory, if they are not for these large tables this would quickly fill the memory space. To fix this issue, I am going to experiment with changing the structure of the program as it currently stands. In the new format the user will be prompted to set the wordlist at program start. Then, they may encrypt or decrypt as many files as they wish, with the current translation tables being retained in memory. A separate menu option will allow the user to switch between tables to reduce the amount of memory wasted.

# Conclusion

In conclusion, this study of the Ishmael process has revealed a few important factors about the program. Firstly, the amount of file inflation is large, but can be controlled through careful wordlist construction. Wordlists constructed to control against file size inflation also perform the faster than other lists. This analysis has also uncovered a flaw in the current structure of the program that leads to reaching the limits of python’s memory for certain wordlists after only a few encryption attempts, something that can be corrected for with new program flow. This new program flow should also offer benefits for overall encryption and decryption speed. Once these changes are made, I will do a comparative analysis between the old version of Ishmael and the new.