THESIS

ENTROPY-BASED FILE TYPE IDENTIFICATION AND PARTITIONING

by

Calvin B. Paul

June 2017

Thesis Advisor: Roberto Cristi
Co-Advisor: Monique P. Fargues

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 2017	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE ENTROPY-BASED FILE TYPE IDENTIFICATION AND PARTITIONING	**5. FUNDING NUMBERS**
6. AUTHOR(S) Calvin B. Paul	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The need for file identification and partitioning in the digital forensic, reverse engineering, and security analyst fields cannot be overstated. In this research, we investigate the use of the Shannon entropy profile derived from the file expressed in byte format to characterize specific file types and identify file segments based on entropy-level changes. The process consists of two stages. In the first stage, a binary representation of the file is partitioned into chunks of fixed-length data bytes and processed to extract the entropy profile. In the second stage, the detrended fluctuation analysis (DFA) method is applied to determine the level of structure in the entropy profile. The Haar continuous wavelet transform (CWT) is then used to partition the files identified as highly structured into areas of distinct changes in entropy level. Experimental results show that the proposed approach is effective in identifying file types and partitioning in segments of different entropy levels.

14. SUBJECT TERMS file type identification, file partitioning, entropy, feature vector, detrended fluctuation analysis, Haar continuous wavelet, statistical measure	**15. NUMBER OF PAGES** 107
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

ENTROPY-BASED FILE TYPE IDENTIFICATION AND PARTITIONING

Calvin B. Paul
Civilian, Armaments Corporation of South Africa (Armscor)
B.Tech., Durban University of Technology, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2017

Approved by:	Roberto Cristi Thesis Advisor
	Monique P. Fargues Co-Advisor
	R. Clark Robertson Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The need for file identification and partitioning in the digital forensic, reverse engineering, and security analyst fields cannot be overstated. In this research, we investigate the use of the Shannon entropy profile derived from the file expressed in byte format to characterize specific file types and identify file segments based on entropy-level changes. The process consists of two stages. In the first stage, a binary representation of the file is partitioned into chunks of fixed-length data bytes and processed to extract the entropy profile. In the second stage, the detrended fluctuation analysis (DFA) method is applied to determine the level of structure in the entropy profile. The Haar continuous wavelet transform (CWT) is then used to partition the files identified as highly structured into areas of distinct changes in entropy level. Experimental results show that the proposed approach is effective in identifying file types and partitioning in segments of different entropy levels.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AES	Advanced Encryption Standard
ANN	Artificial Neural Network
ASCII	American Standard Code for Information Interchange
CWT	Continuous Wavelet Transform
DFA	Detrended Fluctuation Analysis
DWT	Discrete Wavelet Transform
ELF	Executable and Linkable Format
k-NN	k-Nearest Neighbor
MB	Megabyte
OEP	Original Entry Point
PE	Portable Executable
RMSE	Root Mean Square Error
STFT	Short-Time Fourier Transform
SSECS	Suspiciously Structured Entropic Change Score
UPX	Ultimate Packer for Executables

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisors for their direction and continued support. I would like to thank my organization, Armscor, for affording me the opportunity to pursue my master’s. Last but not least, I would like to acknowledge my wife, Trisha, for her unwavering support and patience.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION AND LITERATURE REVIEW

In this chapter we provide the motivations behind the recent developments in the large body of research conducted in file type identification and partitioning, summarize current literature in this field, and outline the remaining structure of the thesis.

A. INTRODUCTION

The recent significant advancements in man-machine interaction have had the undesired consequences of increased exposure across a multitude of disciplines to cyber-attacks from maliciously obfuscated files. The number of malicious attacks has risen in recent years due to hackers circumventing current defense mechanism by employing code obfuscation techniques [1] such as encryption, polymorphism and metamorphism, rendering the obfuscated code immune to standard signature detection [2]. Polymorphic and metamorphic malware are capable of changing their internal structure without altering their malicious behavior. Although dedicated tools exist to study frequently occurring file types and system specific file formats, the study of suspicious binary objects is limited to the most elementary tools [3]. From the security analyst defending against attacks by malicious malware to the digital forensic analyst tasked with carving segmented files, the need for file type identification and partitioning is of paramount importance.

In this thesis we explore the application of statistical measures as a tool to aid in the identification of file types and file partitioning. This approach has applications in cybersecurity as it allows for a quick determination of compressed and encrypted file types which may possibly be malicious. Furthermore, this technique can also be of interest in forensic analysis of corrupted files, also called file carving.

The rest of the thesis is organized as follows. In the rest of Chapter I, current literature dealing with statistical measures used for file type identification and partitioning is reviewed. The proposed methodology and the techniques adopted are concisely defined in Chapter II. The experiment setup for file analysis and identification detailing how the data set is generated, feature vector extracted, and file segmented is covered in Chapter III. Results and analysis are reported in Chapter IV, followed with concluding remarks and recommendations for future work in Chapter V.

B. LITERATURE REVIEW

A significant body of work to classify file types using statistical measures including entropy has been reported in the literature. Using entropy as a measure of randomness and unpredictability in a series of data values or event sequence is a standard technique and appears in a number of related works [4]. Static analysis is used to determine those file characteristics that are necessary for their classification without having to execute the file, which could possibly be malicious malware within a protected environment.

Orthogonal methods to analyze malware by using signal and image processing techniques were explored by [1] where malware samples are represented as images or signals. Image and signal based features were then extracted to characterize the malware. The efficacy of the methods on malware classification, detection, and retrieval was demonstrated and extended to the data forensics and data type classification field.

The common trend in the case studies below is the application of file entropy profile as a measure for file type identification.

Donabelle et al. [2] applied previous work by Sorokin [8] on structural entropy to the metamorphic detection problem of classifying whether a given file belongs to a specific metamorphic family. The results obtained indicated that similarity measures based on structural entropy can be implemented as a tool to classify potentially metamorphic malware.

Conti et al. [3] used 1,000 segments, each of byte length equal to 1,024, from 14 commonly encountered primitive files to develop statistical signatures for properly classifying segments into various types. Four statistical measurements, namely Shannon entropy, Hamming weight, Chi-squared value obtained from the Chi-Square Goodness of Fit Test, and the mean byte value, were selected as components of a feature vector to characterize each segment. The actual classification stage was implemented with a k-nearest neighbor (k-NN) algorithm, and classification results were high for segments of known primitive types but did not generalize well.

Bintropy, a binary entropy analysis tool to discriminate between native executables and those with packed or encrypted formats, was developed and used in [4]. This approach relied on the fact that encrypted and compressed files tend to have high entropy values. Bintropy processes files by iterating through fixed-length binary data blocks of 256 bytes and generating the entropy profile. Next, the average and highest entropy values are derived. Finally, confidence intervals of these quantities are computed and used as basis for file type identification.

Jochheim [5] used signal processing techniques to automatically detect malicious binary code possibly embedded within regular data. The Shannon entropy profile was also used to extract changes in the file structure. The Short-Time Fourier transform (STFT) was applied to the entropy profile to generate a set of power related parameters used as inputs to an artificial neural network (ANN) classifier. Results show that the proposed method is able to detect a variety of shellcode attacks with low system overload notwithstanding the following limitations; the test-data file size is limited to 1MB per file-type and the classifier results in a number of false detections, which limits its usage.

Fitzgerald et al. [6] explored the application of supervised machine learning techniques commonly applied in natural language processing to file segment classification. A large data set of file segments from 24 different file types was used for the analysis. A feature vector consisting of unigram and bigram counts of bytes in each segment and other statistical parameters was used to represent each file segment, and a support vector machine used as classifier. Results showed a wide variation in classification rates for the file types considered in the study, ranging from 99.7% (for CSV types) to 2.3% (for pptx types), with better rates obtained for segments with low entropy levels, resulting in an overall average rate equal to 48%. The authors claimed their results were better than those obtained with studies based on a similar wide range of file types.

Jeong et al. [7] proposed a generic unpacking mechanism using entropy analysis to find the original entry point (OEP) of packed executables. They experimented with 110 packed executables and demonstrated that the proposed mechanism can locate the OEPs of 72% of the packed executables and can also be applied to packed malware; however, the approach required unpacking of the packed executable in order to determine the OEP.

In the approach adopted by [8], the entropy profile of the file structure was first computed using a sliding window approach; thereafter, file segmentation was carried out by applying the Haar discrete wavelet transform (DWT) to the computed file entropy profile. The segments of the entropy profile were then compared to segments with known malware file types using the Levenshtein distance method based on the assumption that they had expected standard characteristics. The method achieved a degree of similarity between two sequences of 87.56% and has a number of malware detection applications; however, the algorithm resulted in false alarms. As a result, the solution was only useful as a preliminary trigger to run other tests.

The Haar DWT decomposition of structured entropy was used by [9], a commercial cybersecurity company, to reveal potential malware using the Haar DWT computed from the file entropy profile. In their study, about 40,000 portable executable (PE) files were studied with 50% containing malware. A single scalar feature denoted as the Suspiciously Structured Entropic Change Score (SSECS) was defined to quantify files as malicious or not, and malware prediction accuracy results were shown to be equal to 68.7%.

In this section, we described recent work conducted in the computer file segment type identification field and illustrated the use of entropy as a potential feature in several of these approaches. The approach used in our work is presented in Chapter II.

II. METHODOLOGY AND PROPOSED TECHNIQUES

The method adopted and techniques used for collecting and analyzing the data needed for file type identification and partitioning are described in this chapter.

A. METHODOLOGY

This study has two main goals; first, to identify the specific file types for the files investigated and, second, to partition files identified as highly structured (i.e., with highly variable entropy profiles) into segments based on entropy changes. Three statistical parameters are considered in this study: Shannon entropy, Hamming weight, and Arithmetic mean. Files are split into chunks and statistical parameters computed in each chunks resulting in entropy profiles, Hamming weight profiles, and arithmetic mean profiles, respectively. This study is restricted to the four following file types: plain text, native Portable Executable (PE), packed native PE, and encrypted native PE files. A detailed description of the process shown in Figure 1 is provided below with the techniques adopted provided thereafter.

First, each file is converted into a hexadecimal representation compatible with the Matlab platform, as analysis takes place in the Matlab environment using scripts included in Appendix A. Second, each file is converted into a binary vector and locally analyzed using a sliding window with a predefined window length equal to 256 and no overlap to segment the file. Third, the statistical measures considered in the study are applied to each file segment to extract statistical parameters, and a segment type decision is made from these values. In addition, a decision on the overall file type can be made by comparing the average entropy and the highest entropy parameters to the values derived from the file type dataset.

Our simulations showed that native PE files and packed files included in our dataset exhibited distinct transition levels in their entropy profiles, while plain text and encrypted files did not.

Fourth, the Detrended Fluctuation Analysis (DFA) is selected as a tool to automate the decision regarding the presence of distinct transitions in the entropy profile.

Finally, for files identified as highly structured, the Haar CWT is applied to identify the specific location of the entropy profile transitions.

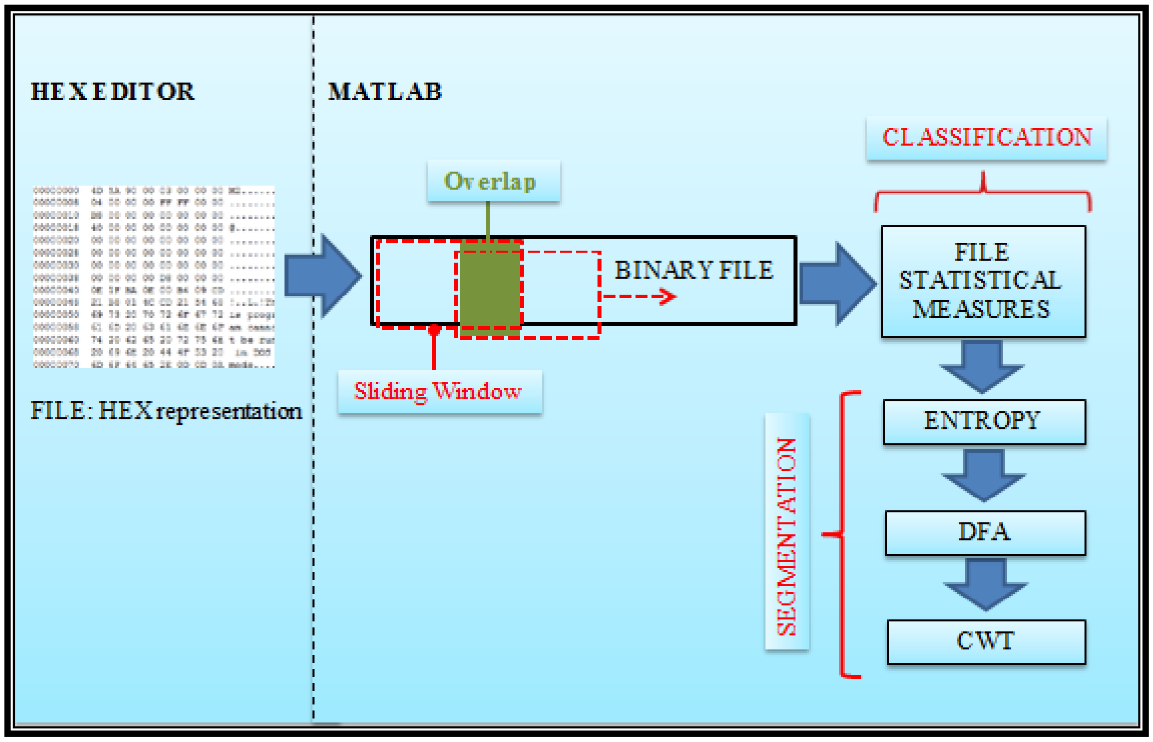

Figure 1. Process Adopted for File Type Identification and Partitioning

In the rest of the chapter, we provide an overview of the techniques adopted for file type identification and partitioning operations.

B. STATISTICAL TECHNIQUES

Three particular statistical measures were adopted in the algorithm: Shannon entropy, Hamming weight, and Arithmetic mean.

Although these measures are in line with current research work [3], our investigations showed that only the entropy is suitable for identification for the dataset selected in our study; however, we present all for completeness.

1. Shannon Entropy

In information theory, Shannon entropy is an established technique for measuring the amount of randomness or disorder, i.e., information contained in a given segment. It yields the amount of "information" and "randomness" in the given data in terms of number of bits per sample. Its root is in combinatorics and is related to the total number of realizations of data sequences associated to the given statistics.

The Shannon entropy ($H(X)$) is computed as [5]

$$H(X) = -\sum_{i=1}^{n} p(X_i)\log_2 p(X_i) \ , \qquad (1)$$

where X is a vector of data where each symbol belongs to a finite alphabet composed of "n" symbols. In the case of byte-level entropy analysis, as applied in this thesis, each entry of X belongs to an alphabet composed of 256 symbols, corresponding to an 8-bits/symbol encoding. The alphabet size n is of the form $n = 2^b$ where b is the number of bits (b=8 bits in a byte). The probability mass function $p(X_i)$ is the probability of occurrence of byte value i in the segment. Byte-level entropy analysis yields entropy values ranging from 0 to $\log_2 2^8 = 8$.

As an example, using an alphabet of only two characters, {0;1}, results in an entropy range between 0 and $\log_2 2 = 1$. If X = [0, 1, 1, 1], then $p(X_1) = p(0) = 0.25$ and $p(X_2) = p(1) = 0.75$ which yields

$$H(X) = -[(0.25 \times \log_2 0.25) + (0.75 \times \log_2 0.75)] = 0.8113.$$

Entropy analysis offers a convenient and quick method for analyzing a file at the binary level as a possible preprocessing step to identify suspicious file regions. As we see later in the work, files containing binary code, compressed data or text can be distinguished fairly reliably by the entropy measure. As a consequence, file regions identified as suspicious can be disassembled and further analyzed with reverse-engineering disassembling tools such as OllyDbg and IDAPro [4].

2. Hamming Weight

To determine the Hamming weight, each byte of the given segment is converted into an 8-bit binary representation, i.e., a binary alphabet of zeros and ones. The Hamming weight is the ratio of the number of ones to the total number of bits in the given segment. As an example, the Hamming weight for an 8-bit binary number X = [10 11 00 10] is computed as

$$HW(X) = \frac{Number\ of\ ones}{Total\ number\ of\ bits} = \frac{4}{8} = 0.5.$$

3. Arithmetic Mean

The arithmetic mean is defined as being equal to the sum of the byte values in the given segment divided by the segment size

$$\mu = \frac{1}{n}\sum_{i=1}^{n} X_i \quad , \tag{2}$$

where X_i and n are the byte values (in decimal format) and the segment size, respectively. As an example, the arithmetic mean for X = [100, 120, 256, 200] is computed as

$$\mu = \frac{100+120+256+200}{4} = \frac{676}{4} = 169.$$

C. DETRENDED FLUCTUATION ANALYSIS

The DFA algorithm may be applied to determine the level of structure in a time series. The algorithm, introduced in [10], determines the potential level of stationarity of a time series by analyzing the integral (or running sum) of the detrended sequence.

Following the analysis in [10], we can use a specific DFA statistic scaling factor (α) to assess the stationarity of the data file. In particular, the larger the value of α the more structured (non-stationary) the entropy profile tends to be [10]. Time series with a value of α exceeding a given threshold can be further partitioned into regions of different entropy values. The larger the value of α the more viable the file is to partitioning using the CWT. The DFA process is described in the following steps [11] with the aid of Figure 2.

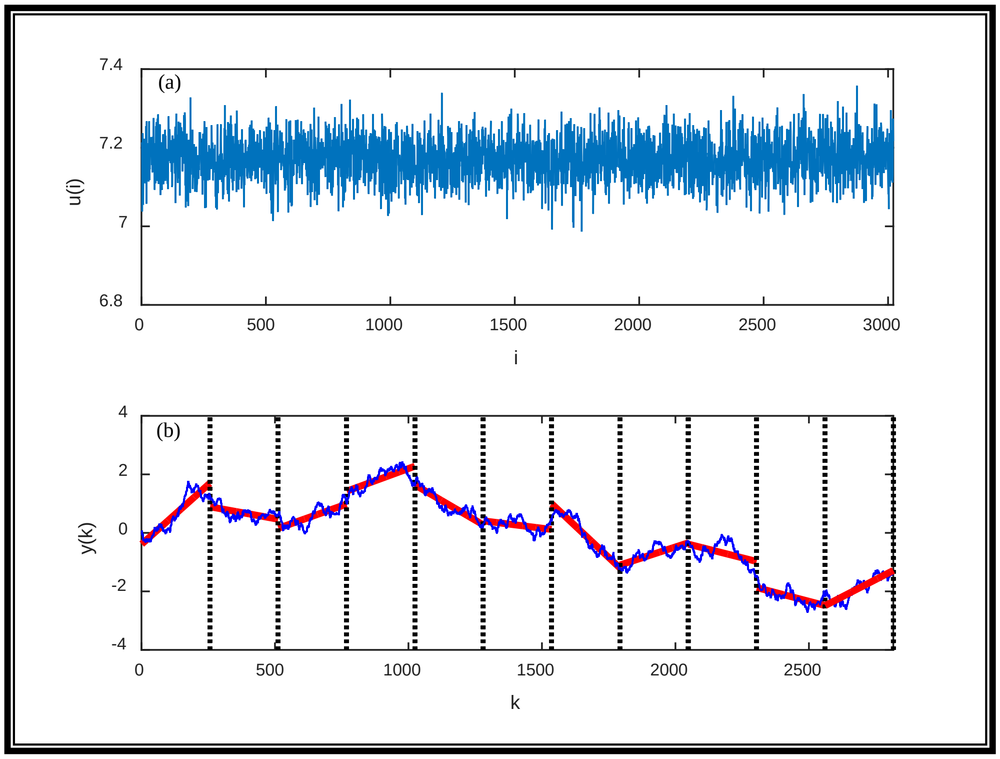

Figure 2. Depiction of Detrended Fluctuation Analysis. Adapted from [11].

1. Step 1

The entropy profile $u(i)$, illustrated in Figure 2 subplot (a), where $i = 1, \ldots,$ N, and N is the length of the entropy profile in bytes, is numerically integrated by computing the running sum of the detrended times series as

$$y(k) = \sum_{i=1}^{k} \left[u(i) - \langle u \rangle \right] , \tag{3}$$

where $\langle u \rangle$ is the mean of the entropy profile [11].

2. Step 2

The resulting sequence *y* illustrated in Figure 2 subplot (b) is divided into chunks of window length equal to *n*. A window length of 256 is used in this thesis, illustrated by the dotted vertical lines in Figure 2 subplot (b). Next, in each window we fit *y* with an ordinary least-square regression line y_n that is representative of the trend in the particular window, illustrated by the solid red lines in Figure 2 subplot (b).

3. Step 3

The sequence y is detrended by subtracting the local trend y_n in each window, and the root-mean square error fluctuation within each window is calculated using the standard formula,

$$RMSE(n) = \sqrt{\frac{1}{N}\left(\sum_{k=1}^{N}\left[y(k) - y_n(k)\right]^2\right)}. \tag{4}$$

4. Step 4

The DFA statistic scaling factor (referred to asα) is calculated by fitting a linear regression line to the sequence $\log\left[RMSE(n)\right]$ as a function of log *n*. The slope of the fitted linear regression line is the DFA statistic scaling factor α [10].

The decision to segment the entropy profile using the CWT is based on the size ofα. The following criteria were applied [10]:

- $\alpha > 1 \Leftrightarrow$Non-Stationarity present, therefore, viable to partitioning using the CWT,
- $\alpha \approx 0.5 \Leftrightarrow$Stationary entropy times series (white noise), not viable to partitioning using the CWT.

If a time series is assessed as non-stationary, then it can be partitioned by a number of techniques. As computational complexity was not an issue in this thesis research and we were experimenting with digital signal processing techniques, we selected the CWT in this study for its effectiveness; thus, the CWT is used to partition the files determined by the DFA analysis as non-stationary.

D. WAVELET ANALYSIS

Fourier analysis, as described in [12], decomposes a signal into its constituent sinusoids of different frequencies, transforming the view of the signal from a time domain to a frequency domain. A drawback of the Fourier analysis is that time information (determining when a particular event took place) is lost during the transformation, thereby making the Fourier analysis unsuited to detecting important signal characteristics such as the beginning and end of events, drifts, abrupt changes, and trends.

The Short-Time Fourier-Transform (STFT) addresses the above drawback of Fourier analysis by computing the frequency spectrum on a sliding window, resulting in a time-frequency spectrum so that events can be localized in both time and frequency [12]. The drawback of the STFT is what is called the “uncertainty principle,” for which an event can be accurately localized in either time of frequency but not both.

Wavelet analysis overcomes the fixed window drawback of the STFT by using windows of variable length according to the frequency content: longer windows for lower frequencies and shorter windows for higher frequencies. Since sudden transitions are associated to higher frequency components, a shorter window provides an accurate localization of the transition [12]. From Figure 3, the time resolution (horizontal axis) for different values of the “scaling factor” defined as the reciprocal of frequency (vertical axis) can be seen. Essentially, per [12], wavelet analysis consists of breaking up the signal being analyzed into shifted and scaled versions of the original (mother) wavelet.

In this chapter we proposed an approach to identify the file type and partition the file based on Shannon entropy, Hamming weight and Arithmetic mean measures. In the next chapter the proposed techniques are applied to the identified file types to determine the effectiveness of the proposed methods.

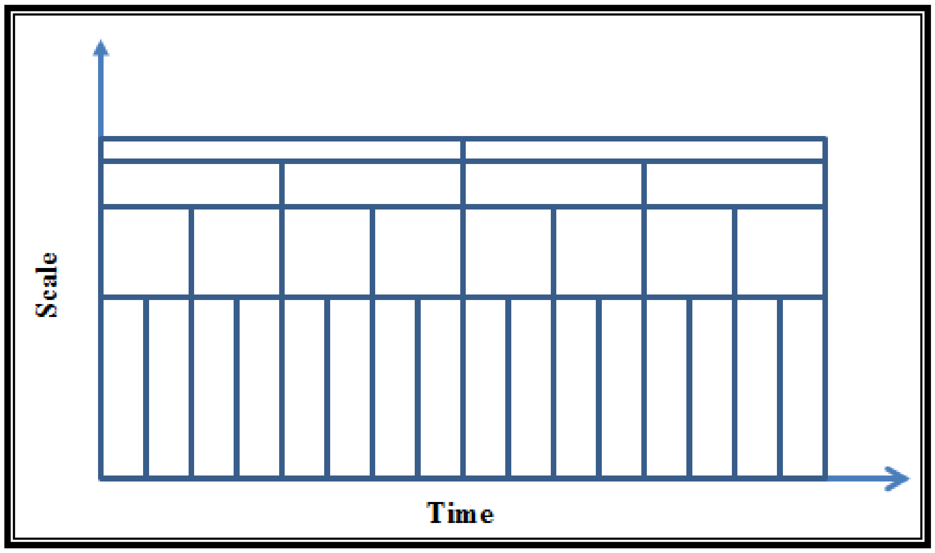

Figure 3. Wavelet Analysis. Adapted from [12].

III. FILE ANALYSIS AND IDENTIFICATION

In the previous chapter we described our methodology and proposed techniques to classify and partition files. In this chapter we apply the proposed techniques to the data sets and assess the effectiveness of the proposed methods to file type identification and partitioning. The experiment consists of the processes presented in Figure 4 with the details presented in the sections that follow.

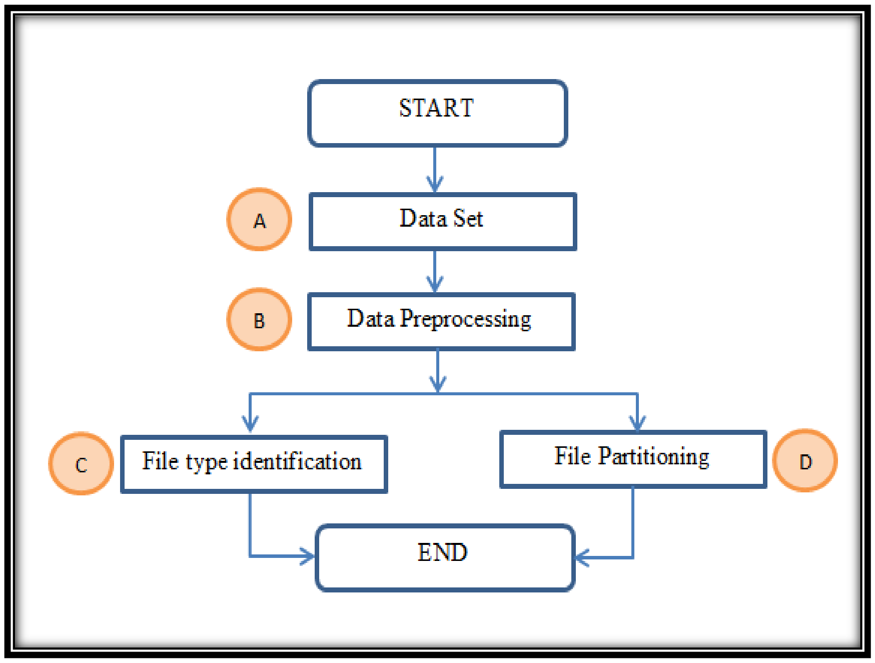

Figure 4. Flow Chart of the Experimental Process

A. DATA SET

The data set considered for our analysis presented in Table 1 was generated from multiple diverse sources. We acquired files of different types with varying statistical measure values to validate a general claim found in the literature that different file types have different entropy properties. It is well documented that higher entropy levels tend to correlate with the presence of encryption, compression, random data and binary executables [3]. Furthermore, segments with medium entropy levels that exhibit a noticeable file structure are indicative of machine code and human languages, whereas low entropy levels are indicative of segments with redundant information, such as plain-text files, uncompressed media, etc. [3].

Table 1. Data Set Properties

File Type	Number of files	Number of file chunks	Source
Plain Text	32	214322	*https:www.gutenberg.org/*
Native Portable Executables (PE)	84	207648	*Windows System Files (alphabetically chosen)*
• *32-bit PE*	*42*	*88858*	
• *64-bit PE*	*42*	*118790*	
Packed PE	84	119035	*Native PE files packed using UPX Packer(http://upx.sourceforge.net)*
• *32-bit Packed PE*	*42*	*57898*	
• *64-bit Packed PE*	*42*	*61137*	
256-bit Encrypted PE	84	253485	*Native PE files encrypted using AES (https://www.aescrypt.com/)*
• *32-bit Encrypted PE*	*42*	*106535*	
• *64-bit Encrypted PE*	*42*	*146950*	

Plain text files included text-only books encoded in US-ASCII and were obtained from https:www.gutenberg.org/.

Portable Executables (PE) file format used by Microsoft Operating Systems is a format for executables that are not architecture specific and is highly popular amongst computer users and source of security breaches [13]. The native PE files were alphabetically selected from the system folders on a Windows 7 Professional Service Pack 1, 32-bit and 64-bit operating system environment.

Packing is a method used to obfuscate an executable file by encrypting or compressing it to protect its original content from reverse engineering. A packed executable is in essence an executable file packed inside another executable file [14] and is a technique used for malicious intent by malware authors to hide a malicious payload inside an executable file, thus avoiding detection by malware detectors. When executed, the outer executable unpacks the contents of the inner executable which could be malicious code into memory and executes it [14]. As the malicious payload is only unpacked at runtime and physically exists only in memory, the intent of this thesis is not to unpack and analyze the packed PE files but rather to identify segments of files that contain packed code which could potentially be malicious. The native PE files included in this study were packed using the UPX packer, which is available online at http://upx.sourceforge.net.

The Advanced Encryption Standard (AES) symmetric cipher with a 256-bit key length was used to encrypt the native PE files using encryption software that is freely available at https://www.aescrypt.com/. The encryption performs various substitutions and transformations on the PE file creating a cipher text, which is an unintelligible random stream of data [15].

B. DATA PREPROCESSING

The goal of the data preprocessing stage is to convert the files in the data sets into a file format compatible with Matlab. A free version of a “hexadecimal” (or hex) editor was used towards that goal where the inputs to the hex editor application are the individual data files (each processed separately), and the output is a hexadecimal representation of the input file.

We applied the statistical measures described in Chapter II, paragraph B using Matlab to each of the data sets to generate individual feature vectors for file type identification. Data located at the end of the binary file that did not constitute a complete segment were ignored.

C. FILE TYPE IDENTIFICATION

Files of the same type exhibit similar statistical measures and, therefore, have similar feature vectors, whereas dissimilar files types exhibit different statistical measures and have different feature vectors. The difference in feature vectors is the basis for file type identification. The feature vector identification process is extended to include a confidence interval (CI) for the aggregated average and highest average statistical measures for each data set type. The CI is a measure of the uncertainty in the statistical data given the randomness of the data and provides a range in which to allow the statistical measures to vary. Given that both the mean and variance of the data set is estimated, the t-distribution is normally the applicable distribution when calculating the CI bounds; however for data sample size N > 30, the normal distribution is used as it approximates the t-distribution [16]. The normal distribution with various CI levels was tested with a final CI of 95% determined as optimal since using CIs larger than 95% results in range overlap in statistical measures between different file types and as such is not useful as a discriminant. A 95% CI was applied to the aggregated average and highest average measures to generate a feature vector for file type identification.

The upper and lower CI limits were calculated using [16]

$$\hat{m}_x - \frac{z_{\alpha/2}\,\hat{\sigma}_x}{\sqrt{N}} \le m_x \le \hat{m}_x + \frac{z_{\alpha/2}\,\hat{\sigma}_x}{\sqrt{N}}, \tag{5}$$

where $\hat{m}_x$ is the estimated mean, $\hat{\sigma}_x$ is the estimated standard deviation, and α is the user specified level of significance used to quantify the CI as: $CI = 1 - \alpha$.

D. FILE PARTITIONING

Any file can be characterized by its structure, and distinctive code and data areas can be identified when they have different entropy levels; thus, detecting the significant changes in the file entropy profile is the key to the file partitioning conducted in our work. That detection step is performed post determination by the DFA analysis that the file can be partitioned using the CWT. Evaluation of various wavelet basic functions revealed that the Haar CWT is best suited for file partitioning, as illustrated by its superiority over the Morlet CWT in partitioning the entropy profile in Figure 5 subplot (a) into regions of significant changes in entropy level; refer to Figure 5 subplot (b) and subplot (c), respectively, for illustration purposes.

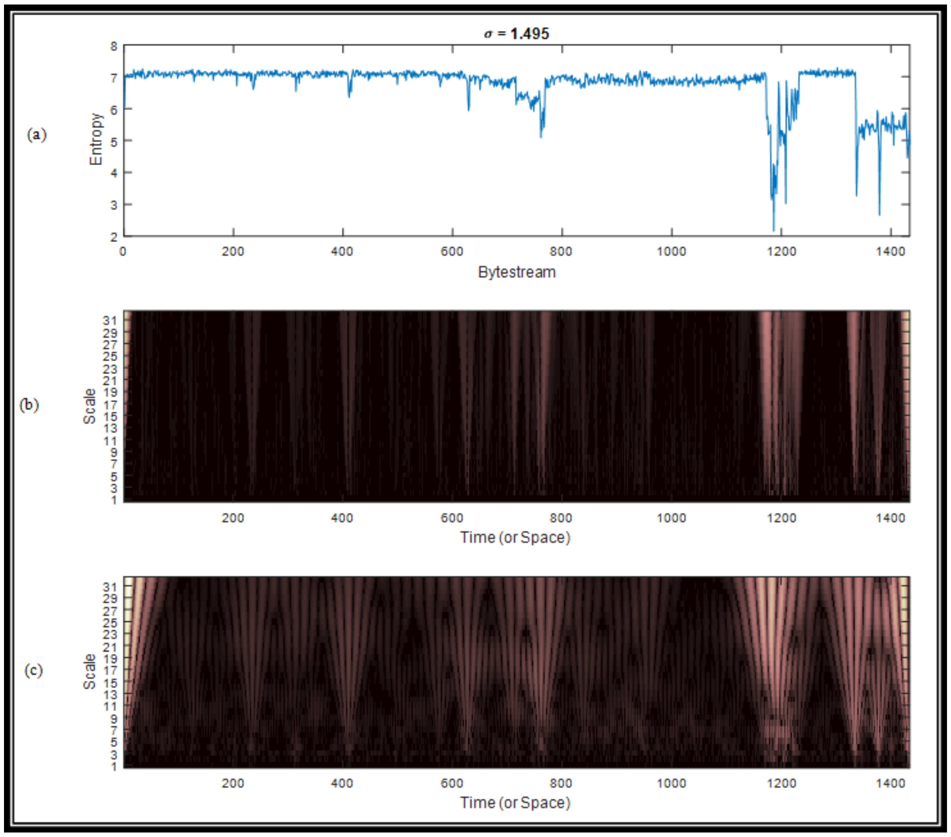

Figure 5. Entropy Profile Partitioning Illustrating the Haar versus the Morlet CWT

In this chapter we provided a brief description of the types of files included in the data set used in this study. Next, we assessed the effectiveness of the proposed methods to file type identification and partitioning, highlighting the superiority of the Haar CWT in partitioning the entropy profile. In the next chapter, the results of the application of the proposed techniques to the data sets are analyzed and conclusions drawn.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. RESULTS

The results of the approach investigated are presented in this chapter. The experiment is applied to the dataset designed for this study and individual and aggregated statistical measures derived for all files contained within. Given the large number of files analyzed and the repetitive nature of the results within file types, the tabulated results presented are limited to a sample of ten files per file type with figures for two files within each data set type presented for purposes of illustration followed by a tabulated summary of the aggregated results for the entire data set in Table 9 through Table 11. Additional entropy profile plots are presented in Appendix B.

A. STATISTICAL TECHNIQUES

1. Plain Text

Typical structural entropy profile plots obtained for plain text files are shown in Figures 6 and 7, and the statistical analysis results obtained are provided in Table 2.

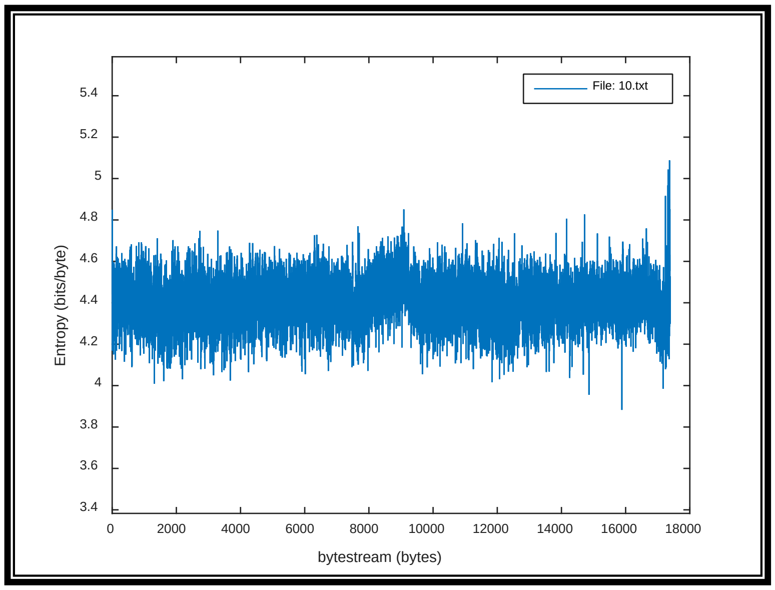

Figure 6. File: ‘10.txt’ Entropy Structure

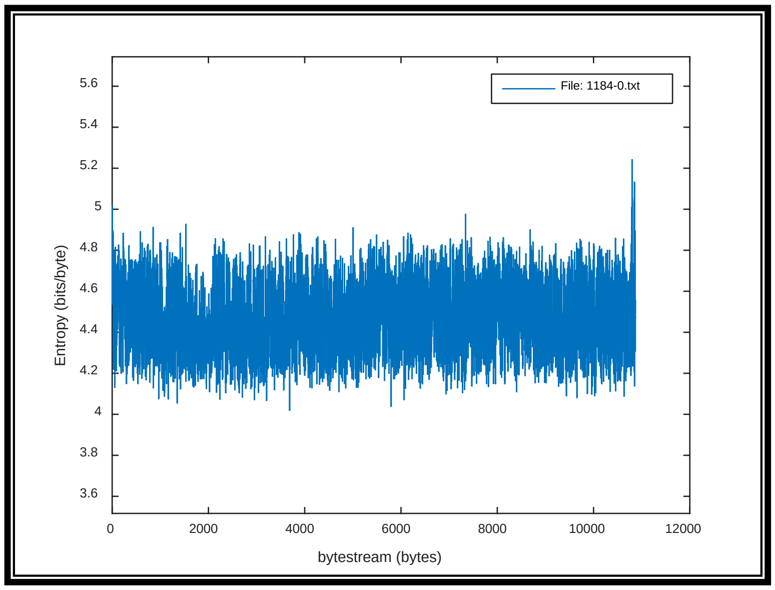

Figure 7. File: ‘1184–0.txt’ Entropy Structure

Table 2. Computed Statistical Measure Values: Plain Text Files

File Name	**Average Entropy**	**Average Hamming Weight**	**Average Arithmetic mean**
‘10.txt’	4.4076	0.4372	86.2056
‘1184-0.txt’	4.4775	0.4465	92.1870
‘1260.txt’	4.3789	0.4472	87.9721
‘1399-0.txt’	4.3978	0.4475	89.7869
‘1400-0.txt’	4.4370	0.4451	90.7995
‘1497.txt’	4.2934	0.4507	89.6131
‘203-0.txt’	4.4798	0.4459	91.5109
‘2554-0.txt’	4.4554	0.4494	91.4683
‘28054-0.txt’	4.4496	0.4466	91.9975
‘3090-0.txt’	4.3917	0.4461	90.2056

Results show the different files in the plain text data set analyzed have entropy profiles with no noticeable variations in behavior and do not visually qualify for CWT partitioning. This behavior is expected as the file structure consists entirely of plain text, i.e., it is homogeneous in nature. Further, results show the files have average entropy levels of approximately 4.4 bits/byte, and low average arithmetic means in the range 86 to 92, with approximately 45% of the files structure comprised of binary ones.

2. Native PE Files

The 32-bit and 64-bit native PE files were selected alphabetically from a 32-bit and 64-bit windows operating system.

a. 32-bit Native PE Files

The results of the analysis conducted on the 32-bit Native PE files are presented in Table 3 and Figures 8 and 9.

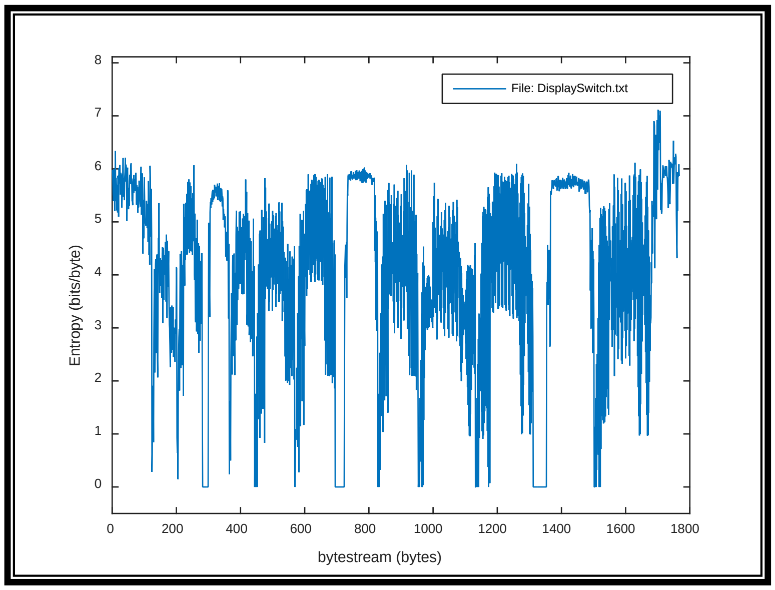

Figure 8. File: 32-bit ‘Displayswitch.txt’ Entropy Structure

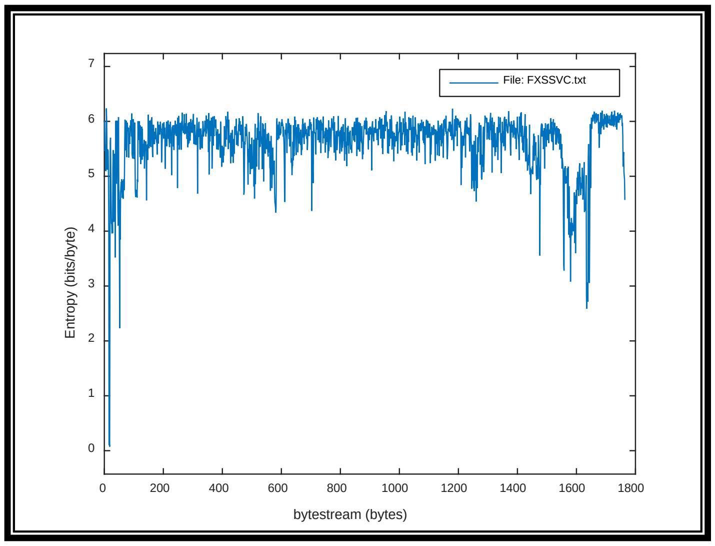

Figure 9. File: 32-bit ‘FXSSVC.txt’ Entropy Structure

Results show that the 32-bit native PE files also have similar statistical measure values irrespective of file size, as can be seen in Table 3.

Table 3. Computed Statistical Measure Values: 32-bit Native PE Files

File Name	Average Entropy	Average Hamming Weight	Average Arithmetic mean
‘DisplaySwitch.txt’	4.0511	0.4571	147.5016
‘FXSSVC.txt’	5.6148	0.4732	115.3340
‘aitstatic.txt’	5.9786	0.4927	118.2434
‘calc.txt’	6.0695	0.5021	129.7081
‘certutil.txt’	5.5500	0.4865	119.5562
‘dccw.txt’	6.0067	0.4660	115.7338
‘dfrgui.txt’	6.1701	0.5340	138.7358
‘icardagt.txt’	5.5580	0.4927	125.9011
‘ie4uinit.txt’	4.9052	0.4650	90.0859
‘lpksetup.txt’	5.6778	0.5002	128.2776

From Figures 8 and 9, we see that the 32-bit native PE files have highly structured (non-stationary) entropy profiles with noticeable variations in the entropy levels, visually qualifying these files for partitioning using the Haar CWT. The highly structured entropy profile is expected as the file architecture of the native PE files consists of distinct areas of code and data with different levels of entropy [8].

Notwithstanding a few exceptions, from the computed statistical measure values shown in Table 3, results show the 32-bit native PE files have an average entropy level of approximately 5.5 bits/byte, average Hamming weight values comparative with those of the plain text files, and average arithmetic means that are clearly distinguishable from the plain text results. Overall results show the entropy statistical measure values differ from those obtained for the plain text files sufficiently to facilitate file type discrimination and identification.

b. 64-bit Native PE Files

Sixty-four-bit native PE files are analyzed to reinforce the findings obtained with 32-bit native PE files. These files were analyzed separately and not as part of a grouping of all native PE files in order to determine if files generated from different operating systems produce similar results.

From Figures 10 and 11, we see that the 64-bit native PE files have varying entropy profiles, also representative of the distinct code and data sections [8]. This non-stationary sequence behavior exhibited by the entropy profile visually qualifies such files for partitioning. Further, results show the entropy profile plots are similar to those obtained with the 32-bit native PE files.

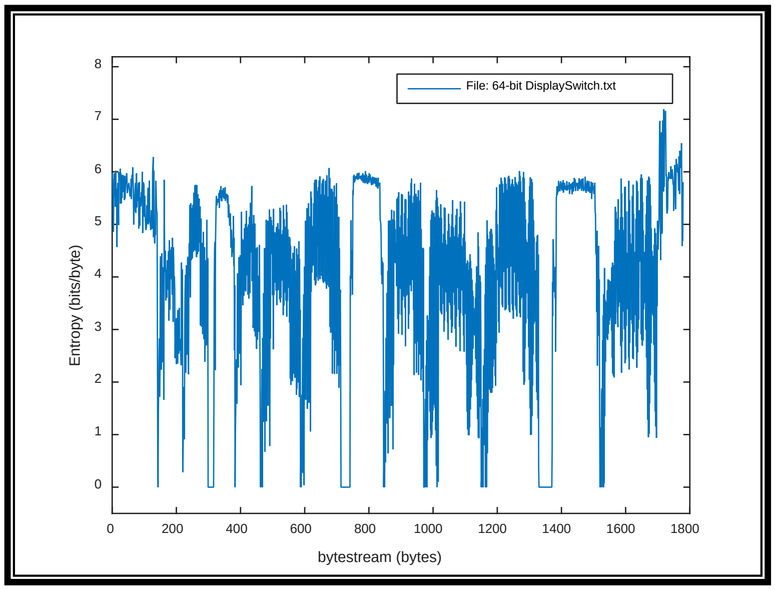

Figure 10. File: 64-bit ‘DisplaySwitch.txt’ Entropy Structure

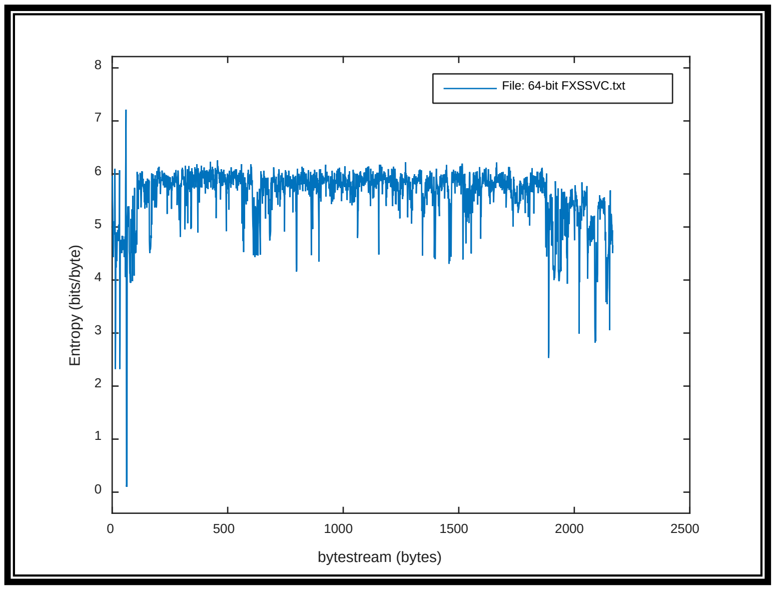

Figure 11. File: 64-bit ‘FXSSVC.txt’ Entropy Structure

Comparison of the results presented in Table 4 for the sample 64-bit native PE files and the results for the 32-bit native PE files tabulated in Table 3 further reinforce the intra-file type similarity in statistical measure values.

Table 4. Computed Statistical Measure Values: 64-bit Native PE Files

File Name	Average Entropy	Average Hamming Weight	Average Arithmetic mean
‘DisplaySwitch.txt’	4.0609	0.4550	146.8884
‘FXSSVC.txt’	5.6064	0.4316	108.3282
‘aitstatic.txt’	5.9240	0.4797	115.8622
‘calc.txt’	5.9680	0.4666	120.5114
‘certutil.txt’	5.5675	0.4387	113.3509
‘dccw.txt’	6.0031	0.4616	114.9336
‘dfrgui.txt’	6.1270	0.5214	135.6445
‘icardagt.txt’	5.2734	0.4240	113.4340
‘ie4uinit.txt’	4.9072	0.4511	87.5482
‘lpksetup.txt’	5.6148	0.4413	113.6758

3. Packed Native PE Files

The packed native PE files were generated by applying a transformation algorithm to the native PE files. Although many transformation algorithms are available, testing was restricted to the application of the freely available UPX transformation algorithm to native PE files. Packing of the native PE files using the UPX transformation algorithm resulted in a compressed file with a change in the entropy profile structure and an increase in the files entropy level due to the randomness introduced by the packer.

a. Packed 32-bit Native PE Files

From Figures 12 and 13, we conclude that packing the 32-bit native PE files increases the entropy level and changes the entropy structure of the native PE files.

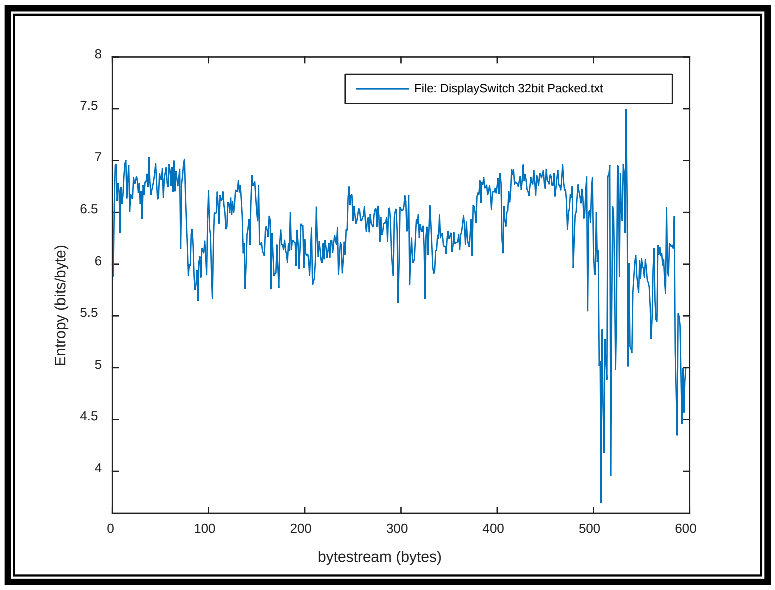

Figure 12. File: ‘DisplaySwitch 32bit Packed.txt’ Entropy Structure

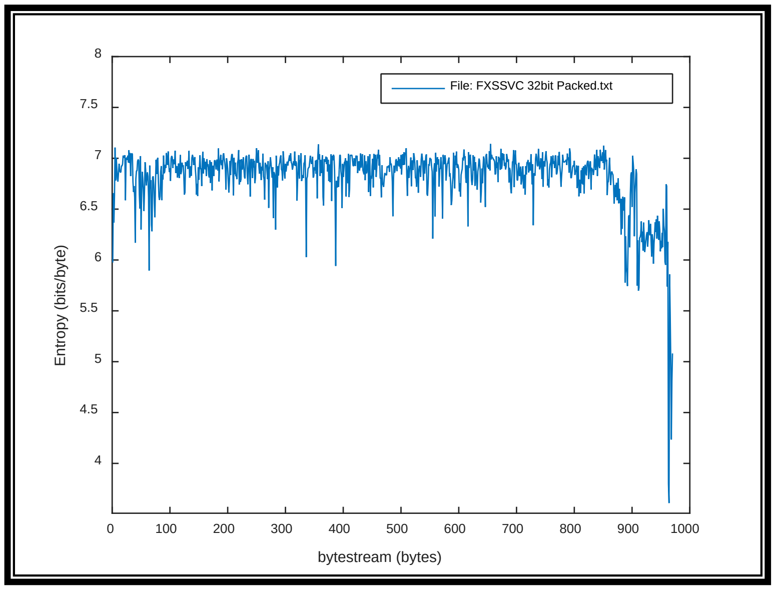

Figure 13. File: ‘FXSSVC 32bit Packed.txt’ Entropy Structure

The statistical measure values obtained for the packed 32-bit native PE files are tabulated in Table 5. Results show that compression of the 32-bit native PE files results in a change in statistical measure values noticeable in the average entropy level when compared to the uncompressed 32-bit native PE files, the plain text files, and the 64-bit native PE files. Packing has reduced the structure of the highly structured 32-bit native PE files, making files less viable for partitioning by increasing the level of randomness in the file structure.

Table 5. Computed Statistical Measure Values: Packed 32-bit Native PE Files

File Name	**Average Entropy**	**Average Hamming Weight**	**Average Arithmetic mean**
‘DisplaySwitch 32bit Packed.txt’	6.3460	0.5451	135.2697
‘FXSSVC 32bit Packed.txt’	6.8150	0.4618	110.6398
‘aitstatic 32bit Packed.txt’	6.6113	0.5317	133.2898
‘calc 32bit Packed.txt’	6.6252	0.5145	129.8092
‘certutil 32bit Packed.txt’	6.8376	0.4755	114.2390
‘dccw 32bit Packed.txt’	6.7279	0.5402	137.2465
‘dfrgui 32bit Packed.txt’	6.7058	0.5355	135.5262
‘icardagt 32bit Packed.txt’	6.8073	0.4668	112.0572
‘ie4uinit 32bit Packed.txt’	6.7209	0.4681	110.2883
‘lpksetup 32bit Packed.txt’	6.7993	0.4916	121.4254

b. *Packed 64-bit Native PE Files*

From Figures 14 and 15, we see that packing the 64-bit native PE files increases the entropy level and reduces its structure.

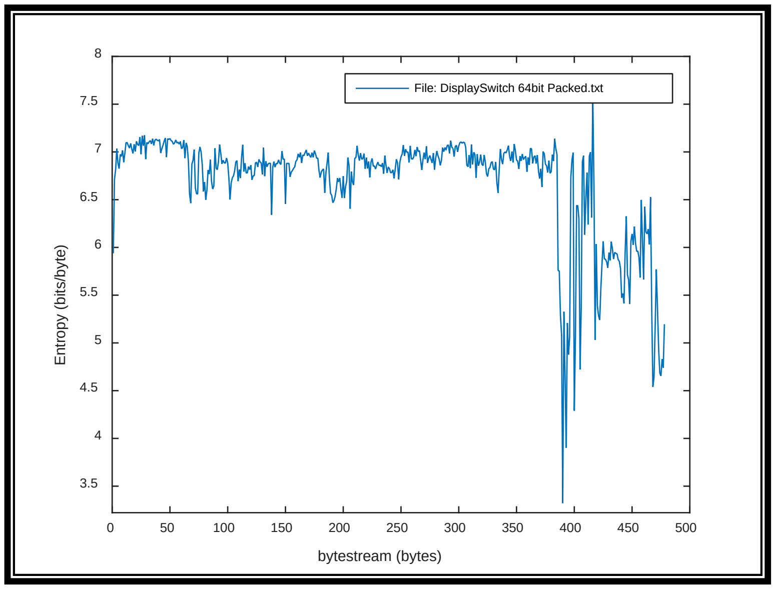

Figure 14. File: 'DisplaySwitch 64bit Packed.txt' Entropy Structure

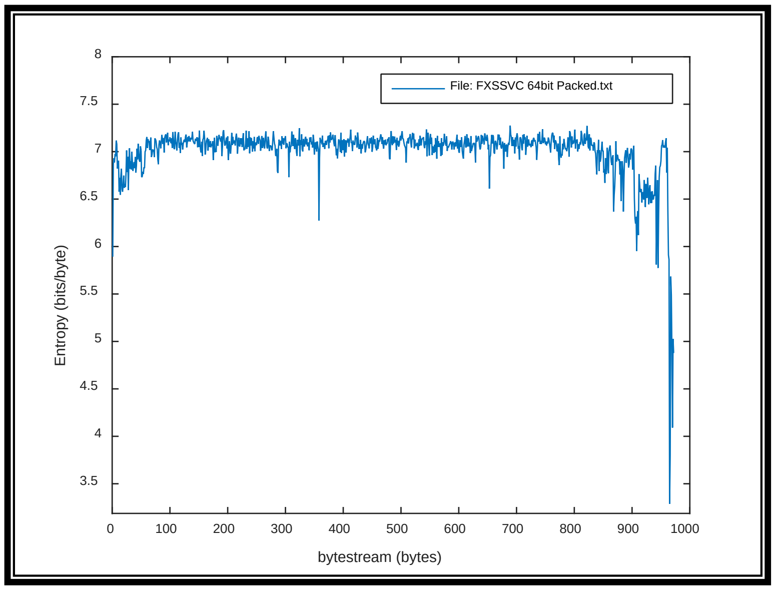

Figure 15. File: 'FXSSVC 64bit Packed.txt' Entropy Structure

Statistical measure values obtained for the packed 64-bit native PE data set are shown in Table 6.

Table 6. Computed Statistical Measure Values: Packed 64-bit Native PE Files

File Name	Average Entropy	Average Hamming Weight	Average Arithmetic mean
‘DisplaySwitch 64bit Packed.txt’	6.6782	0.5098	126.0513
‘FXSSVC 64bit Packed.txt’	7.0063	0.4708	115.0405
‘aitstatic 64bit Packed.txt’	6.8591	0.5235	130.7379
‘calc 64bit Packed.txt’	6.7795	0.5078	128.4976
‘certutil 64bit Packed.txt’	7.0172	0.4686	114.8307
‘dccw 64bit Packed.txt’	6.9114	0.5268	134.8972
‘dfrgui 64bit Packed.txt’	6.9043	0.5175	130.9301
‘icardagt 64bit Packed.txt’	6.9292	0.4721	114.3582
‘ie4uinit 64bit Packed.txt’	6.9486	0.4564	110.4250
‘lpksetup 64bit Packed.txt’	6.9149	0.4896	121.7160

Results show that the packed 64-bit native PE files have similar statistical measure values as those obtained for the packed 32-bit native PE files but differ from those obtained for the plain text and 32-bit native PE file types.

4. Encrypted Native PE Files

Testing was restricted to native PE files encrypted with the 256-bit AES algorithm.

a. Encrypted 32-bit Native PE Files

Typical structural entropy profile plots obtained after encrypting two 32-bit native PE files are shown in Figures 16 and 17. We see that encrypting the 32-bit native PE files increases the entropy level and removes all structure from the entropy profile, rendering the files immune to partitioning using the Haar CWT. This change in the structure of the entropy profile is expected as encryption removes any noticeable variations to avoid possible reverse engineering of files.

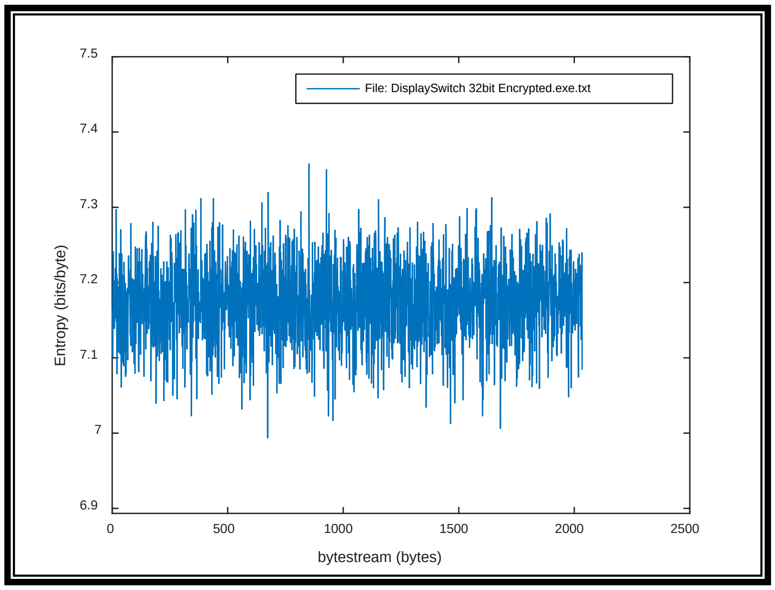

Figure 16. File: ‘DisplaySwitch 32bit Encrypted.exe.txt’ Entropy Structure

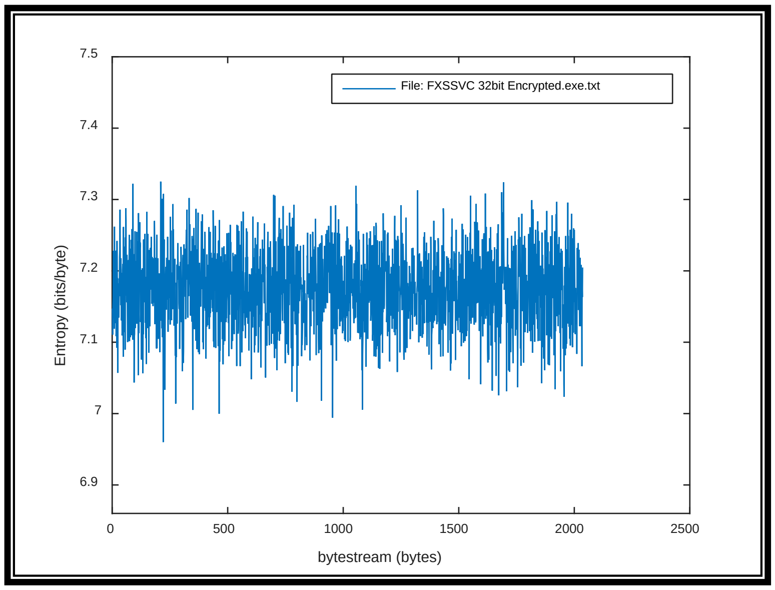

Figure 17. File: 'FXSSVC 32bit Encrypted.exe.txt' Entropy Structure

Statistical measure values obtained for the encrypted 32-bit native PE files are shown in Table 7. Results show that encrypting the 32-bit native PE files results in significant changes in the average entropy statistical measure values when compared to those of the un-encrypted 32-bit native PE files. The computed statistical measure results obtained for the different encrypted 32-bit native PE files vary by negligible amounts indicative of stationarity. Further, these encrypted files have the largest average entropy values relative to all other data set types considered in the study. This alone is not an indication of randomness as a file might have a high level of entropy, but the data might still be highly structured and as such not random [4]. A large entropy value combined with a successful DFA analysis test for stationarity is an indication of randomness.

Table 7. Computed Statistical Measure Values: Encrypted 32-bit Native PE Files

File Name	Average Entropy	Average Hamming Weight	Average Arithmetic mean
‘DisplaySwitch 32bit Encrypted.exe.txt’	7.1740	0.5019	127.9343
‘FXSSVC 32bit Encrypted.exe.txt’	7.1745	0.5022	127.9502
‘aitstatic 32bit Encrypted.exe.txt’	7.1733	0.5020	127.9740
‘calc 32bit Encrypted.exe.txt’	7.1715	0.5021	128.0909
‘certutil 32bit Encrypted.exe.txt’	7.1718	0.5018	127.9664
‘dccw 32bit Encrypted.exe.txt’	7.1726	0.5018	127.9770
‘dfrgui 32bit Encrypted.exe.txt’	7.1709	0.5019	128.0067
‘icardagt 32bit Encrypted.exe.txt’	7.1699	0.5023	128.1335
‘ie4uinit 32bit Encrypted.exe.txt’	7.1728	0.5019	128.1281
‘lpksetup 32bit Encrypted.exe.txt’	7.1738	0.5018	127.9404

b. Encrypted 64-bit Native PE Files

Typical structural entropy profile plots obtained after encrypting two 64-bit native PE files are shown in Figures 18 and 19, with any noticeable changes in the structural entropy profiles removed, also rendering the encrypted 64-bit native PE files immune to partitioning. The computed statistical measure values are shown in Table 8.

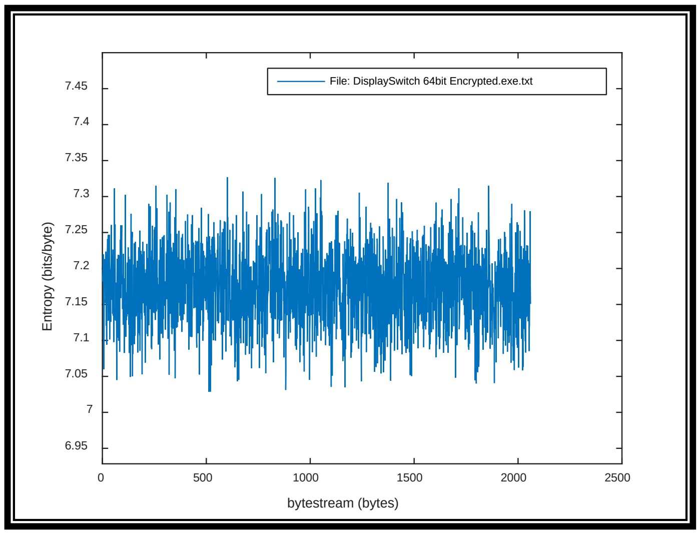

Figure 18. File: ‘DisplaySwitch 64bit Encrypted.exe.txt’ Entropy Structure

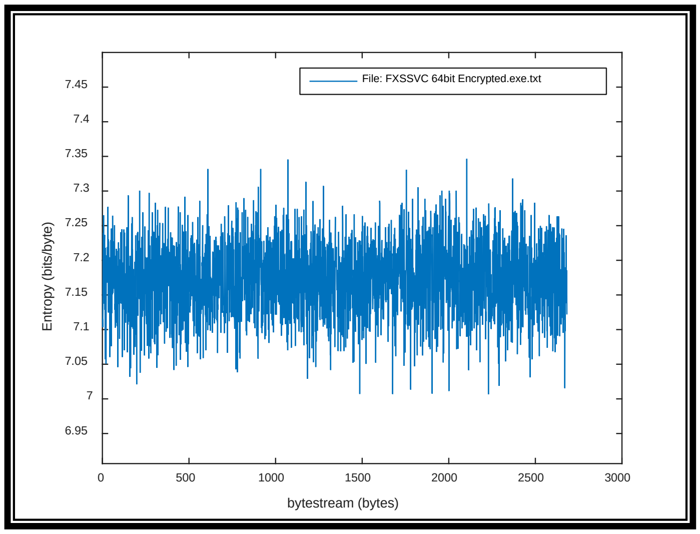

Figure 19. File: ‘FXSSVC 64bit Encrypted.exe.txt’ Entropy Structure

Table 8. Computed Statistical Measure Values: Encrypted 64-bit Native PE Files

File Name	Average Entropy	Average Hamming Weight	Average Arithmetic mean
'DisplaySwitch 64bit Encrypted.exe.txt'	7.1739	0.5018	127.9310
'FXSSVC 64bit Encrypted.exe.txt'	7.1728	0.5019	127.9472
'aitstatic 64bit Encrypted.exe.txt'	7.1728	0.5018	128.0412
'calc 64bit Encrypted.exe.txt'	7.1734	0.5018	127.8774
'certutil 64bit Encrypted.exe.txt'	7.1715	0.5018	127.8792
'dccw 64bit Encrypted.exe.txt'	7.1725	0.5020	128.0127
'dfrgui 64bit Encrypted.exe.txt'	7.1734	0.5018	127.9497
'icardagt 64bit Encrypted.exe.txt'	7.1722	0.5022	128.0133
'ie4uinit 64bit Encrypted.exe.txt'	7.1721	0.5019	127.9789
'lpksetup 64bit Encrypted.exe.txt'	7.1709	0.5017	127.9939

Results show that encrypting the 64-bit native PE files results in a significant change in statistical measure values, most noticeably in the average entropy level when compared to those obtained for the un-encrypted 64-bit native PE files. The results obtained for the encrypted 64-bit native PE files are similar to those obtained for the encrypted 32-bit native PE files; therefore, distinguishing between the two encrypted file types using unique feature vectors is not possible. In summary the different data set file types analyzed exhibit intra-file type similarity and inter-file type dissimilarity. These characteristics are fundamental to generating the unique feature vectors for file type identification.

B. FILE TYPE IDENTIFICATION

A 95% CI is computed for the aggregated average and highest average statistical measure values for each file type included in our data set to create the respective feature vectors for file type identification. A summary of resulting statistical measure values is shown in Tables 9 to 11, with results discussed next.

Table 9. Computed Entropy Statistical Measure Values. Adapted from [4].

File Type	Average Entropy	Average Entropy 95% CI (Low – High)	Highest Entropy (Average)	Highest Entropy 95% CI (Low – High)
Plain Text	**4.3842**	**4.3631 - 4.4053**	**4.4959**	**4.4748 - 4.5170**
Native Portable Executables (PE)	**5.5240**	**5.4245 - 5.6236**	**6.1701**	**6.0705 - 6.2696**
• *32-bit PE*	*5.5795*	*5.4497 - 5.7092*	*6.1701*	*6.0403 - 6.2998*
• *64-bit PE*	*5.4686*	*5.3178 - 5.6194*	*6.1270*	*5.9762 - 6.2778*
Packed Executables	**6.7377**	**6.6807 - 6.7948**	**7.0563**	**6.9992 - 7.1133**
• *32-bit Packed PE*	*6.6524*	*6.5774 - 6.7274*	*6.8472*	*6.7722 - 6.9222*
• *64-bit Packed PE*	*6.8231*	*6.7445 - 6.9017*	*7.0563*	*6.9776 - 7.1349*
Encrypted Executables	**7.1725**	**7.1723 - 7.1727**	**7.1748**	**7.1746 - 7.1750**
• *32-bit Encrypted PE*	*7.1725*	*7.1722 - 7.1728*	*7.1745*	*7.1742 - 7.1749*
• *64-bit Encrypted PE*	*7.1724*	*7.1721 -7.1727*	*7.1748*	*7.1745 - 7.1751*

Table 10. Computed Hamming Weight Statistical Measure Values

File Type	**Average Hamming Weight**	**Average Hamming Weight 95% CI (Low – High)**	**Highest Hamming Weight (Average)**	**Highest Hamming Weight 95% CI (Low – High)**
Plain Text	0.4454	0.4430 - 0.4478	0.4558	0.4534 - 0.4582
Native Portable Executables (PE)	0.4795	0.4712 - 0.4877	0.5779	0.5696 - 0.5861
• *32-bit PE*	*0.4998*	*0.4911 - 0.5085*	*0.5779*	*0.5692 - 0.5866*
• *64-bit PE*	*0.4591*	*0.4481 - 0.4702*	*0.5558*	*0.5447 - 0.5668*
Packed Executables	0.4975	0.4920 - 0.5031	0.5802	0.5747 - 0.5858
• *32-bit Packed PE*	*0.5017*	*0.4926 - 0.5108*	*0.5802*	*0.5711 - 0.5894*
• *64-bit Packed PE*	*0.4934*	*0.4872 - 0.4996*	*0.5304*	*0.5242 - 0.5366*
Encrypted Executables	0.5019	0.5019 -0.5020	0.5024	0.5024 - 0.5025
• *32-bit Encrypted PE*	*0.5019*	*0.5018 - 0.5020*	*0.5024*	*0.5024 - 0.5025*
• *64-bit Encrypted PE*	*0.5020*	*0.5019 -0.5020*	*0.5024*	*0.5024 - 0.5025*

Table 11. Computed Arithmetic Mean Statistical Measure Values

File Type	Average Arithmetic Mean	Average Arithmetic Mean 95% CI (Low – High)	Highest Arithmetic Mean (Average)	Highest Arithmetic Mean 95% CI (Low – High)
Plain Text	89.5569	88.7892 - 90.3246	92.1870	91.4192 - 92.9547
Native Portable Executables (PE)	122.1154	119.3232 - 124.9077	150.5042	147.7119 - 153.2964
• *32-bit PE*	*127.0168*	*123.5183 - 130.5152*	*150.5042*	*147.0057 - 154.0026*
• *64-bit PE*	*117.2141*	*113.3636 - 121.0646*	*146.8884*	*143.0379 - 150.7390*
Packed Executables	123.2129	121.2901 - 125.1356	144.5431	142.6203 - 146.4658
• *32-bit Packed PE*	*124.1355*	*121.0144 - 127.2566*	*144.5431*	*141.4219 - 147.6642*
• *64-bit Packed PE*	*122.2902*	*120.0393 - 124.5411*	*137.7622*	*135.5113 - 140.0131*
Encrypted Executables	127.9981	127.9788 - 128.0173	128.2141	128.1948 - 128.2333
• *32-bit Encrypted PE*	*127.9891*	*127.9585 - 128.0198*	*128.2141*	*128.1834 - 128.2447*
• *64-bit Encrypted PE*	*128.0070*	*127.9837 - 128.0303*	*128.1443*	*128.1210 - 128.1675*

1. Analysis of Results

Results show that the statistical measure CI values for the plain text data set is clearly distinguishable from the results obtained for the other data sets types. The entropy statistical measure CI values for the 32-bit and 64-bit file formats (demarcated by ⬚ cells in Table 9) overlap. There is no clear intra-file type distinction in entropy statistical measure values; thus, overall entropy statistical measure CI values are based on results obtained by combining values obtained with both 32-bit and 64-bit file formats. The entropy statistical measure CI results for the different data set types analyzed, demarcated by ▒ cells in Table 9, shows inter-file type distinction in entropy statistical measure CI range and qualifies the entropy feature as a suitable parameter for file type identification.

Results show the computed Hamming weight statistical measure values obtained for the 32-bit and 64-bit file formats also exhibit overlap in CI range, as demarcated by ▨ cells in Table 10; thus, the final Hamming weight statistical measure values are also based on results obtained by combining results obtained with 32-bit and 64-bit file formats. The combined CI results have ranges of inter-file type overlap in the native PE and packed PE file-types highest average Hamming weight range, demarcated by ▣ cells in Table 10. Further, there is overlap in the CI ranges between the packed and encrypted file-type aggregated average Hamming weight results, demarcated by ⬚ cells in Table 10; thus, overall, the CI range overlaps make the Hamming weight statistical measure unsuitable as a discriminator for file type identification.

The computed arithmetic mean CI results for the 32-bit and 64-bit file formats also exhibit overlap in CI ranges, as demarcated by ▨ cells in Table 11. As for the previous statistical measures, the computed arithmetic mean statistical measure values are based on combining files with 32-bit and 64-bit file formats. The combined CI results have ranges of inter-file type overlap in the native PE and packed PE file-types aggregated average arithmetic mean, demarcated by ▣ cells in Table 11; thus, results show the arithmetic mean also unsuitable as a discriminator for file type identification.

The final feature for file type identification shown in Table 12 includes only the entropy as a discriminant for file type identification.

Table 12. Final Entropy Statistical Measure Values. Adapted from [4].

File Type	**Average Entropy**	**Average Entropy 95% CI (Low – High)**	**Highest Entropy (Average)**	**Highest Entropy 95% CI (Low – High)**
Plain Text	4.3842	4.3631 - 4.4053	4.4959	4.4748 - 4.5170
Native Portable Executables (PE)	5.5240	5.4245 - 5.6236	6.1701	6.0705 - 6.2696
Packed Executables	6.7377	6.6807 - 6.7948	7.0563	6.9992 - 7.1133
Encrypted Executables	7.1725	7.1723 - 7.1727	7.1748	7.1746 - 7.1750

C. FILE PARTITIONING

Files determined by the DFA analysis as having highly structured (non-stationary) entropy profiles are subjected to partitioning using the Haar CWT.

1. DFA Analysis

The DFA analysis is applied to the files in the data sets to determine the DFA statistic scaling factor α which is used to identify files with non-stationary entropy profiles.

a. Encrypted Files

*S*ample figures for the application of the DFA analysis on the encrypted files are shown in Figure 20 and 21. For the samples files shown, the computed value α is around 0.5, which is representative of stationary files that show no significant changes in structural entropy profile and are not viable to partitioning.

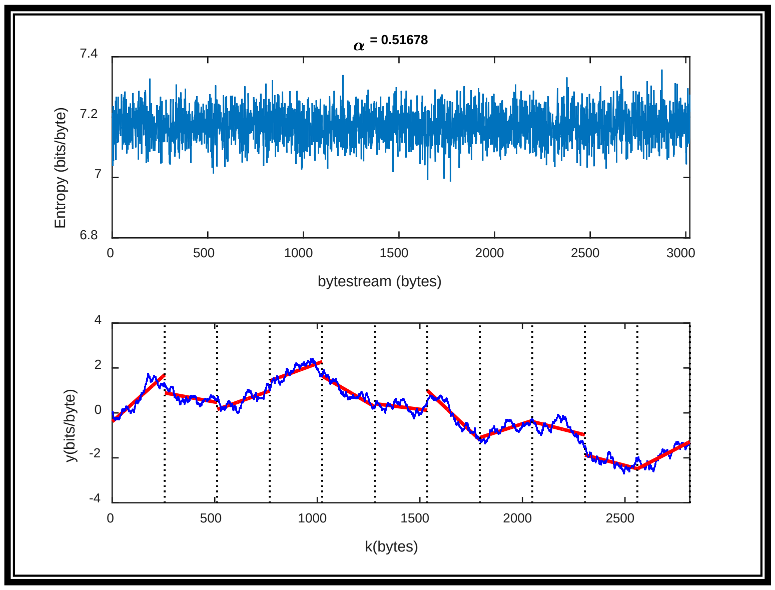

Figure 20. Encrypted File: ‘calc.exe.txt’ Entropy and DFA Plot with α= 0.51678

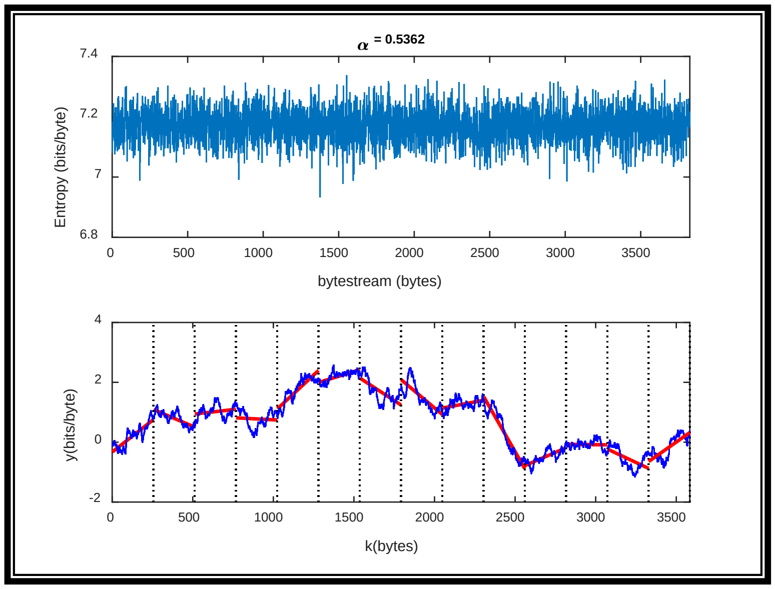

Figure 21. Encrypted File: ‘msdt.exe.txt’ Entropy and DFA Plot with α= 0.5362

b. Plain Text

From earlier figures and analysis, the plain text files used displayed properties of stationarity. Sample figures for the application of the DFA analysis on plain text files are shown in Figure 22 and 23. Results show the computed α value is around 0.5, which is representative of stationary behavior so that further partitioning is not needed. Further, this further strengthened the DFA criterion that files with α≈0.5 exhibit stationary entropy profiles.

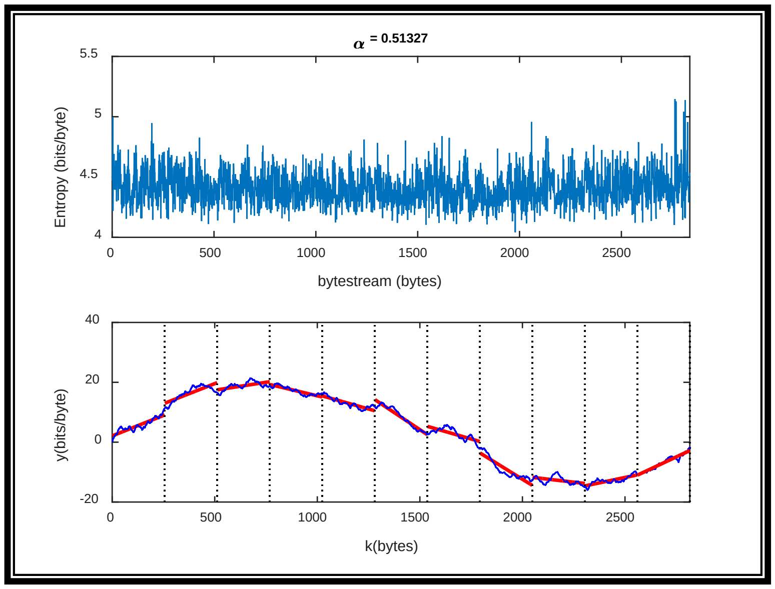

Figure 22. Plain Text File: ‘1342–0.txt’ Entropy and DFA Plot with α= 0.51327

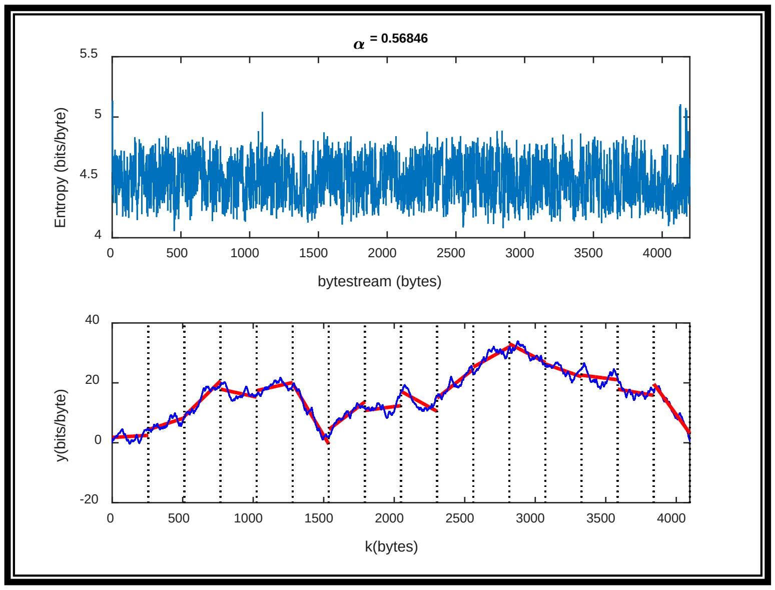

Figure 23. Plain Text File: ‘203–0.txt’ Entropy and DFA Plot with α= 0.56846

c. *Native PE and Packed PE Files*

The native PE files and packed PE files included in our dataset exhibit entropy profiles with large variations, with the native PE files having a greater frequency of occurrence of highly structured files than packed PE files. Evaluations show that files with $\alpha > 1$ are more likely to have highly structured entropy profiles, but this does not preclude files with $\alpha < 1$ from having highly structured entropy, as illustrated in Figures 24 and 25. Depending on the extent and effectiveness of the packing (compression) algorithm, we see that the packed PE files can still exhibit changes in structural entropy, as illustrated in Figure 26. As a quick determination of files with highly structured entropy, the limit of $\alpha > 1$ is a fair assumption without the requirement to visually determine whether files have highly structured entropy profiles or not.

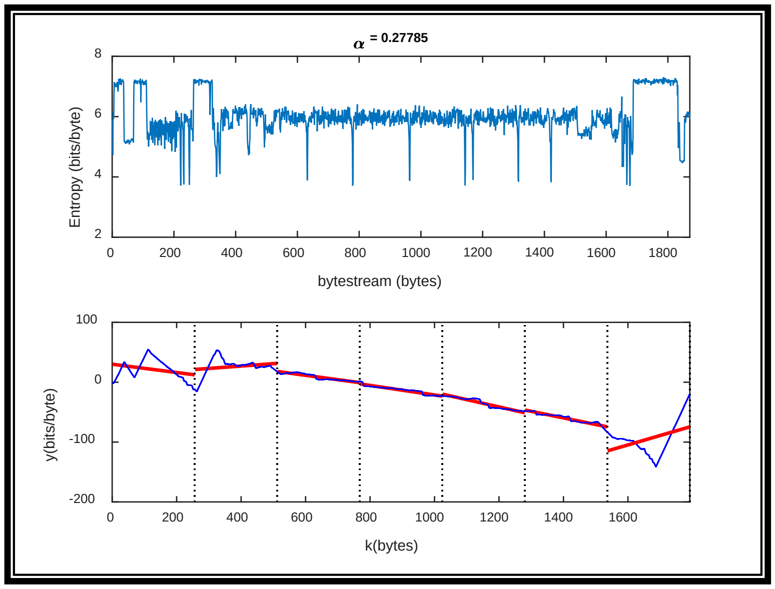

Figure 24. Native PE File: ‘RMActivate_ssp.txt’ Entropy and DFA Plot withα=0.27785

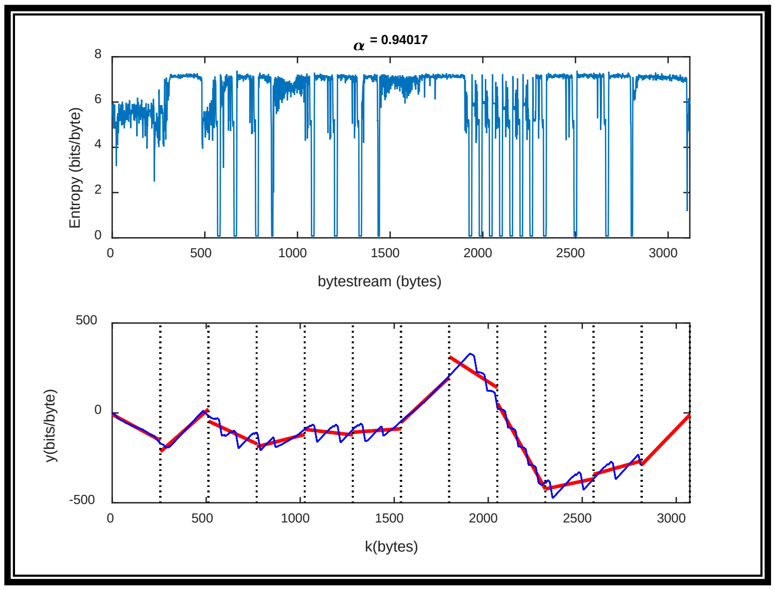

Figure 25. Native PE File: ‘dccw.txt’ Entropy and DFA Plot withα=0.94017

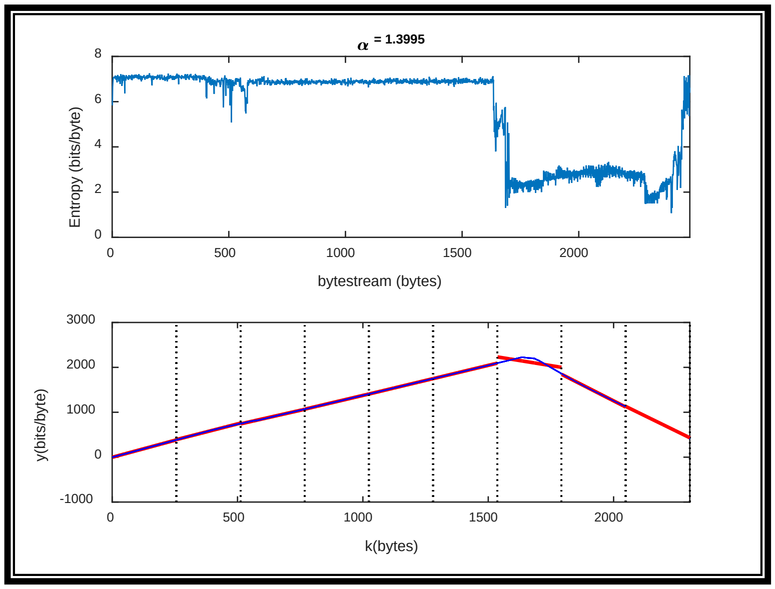

Figure 26. Packed PE File: ‘FlashUtil64_24_0_0_194_ActiveX.txt’ Entropy and DFA Plot withα=1.3995

2. CWT Partitioning

The purpose of this section is to demonstrate the effectiveness of the Haar CWT as a tool for file partitioning. The goal is to provide a convenient and quick method to partition the file into its distinct code and data sections as well as to possibly identify suspicious file regions for further analysis. No detailed entropy analysis of the partitioned structure is undertaken as this requires developing feature vectors for the numerous file types available which could include compressed files (gif, mpeg, jpeg, pdf, etc.), machine code (Windows and Linux operating system based), data, regular functions, library functions, instructions, zipped files, etc. The application is only limited by the availability of the training data to generate the entropy statistical measure values needed for the file type identification step. The ability of the Haar CWT in detecting transitional changes in entropy profile level, making it suitable for file partitioning, is illustrated in Figure 27 through Figure 29. Recall that changes in entropy profile levels are indicative of changes in the file structure, and the Haar CWT can be used to partition the file into distinct regions.

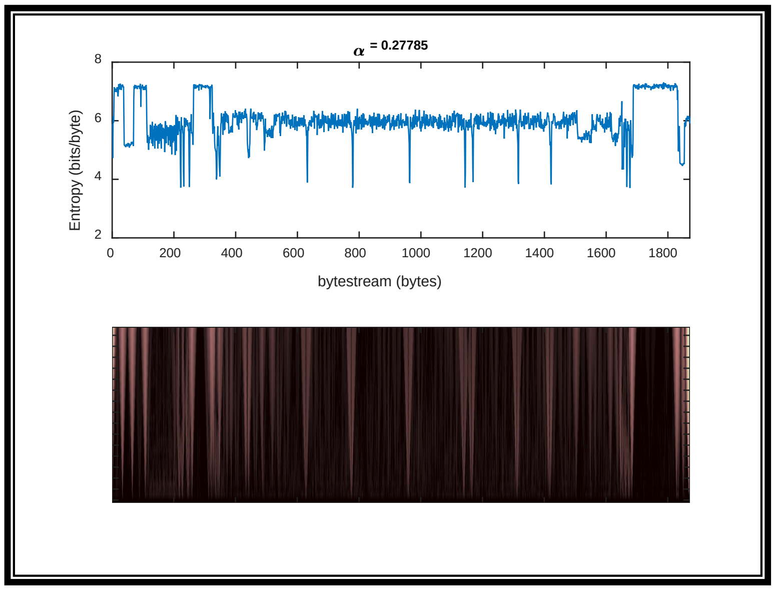

Figure 27. Partitioning of File: ‘RMActivate_ssp.txt’

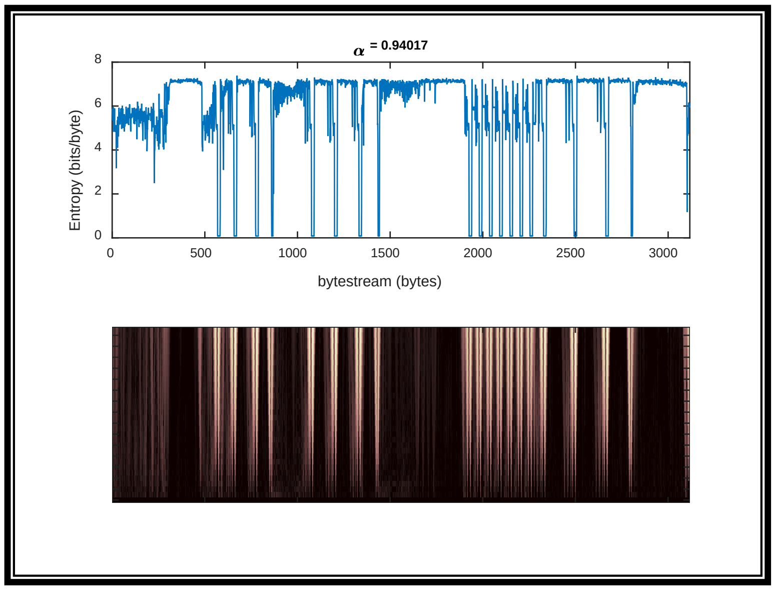

Figure 28. Partitioning of File: 'dccw.txt'

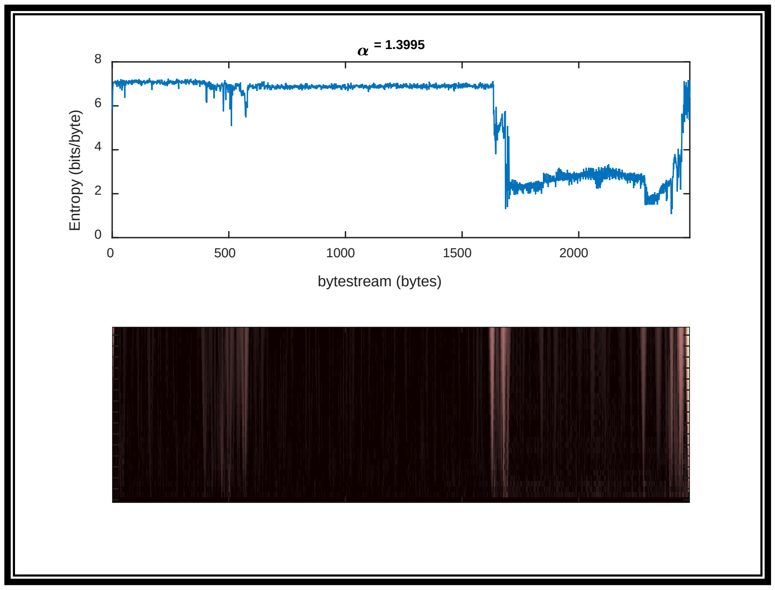

Figure 29. Partitioning of File: 'FlashUtil64_24_0_0_194_ActiveX.txt'

In this chapter the results from the application of the proposed techniques to the data set developed for the study were presented. Results show that the entropy feature is a suitable parameter for file type identification, and DFA analysis can be used as a quick determination of files with highly structured entropy provided that the DFA statistic scaling factor $\alpha > 1$. Further, results show that the Haar CWT is effective in partitioning the file by detecting the transitional changes in entropy profile levels. In the next chapter, we summarize findings and make recommendations for future work.

V. CONCLUSIONS AND FUTURE WORK

In this thesis, we reviewed the current literature dealing with statistical methods for file-type identification and partitioning. Three different statistical measures were investigated in our study: Shannon entropy, Hamming weight and Arithmetic mean. The proposed methods were applied to a generated data set consisting of files with plain text, native PE, packed PE, and encrypted PE formats with the aim of generating a feature vector for file type identification. Experimental results indicate that only the entropy profile was useful to distinguish between plain text, native PE, packed native PE, and encrypted native PE file types.

We also examined the use of the detrended fluctuation analysis as a means of identifying files that have highly structured entropy behavior, which tends to indicate non-stationarity in the entropy profile. Finally, the CWT was applied to the entropy profiles identified as non-stationary to partition the file into distinct regions.

Future work could include analyzing a dataset including a larger set of file types and considering the implementation of an automated approach for file type identification and partitioning.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. MATLAB SCRIPTS

A. FEATURE VECTOR EXTRACTION MAIN SCRIPT

```
%****************************************************************
% Calvin Brendan Paul
% Naval Postgraduate School
% July 2017
% Main Script
% Feature_Vector_Extraction.m
% This script calls the File_Stats and DFA function
% Calculates the data set aggregate average and highest average CI range
% for entropy profile, Hamming weight (HW) profile, arithmetic mean profile
% NOTE1: Input file type of .txt extension
% NOTE2: Limit the number of files evaluated as processing is
% computationally intensive
% NOTE3: Adjust WindowLength and overlap as required.
%%%%%%%%%%%%%%% PARAMETERS %%%%%%%%%%%%%%%
% USER INPUT:
% 1) files: files to be analyzed
% 2) WindowLength: WindowLength over which to calculate file
% statisic measures
% 3) Overlap: Overlap between windows
%****************************************************************
clear all
close all
files = dir('*.txt'); % loads files to be analysed into file struct
WindowLength=256;overlap=0;% define WindowLength and overlap
% Calls Function: File_Stats
% Variables passed: files,WindowLength,overlap
% Variables returned:
% 1) entr_all: entropy profile for data set analysed
% 2) mean_entropy_each: average entropy profile per file analysed
% 3) mean_mean_each: average mean profile per file analysed
% 4) mean_HammingWeight_each: average HammingWeight profile per file analysed
% 5) count: total number of chunks analysed
[entr_all,mean_entropy_each,mean_mean_each,mean_HammingWeight_each,...count]=F
ile_Stats(files,WindowLength,overlap);
%%
%calculates the entropy CI range for the aggregate average and highest
%average entropy profile
stdE_y95=std(mean_entropy_each);% determines the standard deviation
meanE_y95=mean(mean_entropy_each); %de-select if calculating Confidence
%Interval (CI) range for max entropy
```

```
meanE_y95=max(mean_entropy_each); %select if calculating CI
%range for max entropy
CI=0.95; %adjust to required CI
alpha=1-CI; % level of significance
Nb=length(mean_entropy_each);
% calculates the CI range for selected parameter
Z_alpha2=qfuncinv(alpha/2);
myE_lower95=meanE_y95-((Z_alpha2*stdE_y95)/sqrt(Nb));
myE_upper95=meanE_y95+((Z_alpha2*stdE_y95)/sqrt(Nb));
CILimitsE=[myE_lower95;myE_upper95];
%%
%calculates the HW CI range for the aggregate average and highest
%average HW profile
stdHW_y95=std(mean_HammingWeight_each);
meanHW_y95=mean(mean_HammingWeight_each); %de-select if calculating CI
%range for max HW
%meanHW_y95=max(mean_HammingWeight_each); %select if calculating CI
%range for max HW
CI=0.95;% adjust to required CI
alpha=1-CI; % level of significance
Nb=length(mean_HammingWeight_each);
% calculates the CI range for selected parameter
Z_alpha2=qfuncinv(alpha/2);
myHW_lower95=meanHW_y95-((Z_alpha2*stdHW_y95)/sqrt(Nb));
myHW_upper95=meanHW_y95+((Z_alpha2*stdHW_y95)/sqrt(Nb));
CILimitsHW=[myHW_lower95;myHW_upper95];
%%
%calculates the Arithmetic mean(AM) CI range for the aggregate average and
%highest average AM profile
stdAM_y95=std(mean_mean_each);
meanAM_y95=mean(mean_mean_each); %de-select if calculating CI
%range for max AM
%meanAM_y95=max(mean_mean_each); %select if calculating CI
%range for max AM
CI=0.95; % adjust to required CI
alpha=1-CI; %level of significance
Nb=length(mean_mean_each);
% calculates the CI range for selected parameter
Z_alpha2=qfuncinv(alpha/2);
myAM_lowerCI=meanAM_y95-((Z_alpha2*stdAM_y95)/sqrt(Nb));
myAM_upperCI=meanAM_y95+((Z_alpha2*stdAM_y95)/sqrt(Nb));
CILimitsAM=[myAM_lowerCI;myAM_upperCI];
%%%END%%%
```

B. FILE_STATS FUNCTION

```
%****************************************************************
% Calvin Brendan Paul
% Naval Postgraduate School
% July 2017
% File_Stats.m Function
% This script calls the DFA Function
%****************************************************************
%%%%%%%%%%%%%%% PARAMETERS %%%%%%%%%%%%%%%
% Variables passed: files,WindowLength,overlap
% Variables returned:
% 1) entr_all: entropy profile for data set analysed
% 2) mean_entropy_each: average entropy profile per file analysed
% 3) mean_mean_each: average mean profile per file analysed
% 4) mean_HammingWeight_each: average HammingWeight profile per
% file analysed
% 5) count: total number of chunks analysed
function  [entr_all,mean_entropy_each,mean_mean_each,...
mean_HammingWeight_each,count]= File_Stats(files,WindowLength,overlap)
%%%%%declare variables needed
count=0;
mean_entropy_each=[];mean_mean_each=[];
mean_HammingWeight_each=[];
entr_all=[];mean_all=[];
Hweight_all=[];alpha_all=[];
%%%%%
for i=1:length(files)% for loop conditioned on number of files
b=textread(files(i).name,'%2c'); % read in text files as character
b=hex2dec(char(b)); % convert from hexadecimal to decimal
b=single(reshape(b,1,[])); %reshape file as row vector
b=b(b~=0); % remove zeros
NumOfFrames=floor(length(b)/(WindowLength)); % calc number of file frames
M=b(1:NumOfFrames*WindowLength); % truncate file
M=M';
curPos=1; % set curPos to one
entr=[];meanc=[];Hweight=[];
for j=1:NumOfFrames;% for loop conditioned on number of frames (per file)
   %calculates statistical measures per window
   c = M(curPos:curPos+WindowLength-1);
   binranges=unique(c);[bincounts] = histc(c,binranges);
   p=bincounts/sum(bincounts);
   entropy_1=sum(p.*log2(1./p)); % calculates window entropy
   entr=[entr,entropy_1]; % creates entropy profile
   meanc=[meanc,mean(c)]; % creates mean profile
   Hweightf=nnz(dec2bin(c,8).' == '1' )/numel(dec2bin(c,8));
```

```
    Hweight=[Hweight,Hweightf]; % creates Hamming weight profile
    curPos=curPos+WindowLength-overlap; % slide window
    count=count+1; % increment chunk count
end
%Average statistic measure per file analysed
mean_entropy_each=[mean_entropy_each,mean(entr)];
mean_mean_each=[mean_mean_each,mean(meanc)];
mean_HammingWeight_each=[mean_HammingWeight_each,mean(Hweight)];

%Column vectors of statistic measure profiles
entr_all(1:numel(entr),i)=entr;
mean_all(1:numel(meanc),i)=meanc;
Hweight_all(1:numel(Hweight),i)=Hweight;

figure
plot(entr);xlabel('bytestream (bytes)');ylabel('Entropy (bits/byte)');
legend(['File: ' , files(i).name],'fontsize',10);
ylim([min(entr)-0.5 max(entr)+1]);xlim([0 length(entr)]);
%legend boxoff
h =  findobj('type','figure');
n = length(h); % number figures plotted thus far
DFAWindowLength=256; %windowlength to perform DFA over
%calls the DFA function
% Variables passed:
% 1) entr_all: entropy profile
% 2) n: number of figures plotted so far
% 3) DFAWindowLength: windowlength to perform DFA over
% 4) files: files being analysed
% Variables returned:
% 1) alpha: DFA statistic scaling factor
alpha=DFA(entr_all(:,i),n,DFAWindowLength,files(i,:));
alpha_all=[alpha_all,alpha];% DFA statistic scaling factors for all files
end
end
%%%END%%%
```

C. DFA FUNCTION

```
%**********************************************************
% Calvin Brendan Paul
% Naval Postgraduate School
% July 2017
% DFA.m Function
% This script calls the CWT Function
%**********************************************************
%%%%%%%%%%%%%% PARAMETERS %%%%%%%%%%%%%%
```

```
% Variables passed:
% 1) entr_all: entropy profile
% 2) n: number of figures plotted so far
% 3) DFAWindowLength: windowlength to perform DFA over
% 4) files: files being analysed
% Variables returned:
% 1) alpha: DFA statistic scaling factor
%**********************************************************
function alpha = DFA(entr_all,n,DFAWindowLength,files)
ss=size(entr_all); % determine number of entropy profiles
test1=entr_all'; % reshape
k=ss(2); % determine for loop variable
f_all1=cell(1,k);y_all=cell(1,k);% variable setup
for i=1:k % for loop conditioned on the number of entropy profiles
u=test1(i,1:end); % read in entropy profile for analysis
u=u(u~=0); % remove zeros
mt=mean(u); % determine mean
yk=cumsum(u-mt); % determine running sum y(k)
NumOfFrames=floor(length(yk)/(DFAWindowLength)); % calc number of frames
x=1:1:length(yk); % needed for polyfit x variable
yk=yk(1:NumOfFrames*DFAWindowLength); % truncated entropy profile
y_all{1,i}=yk; % stores truncated entropy profiles into a cell
x=x(1:NumOfFrames*DFAWindowLength);%truncated to length of entropy profile
curPos=1; % set curpos to 1
count=0; % count of number of frames analysed
vert=[];f=[];
for j=1:NumOfFrames % for loop conditioned on number of frames
   y_w = yk(curPos:curPos+DFAWindowLength-1); %sliding window for y
   x_w = x(curPos:curPos+DFAWindowLength-1); %sliding window for x
   curPos=curPos+DFAWindowLength; % cursor positioned to start of
   %new window
   count=count+1; % increment counter
   vert=[vert,j*DFAWindowLength];
   [p1,s1]=polyfit(x_w,y_w,1); %determine OLS regression line polynomial
   y1=polyval(p1,x_w);% Determine data points for OLS line
   f=[f,y1]; % vector of OLS line points
   figure (i+n);
   subplot(211);plot(u); % plot entropy profile
   %title(['Entropy using probabilities: ' ,files(i).name],'fontsize',10);
   %legend(['File: ' , files(i).name],'fontsize',10);
   %legend boxoff;
   xlabel('i');ylabel('u(i)');
   xlim([0 length(u)]);
   subplot(212); % plot showing DFA analysis
   if count==NumOfFrames;
```

```
            yTicks = get(gca, 'ytick');
            yline= linspace(min(yTicks),max(yTicks),length(vert));
            for ii=1:length(vert);
                xline = ones(length(yline))*vert(ii);
                plot(xline,yline,'k:','LineWidth',2) % plot vertical lines
            end
        end
        plot(x,yk,'b');hold on  % plot running sum
        plot(x_w,y1,'r-','LineWidth',3); % plot OLS lines
        xlabel('bytestream(bytes)');ylabel('Entropy(bits/byte)');
        xlim([0 length(x)]);
    end
    f_all1{1,i}=f; %cell with OLS line points for all the entropy profile
    end
    s=size(f_all1);
    alpha=[]; % variable to save DFA statistic scaling factor
    for i=1:s(2); %for loop conditioned on number of entropy profiles
    f_all=f_all1(1,i); % read in OLS line point for entropy profile (i)
    f_all=cell2mat(f_all);
    yk=y_all(1,i);% y(k) for specific entropy profile
    yk=cell2mat(yk);
    RMSE=sqrt(1/length(f_all)*cumsum((f_all-yk).^2)); % determine RMSE
    RMSE=log2(RMSE); % log of RMSE
    x=1:1:length(f_all);
    [p2,s2]=polyfit(log2(x),RMSE,1); %Log fit to determine DFA exponential
    gradient=num2str(p2(1)); %alpha converted to a string
    subplot(211);title(['\alpha = ',gradient],'Fontsize',10);
    alpha=[alpha,p2(1)]; %DFA statistic scaling factor for all entropy profiles
    end
    h =  findobj('type','figure');
    n2 = length(h); % number of figures plotted thus far
    % Calls the CWT function for file partitioning
    CWT(files,n2,entr_all); % partitions the entropy profile
    % Note: All files are partitioned however the function could be
    % conditioned to partition only files that meet the DFA requirements
    % Variables passed:
    % 1) entr_all: entropy profile
    % 2) n2: number of figures plotted so far
    % 3) files: files being analysed
    % Variables returned: None
    % 1) alpha: DFA statistic scaling factor
    end
    %%%END%%%
```

D. CWT FUNCTION

```
%*************************************************************
% Calvin Brendan Paul
% Naval Postgraduate School
% July 2017
% CWT.m Function
% OUTPUT: Plots graphs showing CWT segmentation
% INPUT: files, n2, entr_all
%*************************************************************
%%%%%%%%%%%%%%%% PARAMETERS %%%%%%%%%%%%%%%%
% Variables passed:
% 1) files: files to be analysed
% 2) n2: number of figures plotted in DFA
% 3) entr_all: entropy profiles
% Variables returned: None
function CWT(files,n2,entr_all)
w=size(entr_all);
w=w(:,2); % determine number of files to segment
for i=1:w; % for loop conditioned on number of files
entr_series=entr_all(:,i); % read entropy profile to be segmented
entr_series=entr_series(entr_series~=0);
figure (i+n2); % plots sequential figures
subplot(211);plot(entr_series); % plot entropy profile
legend(['File: ' , files(i).name],'fontsize',10);
%legend boxoff;
ylabel('Entropy (bits/byte)');xlabel('bytestream (bytes)');
xlim([0,length(entr_series)]);
subplot(212);cwt(entr_series,1:1:64,'haar','plot'); %partitioned
%entropy profiles
title(' ');ylabel('Scale');xlabel('bytestream (bytes)')
end
end
%%%END%%%
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. ENTROPY PROFILE PLOTS

A. PLAIN TEXT FILES

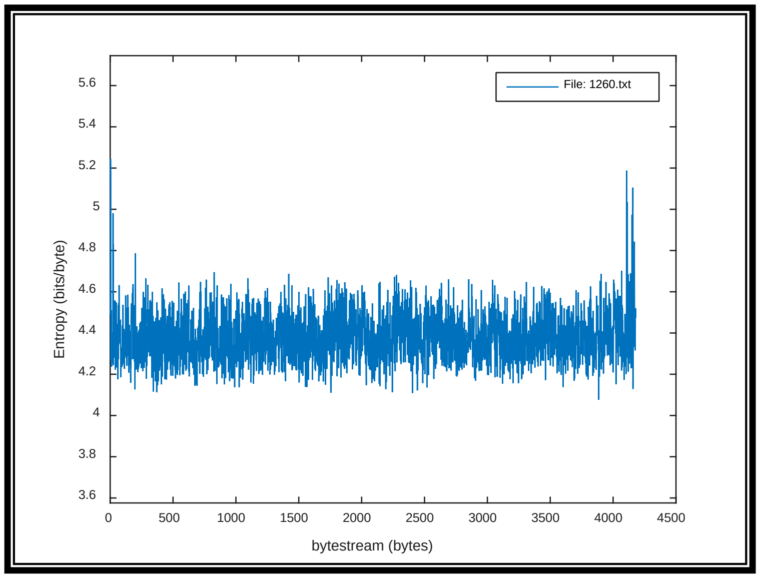

Figure 30. File: '1260–0.txt' Entropy Structure

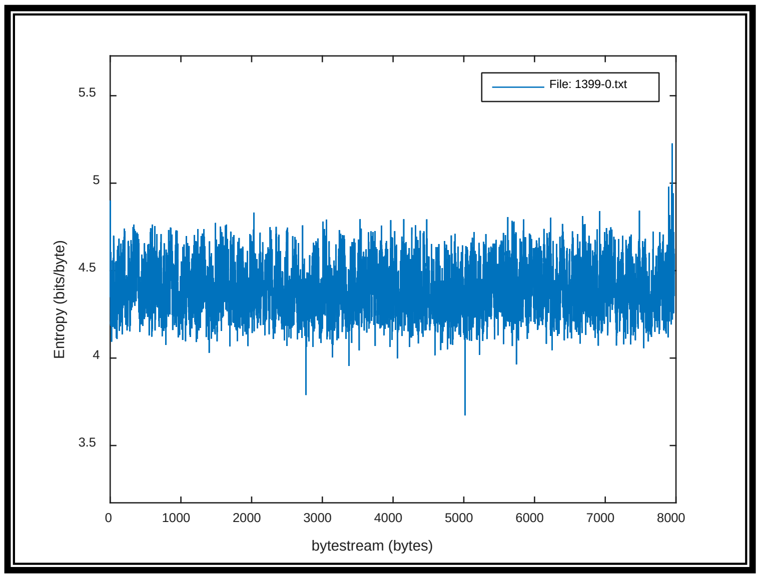

Figure 31. File: '1399.txt' Entropy Structure

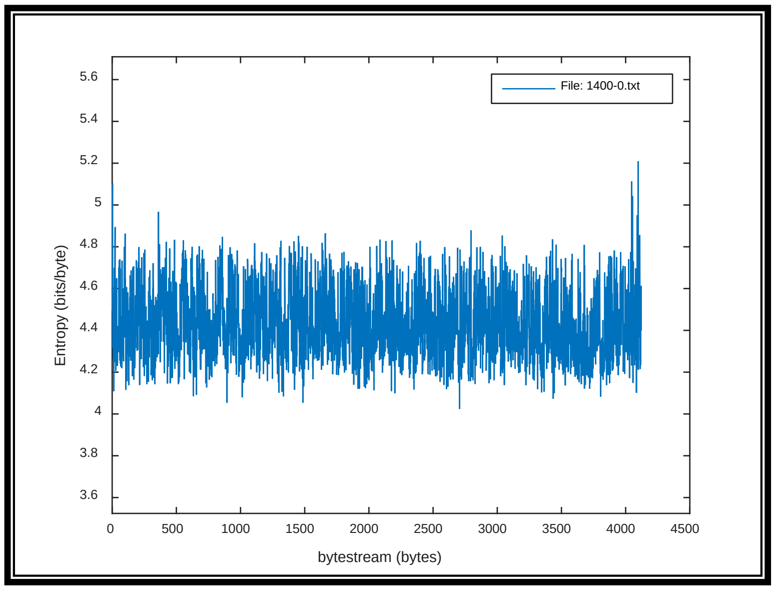

Figure 32. File: ‘1400–0.txt’ Entropy Structure

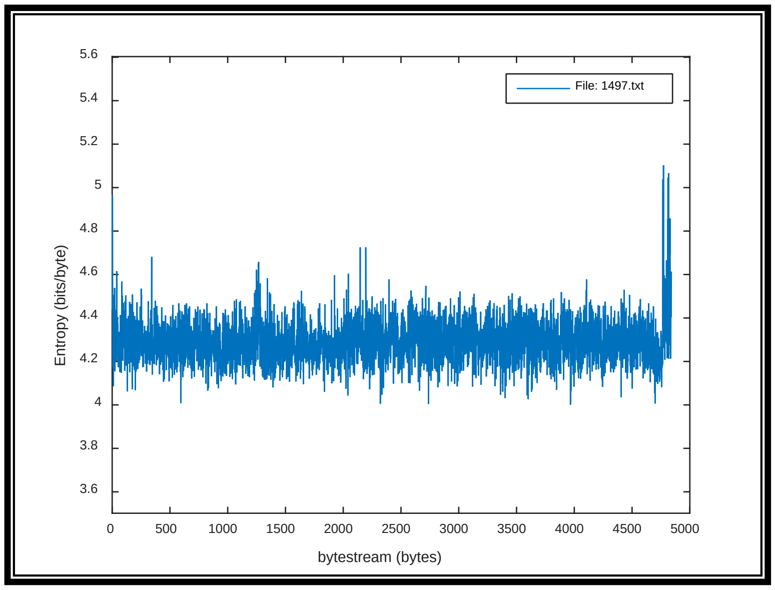

Figure 33. File: ‘1497.txt’ Entropy Structure

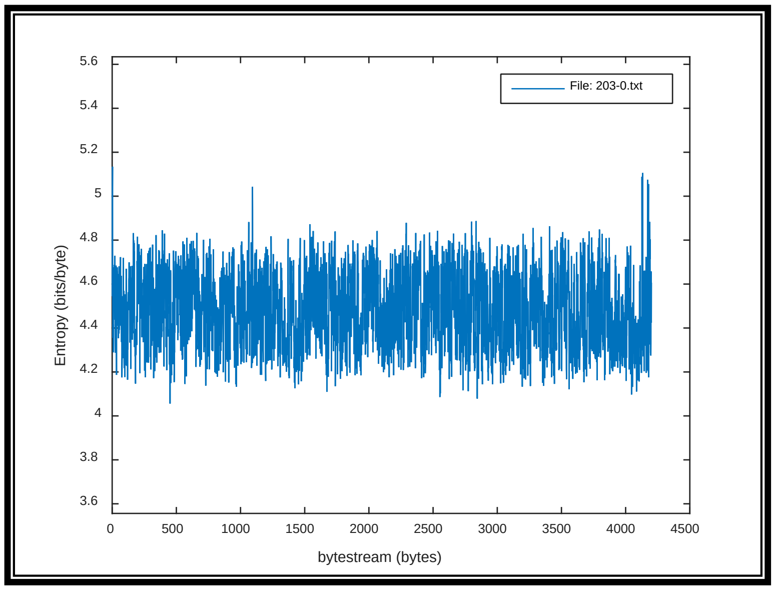

Figure 34. File: '203–0.txt' Entropy Structure

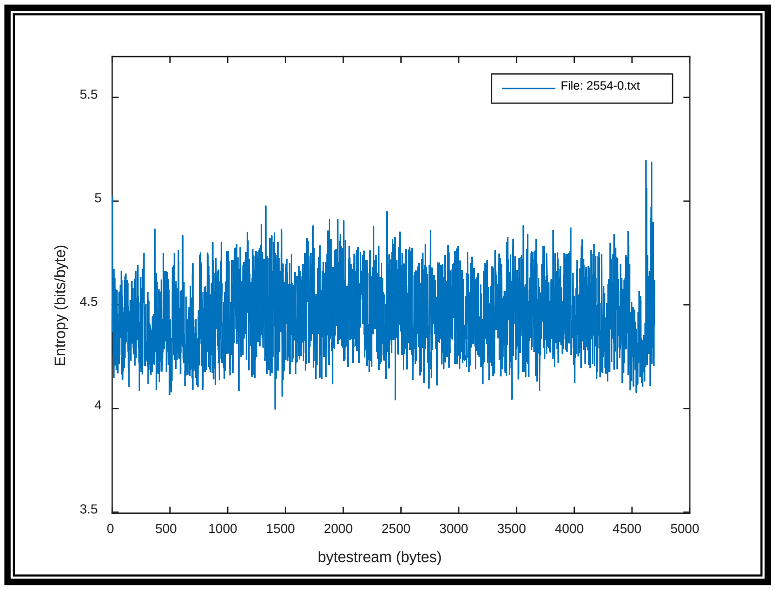

Figure 35. File: '2554–0.txt' Entropy Structure

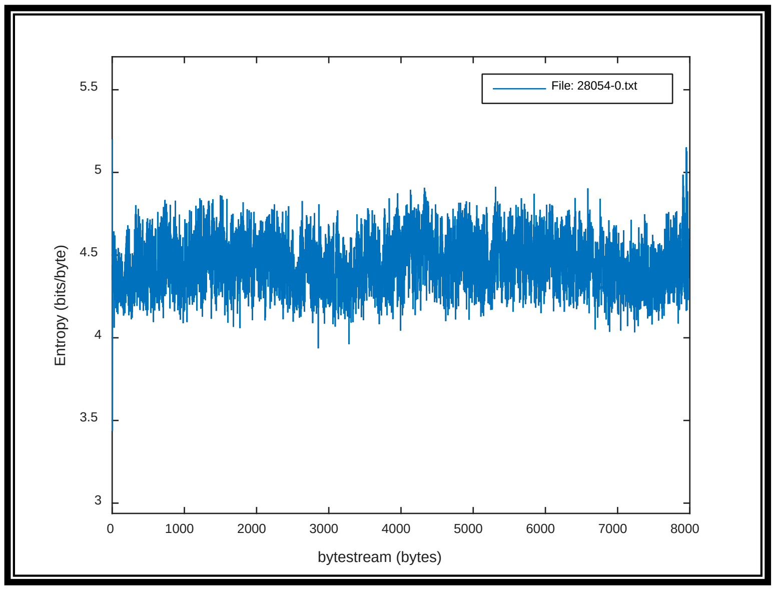

Figure 36. File: ‘28054-0.txt’ Entropy Structure

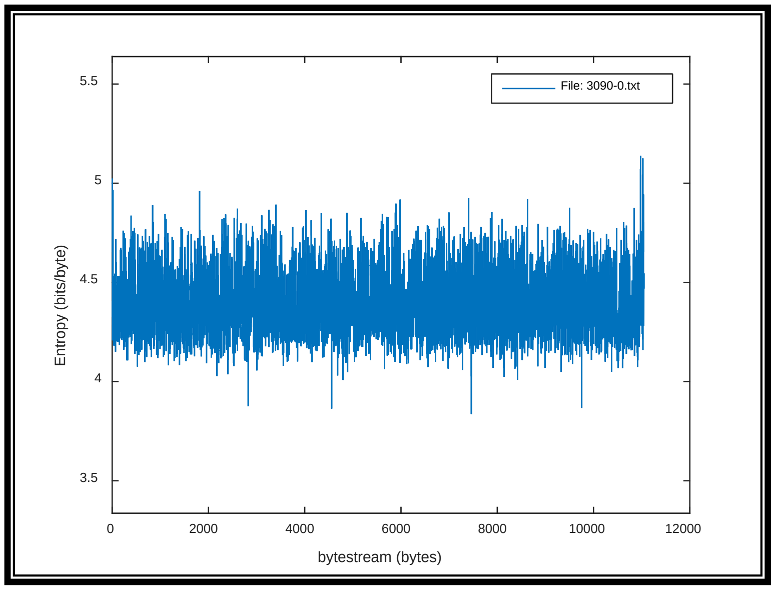

Figure 37. File: ‘3090–0.txt’ Entropy Structure

B. 32-BIT NATIVE PE FILES

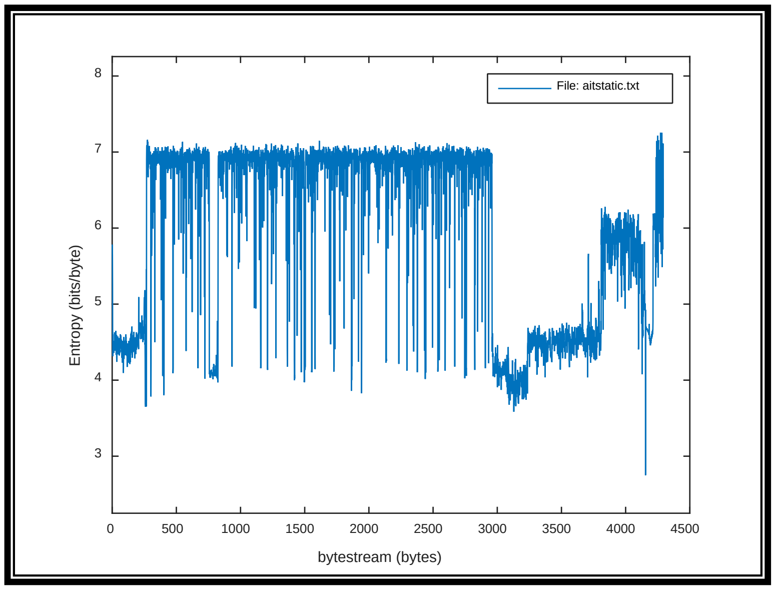

Figure 38. File: 32-bit ‘aitstatic.txt’ Entropy Structure

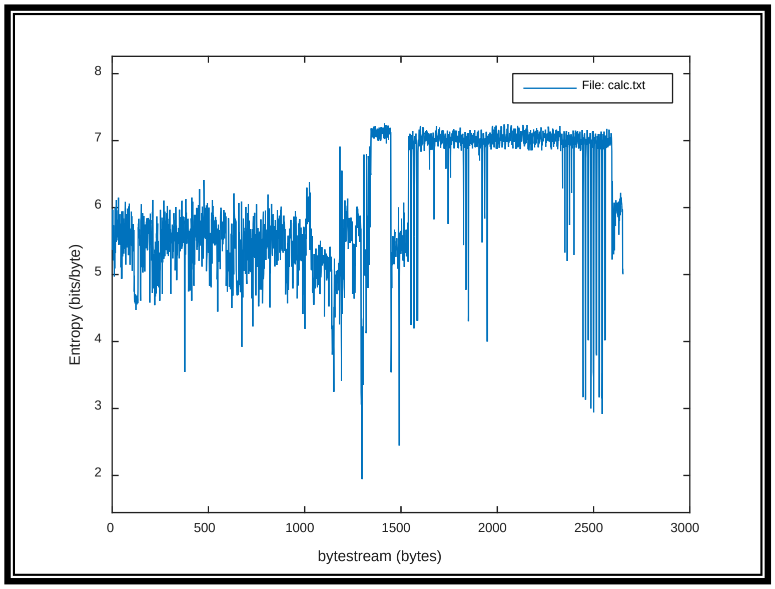

Figure 39. File: 32-bit ‘calc.txt’ Entropy Structure

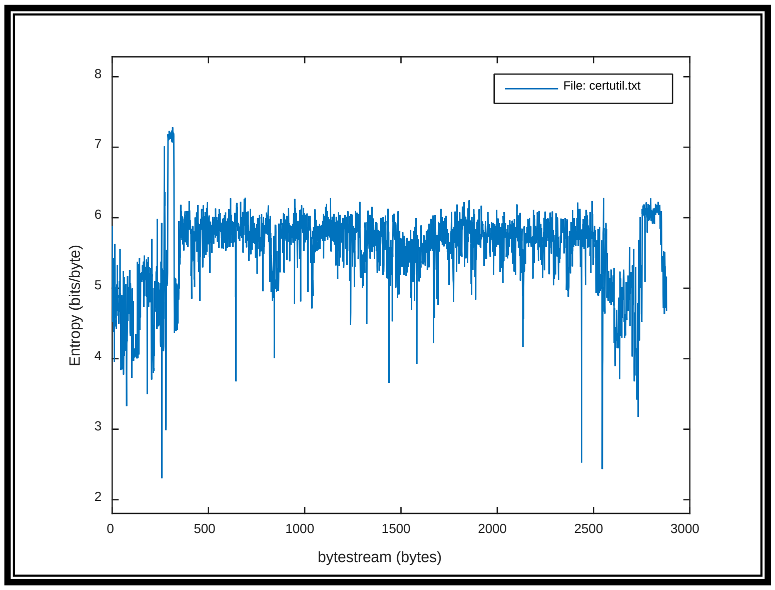

Figure 40. File: 32-bit ‘certutil.txt’ Entropy Structure

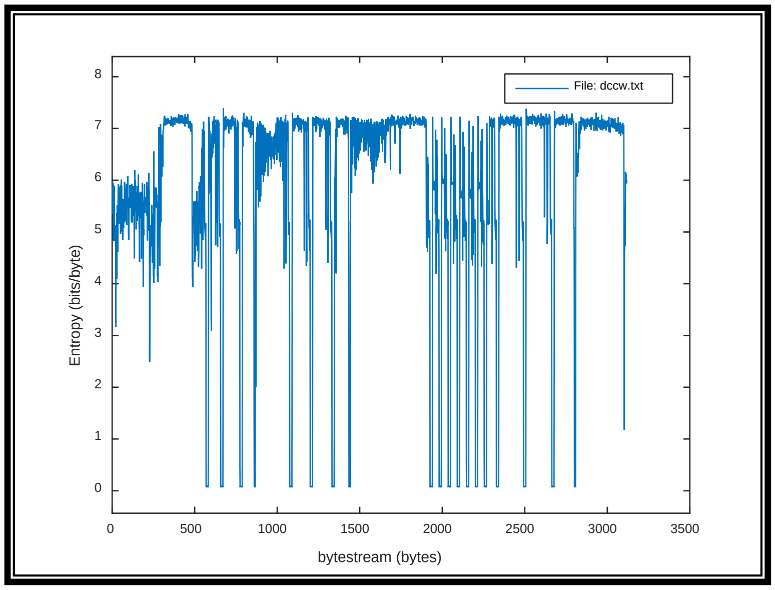

Figure 41. File: 32-bit ‘dccw.txt’ Entropy Structure

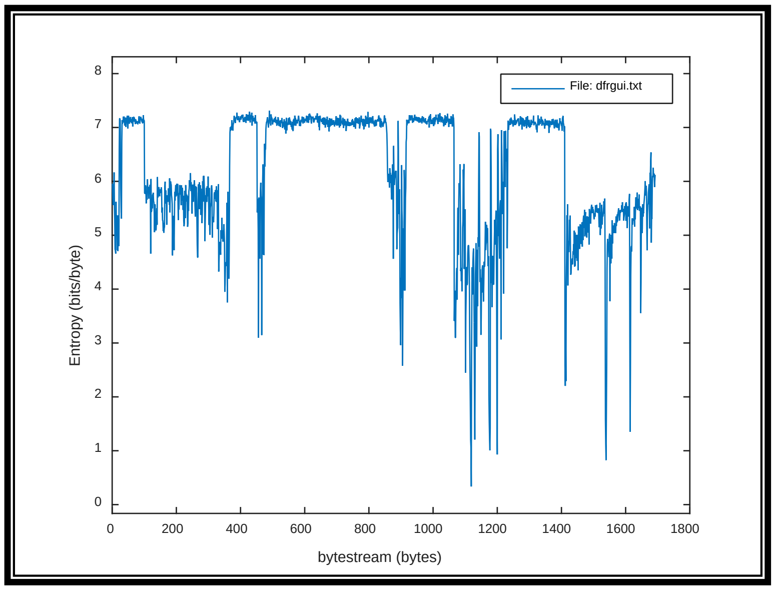

Figure 42. File: 32-bit ‘dfrgui.txt’ Entropy Structure

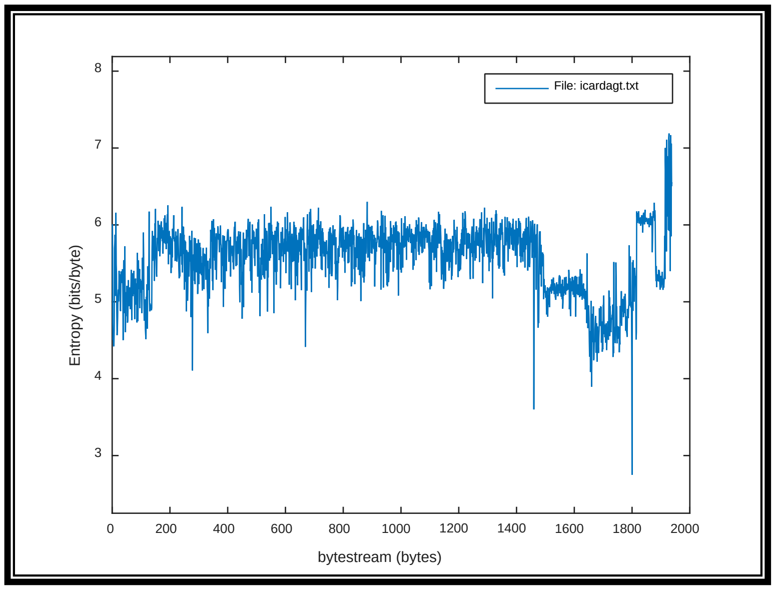

Figure 43. File: 32-bit ‘icardagt.txt’ Entropy Structure

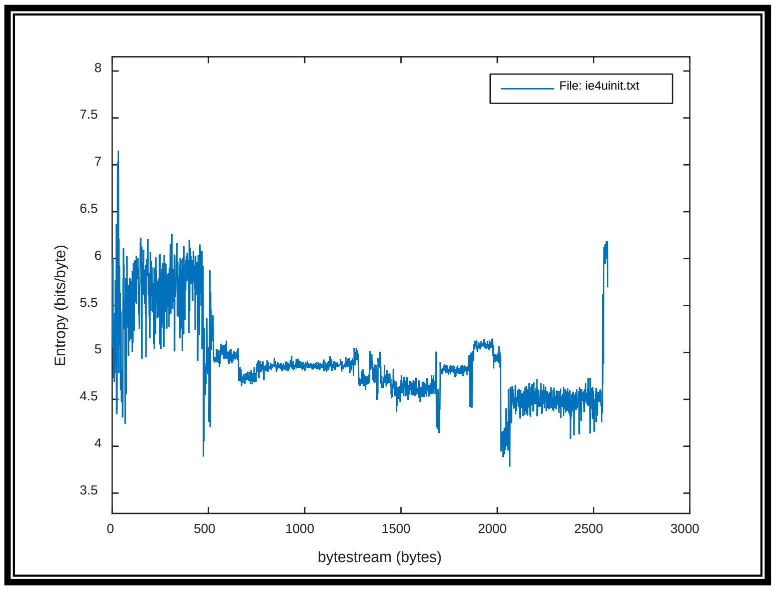

Figure 44. File: 32-bit ‘ie4uinit.txt’ Entropy Structure

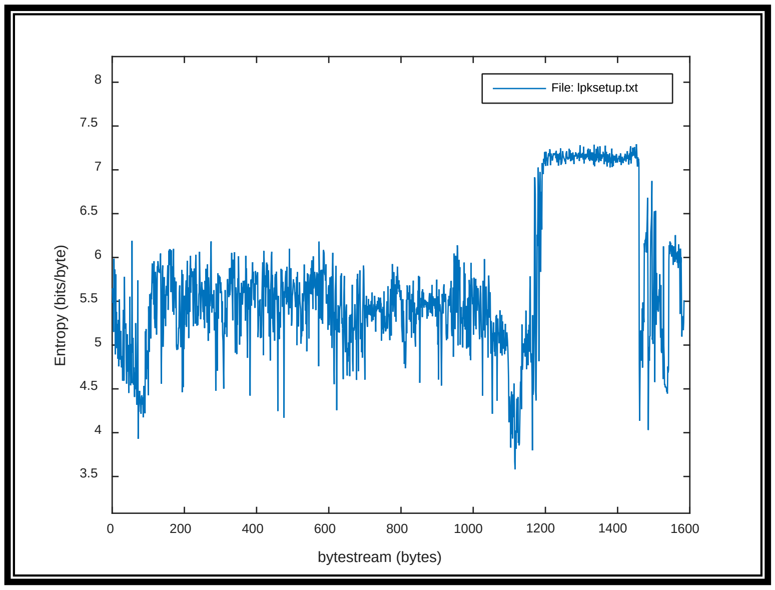

Figure 45. File: 32-bit ‘lpksetup.txt’ Entropy Structure

C. 64-BIT NATIVE PE FILES

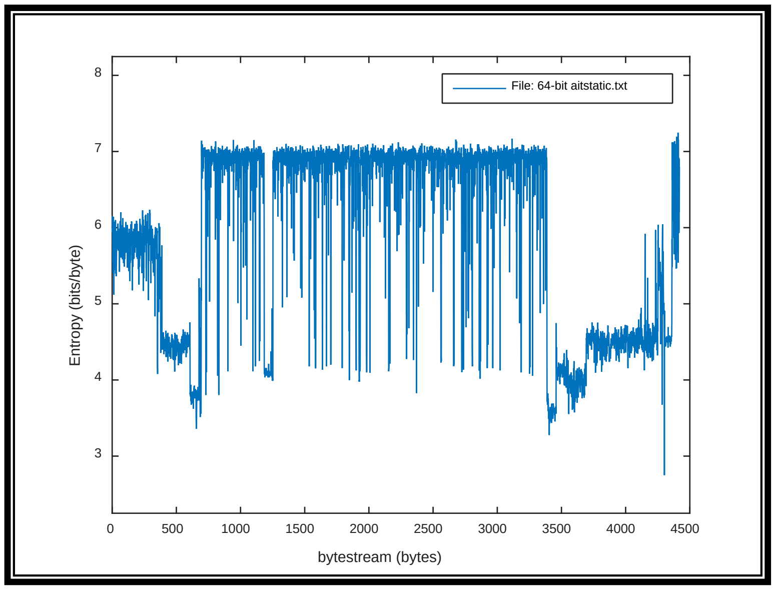

Figure 46. File: 64-bit ‘aitstatic.txt’ Entropy Structure

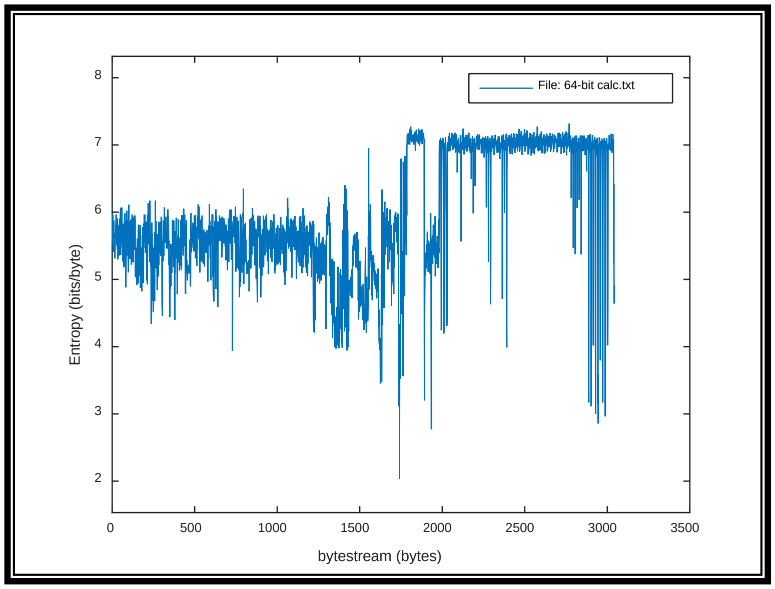

Figure 47. File: 64-bit ‘calc.txt’ Entropy Structure

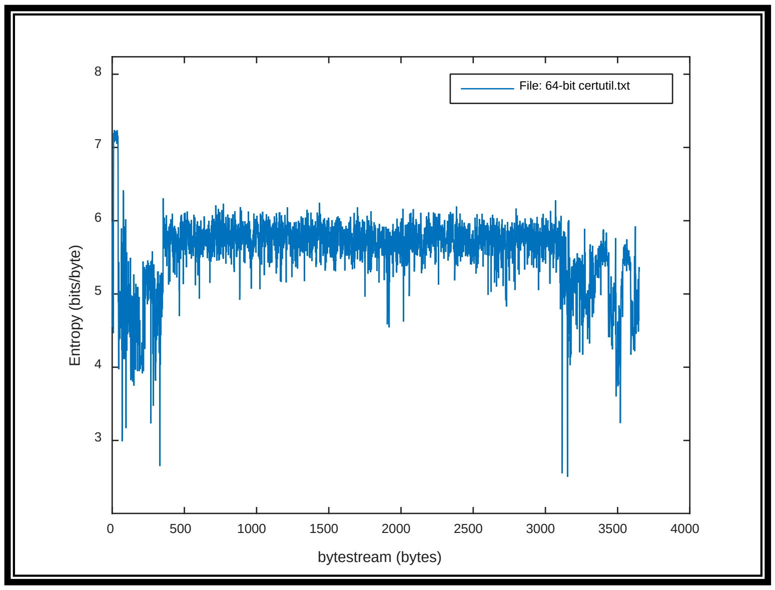

Figure 48. File: 64-bit ‘certutil.txt’ Entropy Structure

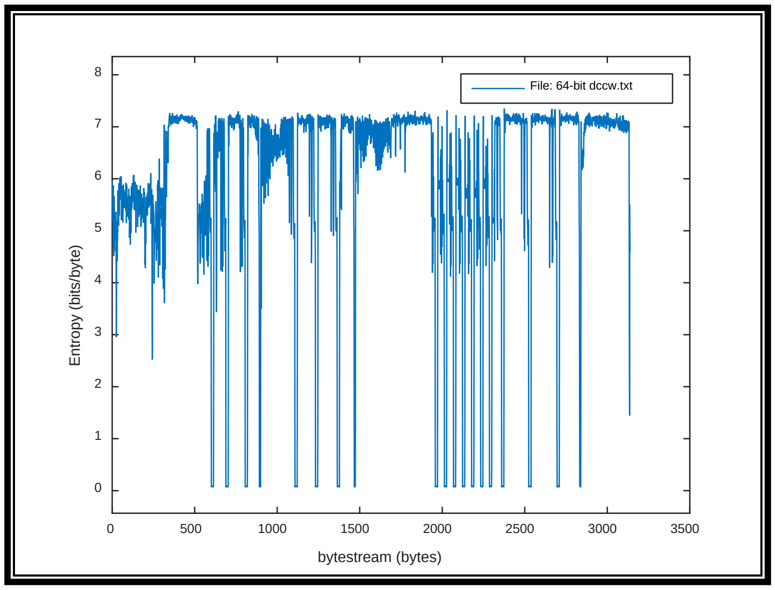

Figure 49. File: 64-bit ‘dccw.txt’ Entropy Structure

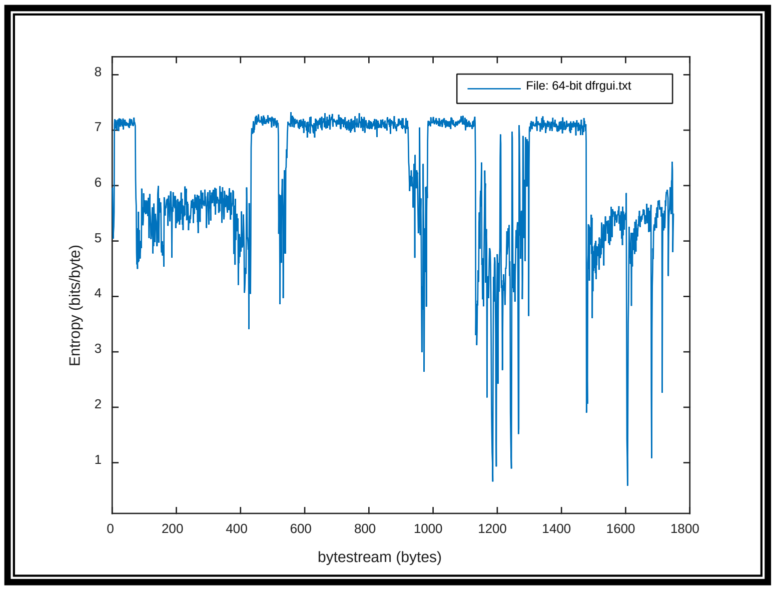

Figure 50. File: 64-bit ‘dfrgui.txt’ Entropy Structure

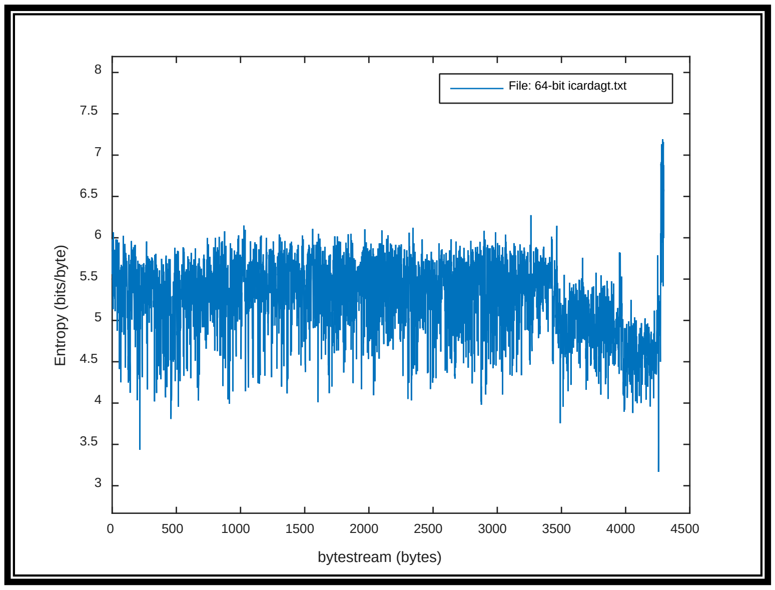

Figure 51. File: 64-bit ‘icardagt.txt’ Entropy Structure

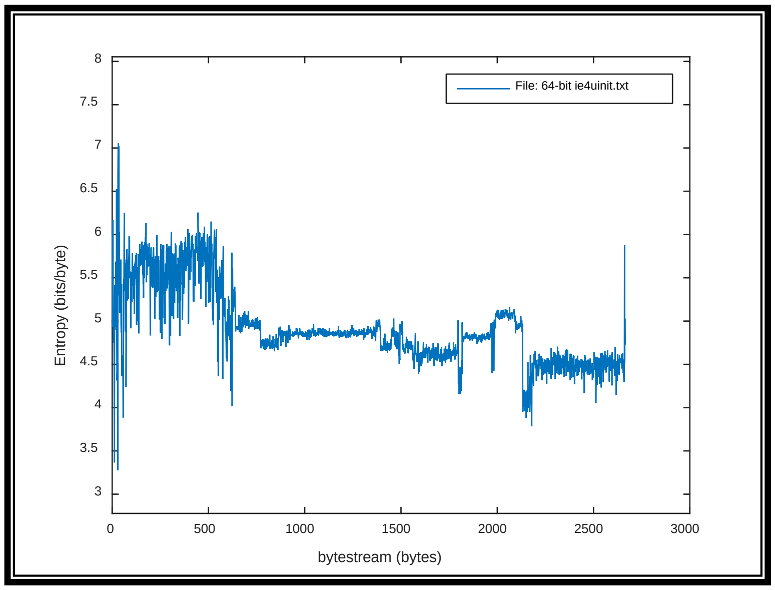

Figure 52. File: 64-bit ‘ie4uinit.txt’ Entropy Structure

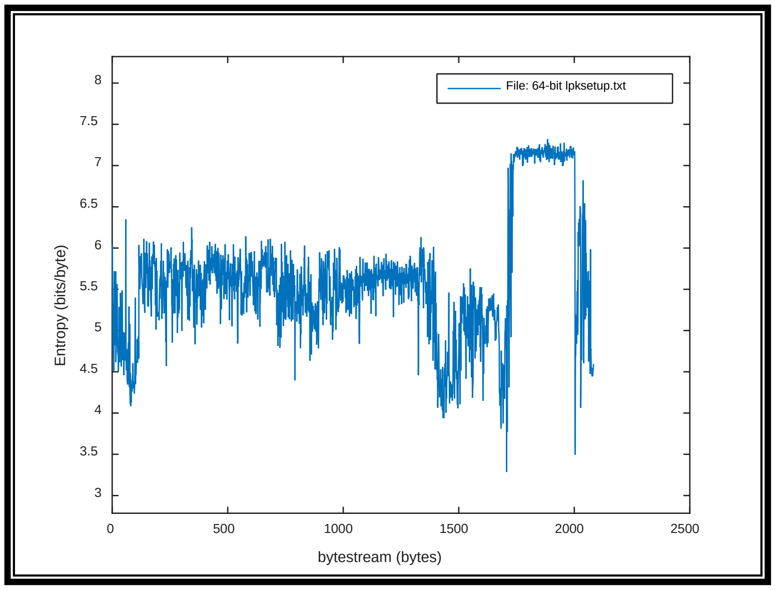

Figure 53. File: 64-bit ‘lpksetup.txt’ Entropy Structure

D. PACKED 32-BIT NATIVE PE FILES

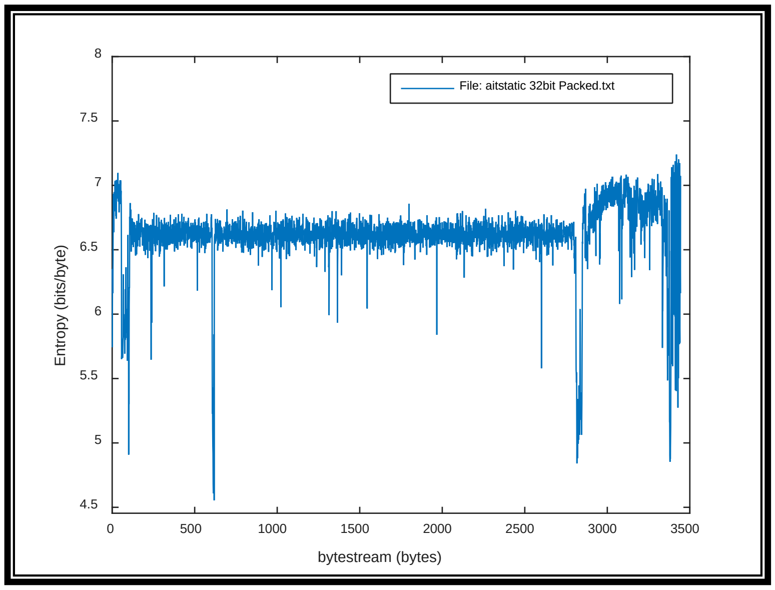

Figure 54. File: 'aitstatic 32bit Packed.txt' Entropy Structure

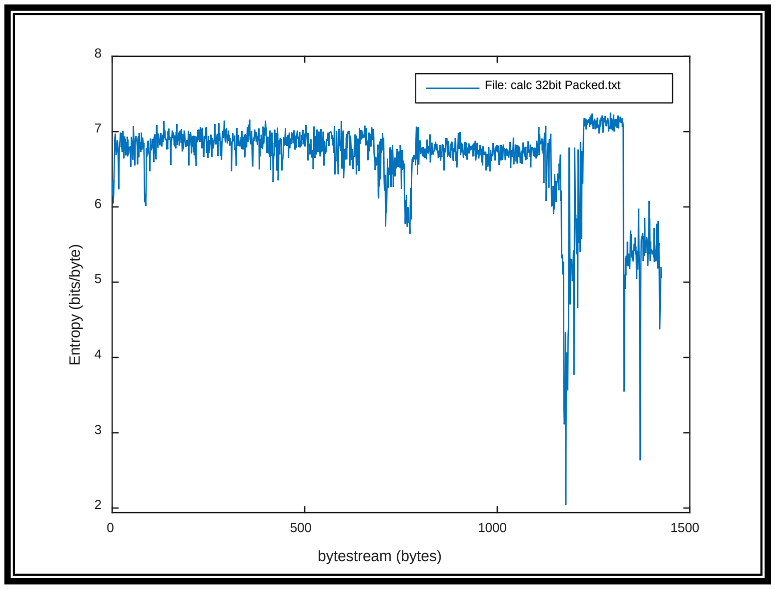

Figure 55. File: 'calc 32bit Packed.txt' Entropy Structure

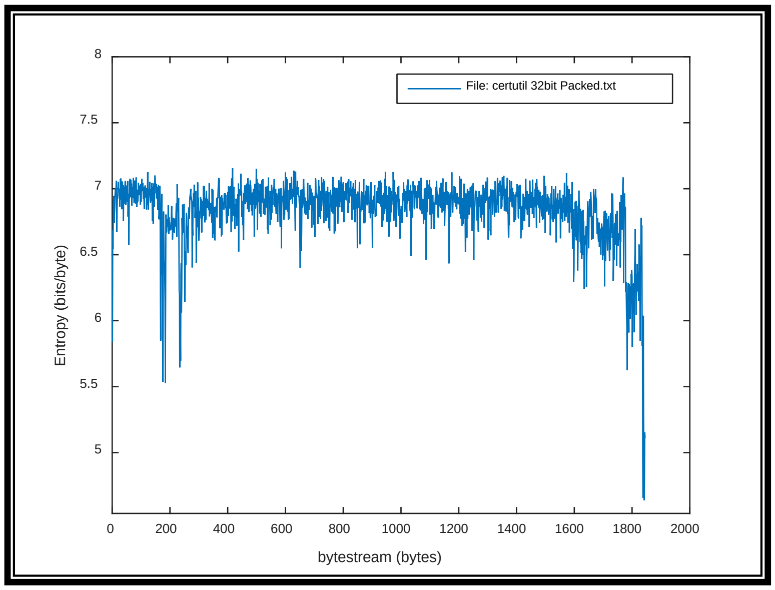

Figure 56. File: 'certutil 32bit Packed.txt' Entropy Structure

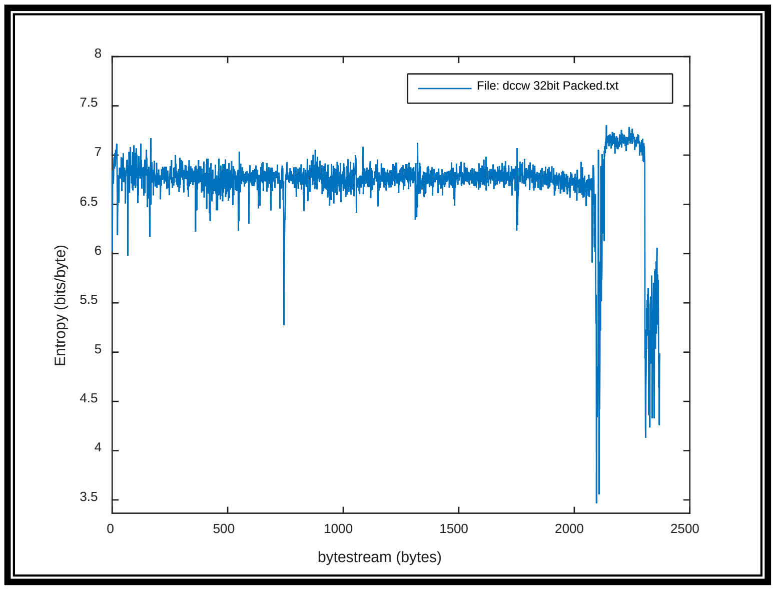

Figure 57. File: 'dccw 32bit Packed.txt' Entropy Structure

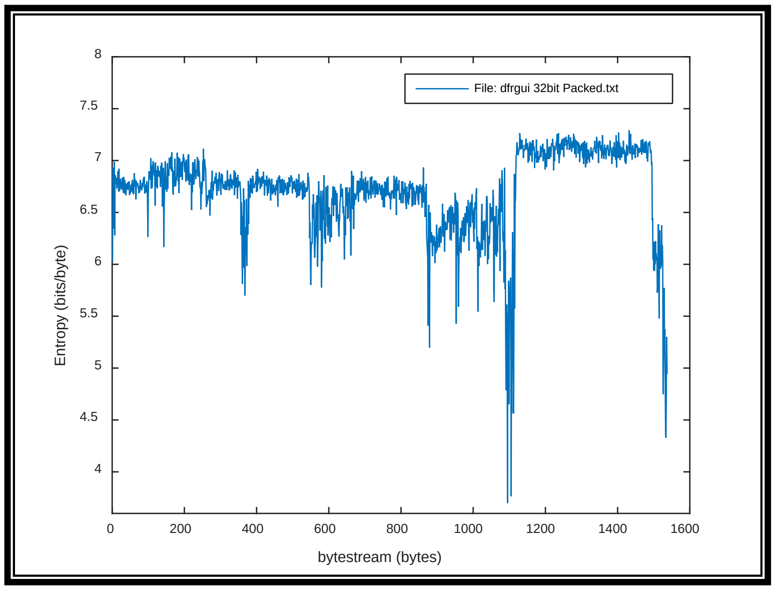

Figure 58. File: ‘dfrgui 32bit Packed.txt’ Entropy Structure

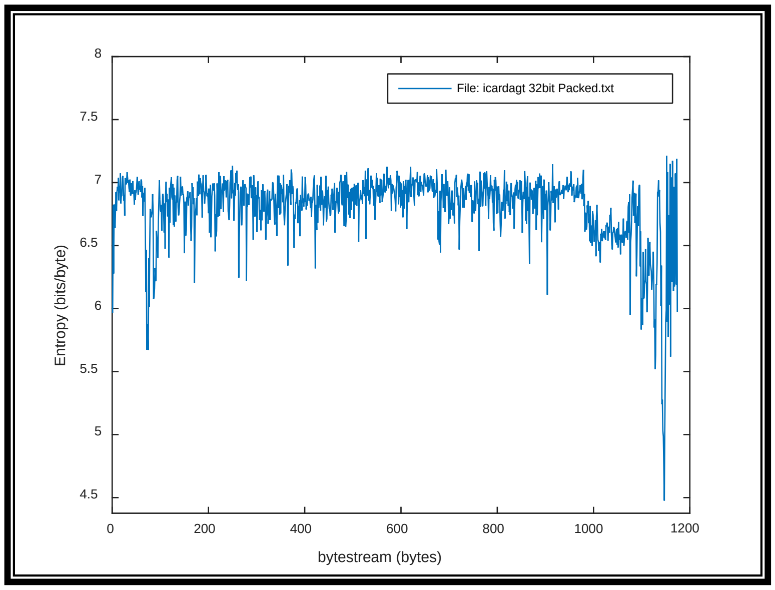

Figure 59. File: ‘icardagt 32bit Packed.txt’ Entropy Structure

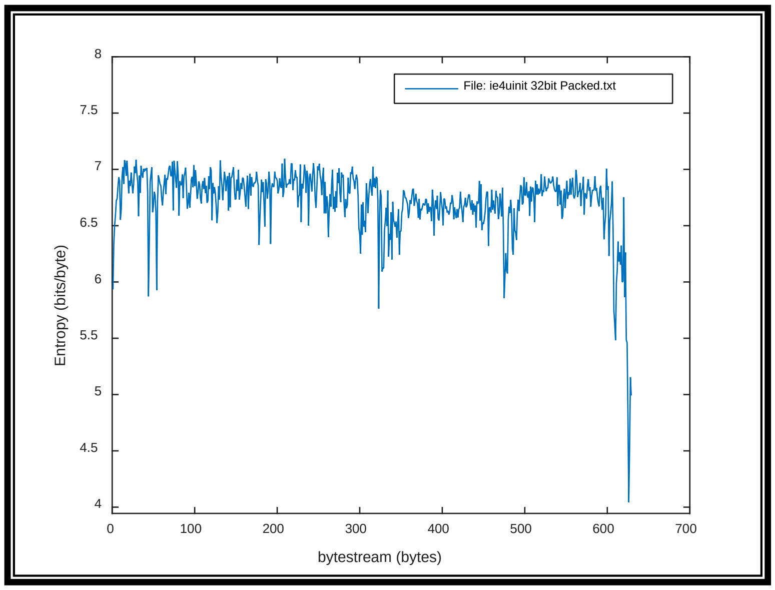

Figure 60. File: ‘ie4uinit 32bit Packed.txt’ Entropy Structure

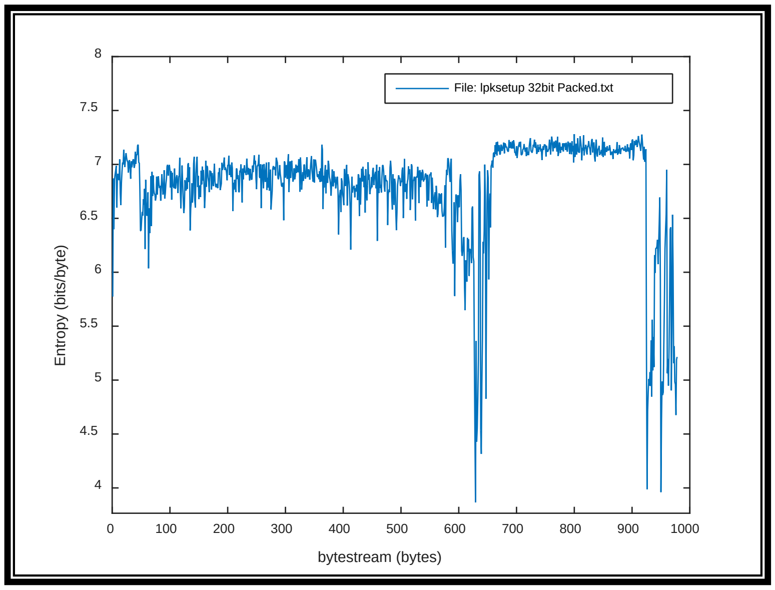

Figure 61. File: ‘lpksetup 32bit Packed.txt’ Entropy Structure

E. PACKED 64-BIT NATIVE PE FILES

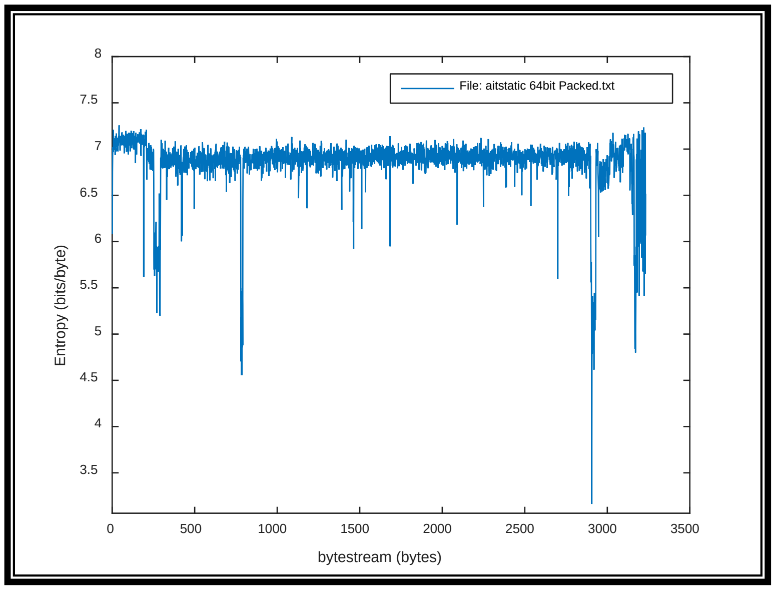

Figure 62. File: 'aitstatic 64bit Packed.txt' Entropy Structure

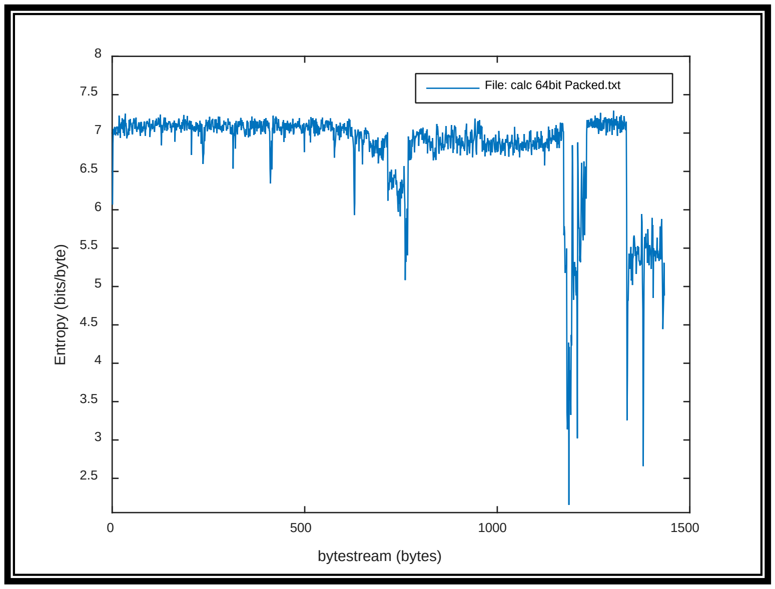

Figure 63. File: 'calc 64bit Packed.txt' Entropy Structure

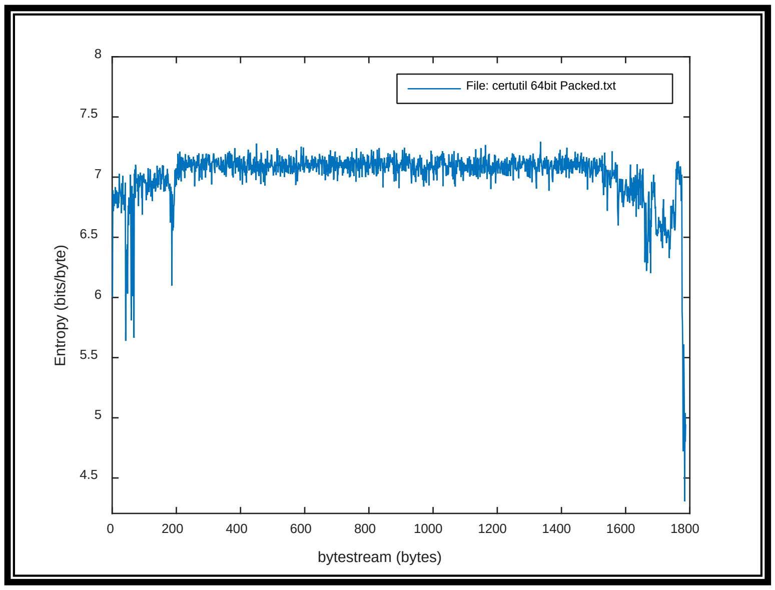

Figure 64. File: ‘certutil 64bit Packed.txt’ Entropy Structure

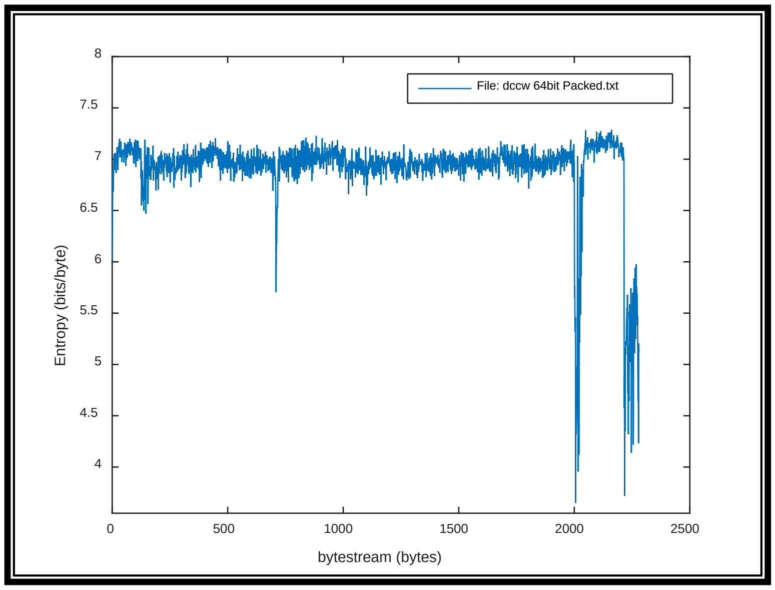

Figure 65. File: ‘dccw 64bit Packed.txt’ Entropy Structure

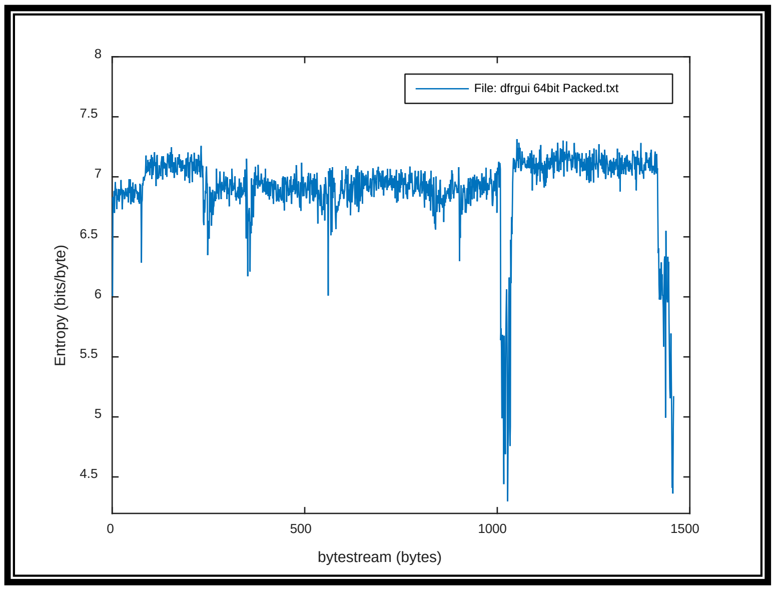

Figure 66. File: ‘dfrgui 64bit Packed.txt’ Entropy Structure

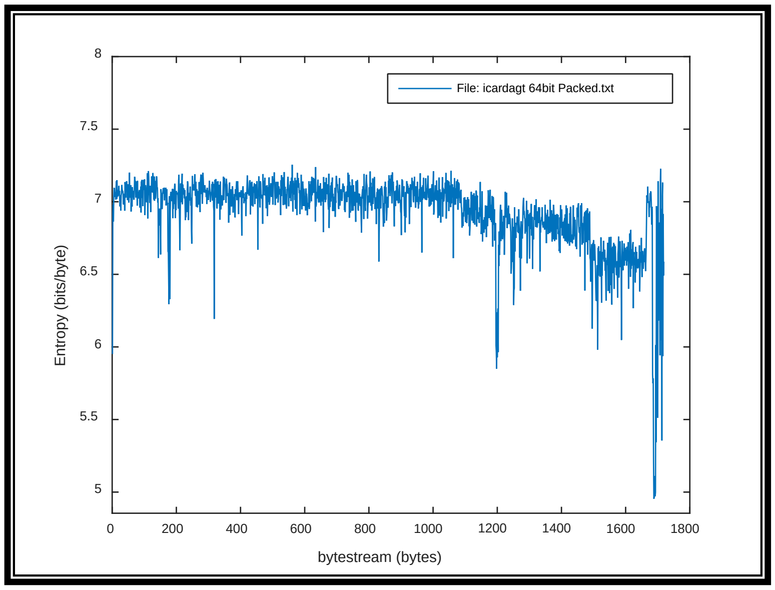

Figure 67. File: ‘icardagt 64bit Packed.txt’ Entropy Structure

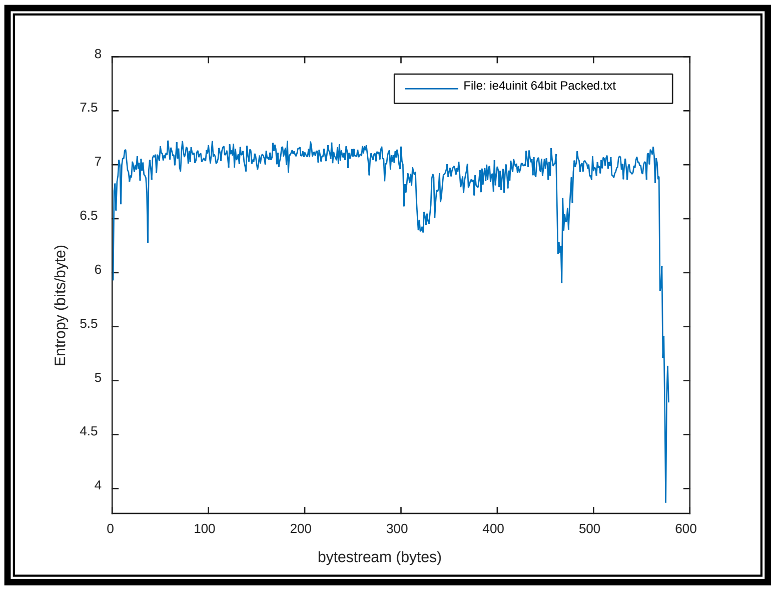

Figure 68. File: ‘ie4uinit 64bit Packed.txt’ Entropy Structure

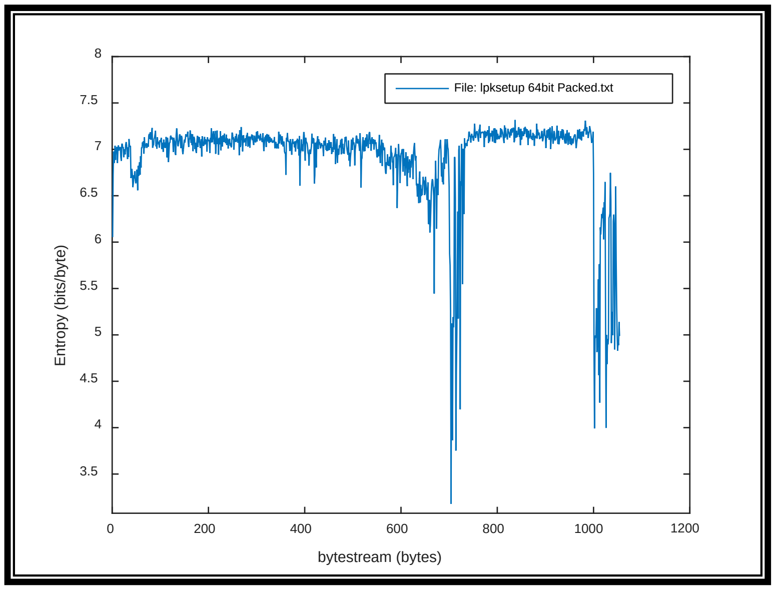

Figure 69. File: ‘lpksetup 64bit Packed.txt’ Entropy Structure

F. ENCRYPTED 32-BIT NATIVE PE FILES

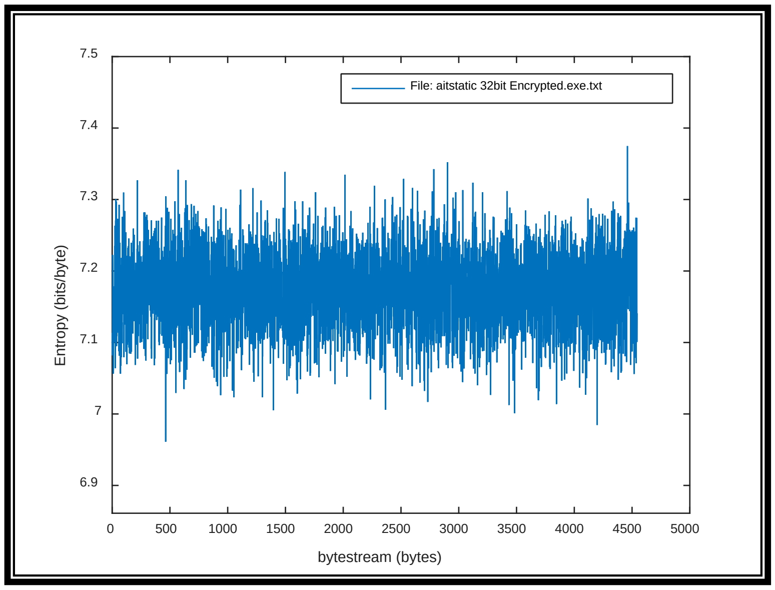

Figure 70. File: 'aitstatic 32bit Encrypted.exe.txt' Entropy Structure

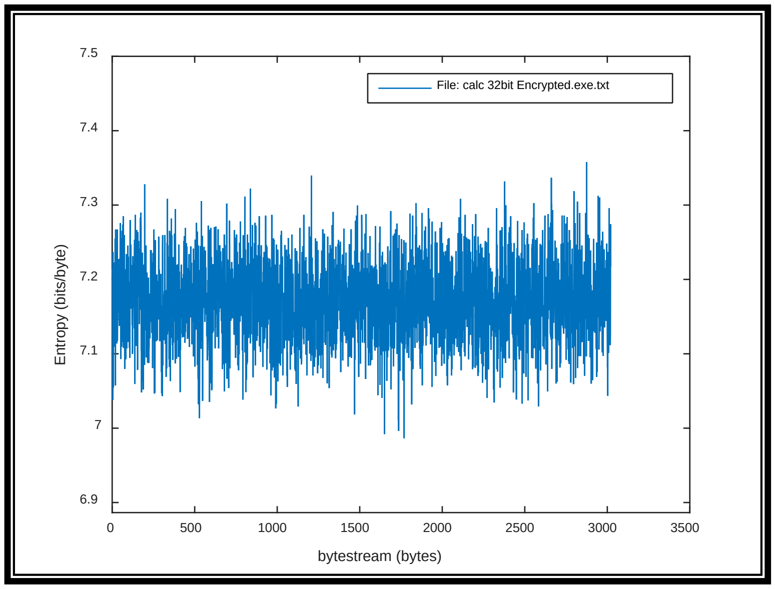

Figure 71. File: 'calc 32bit Encrypted.exe.txt' Entropy Structure

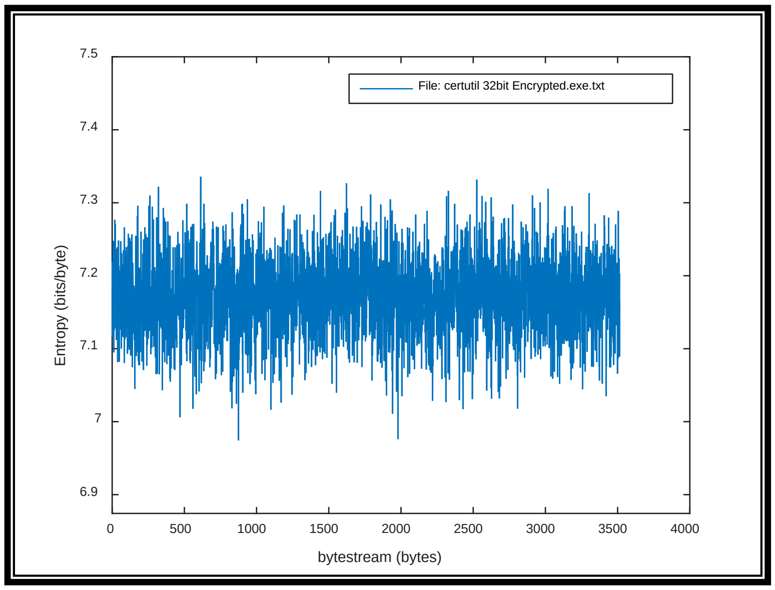

Figure 72. File: ‘certutil 32bit Encrypted.exe.txt’ Entropy Structure

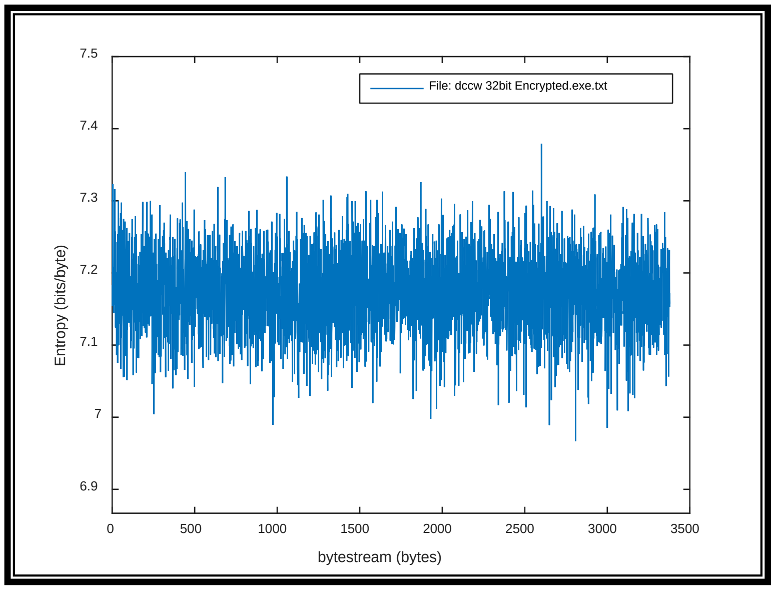

Figure 73. File: ‘dccw 32bit Encrypted.exe.txt’ Entropy Structure

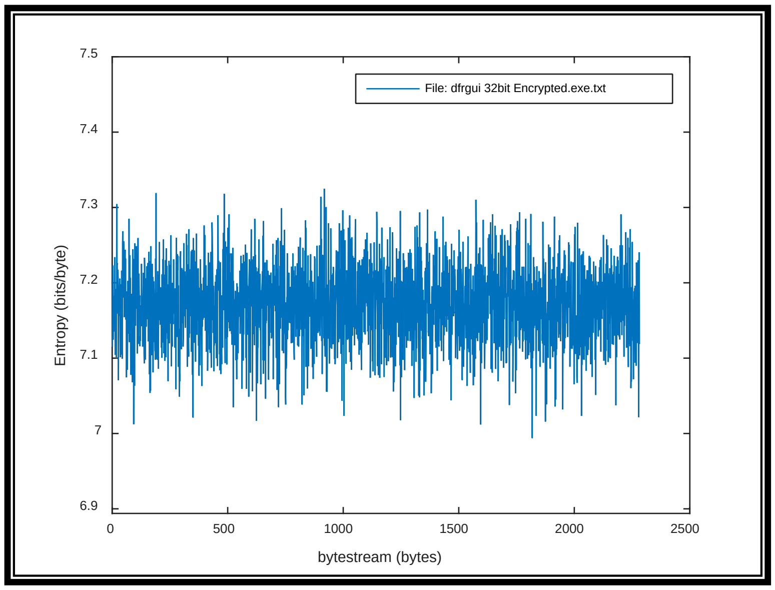

Figure 74. File: ‘dfrgui 32bit Encrypted.exe.txt’ Entropy Structure

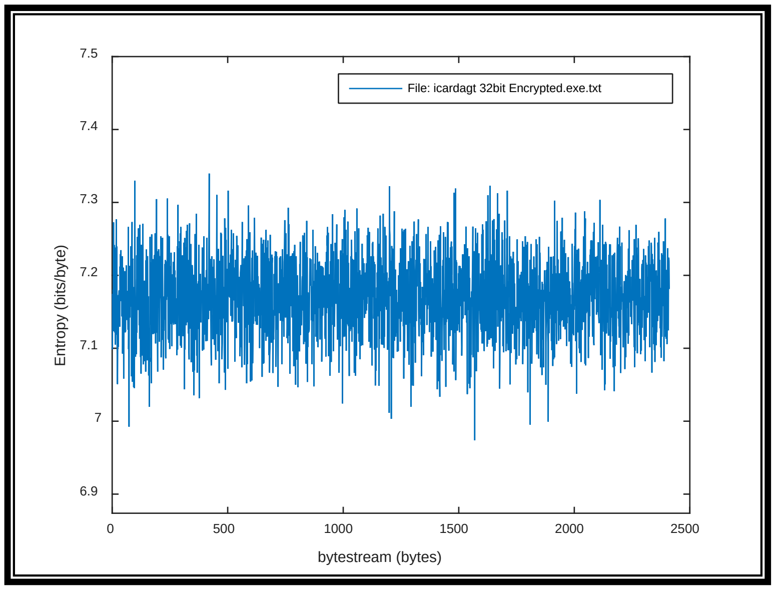

Figure 75. File: ‘icardagt 32bit Encrypted.exe.txt’ Entropy Structure

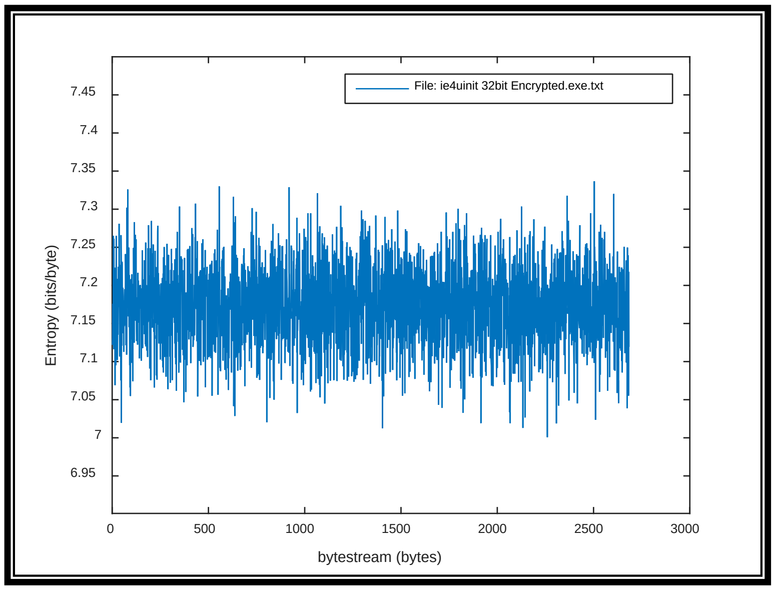

Figure 76. File: ‘ie4uinit 32bit Encrypted.exe.txt’ Entropy Structure

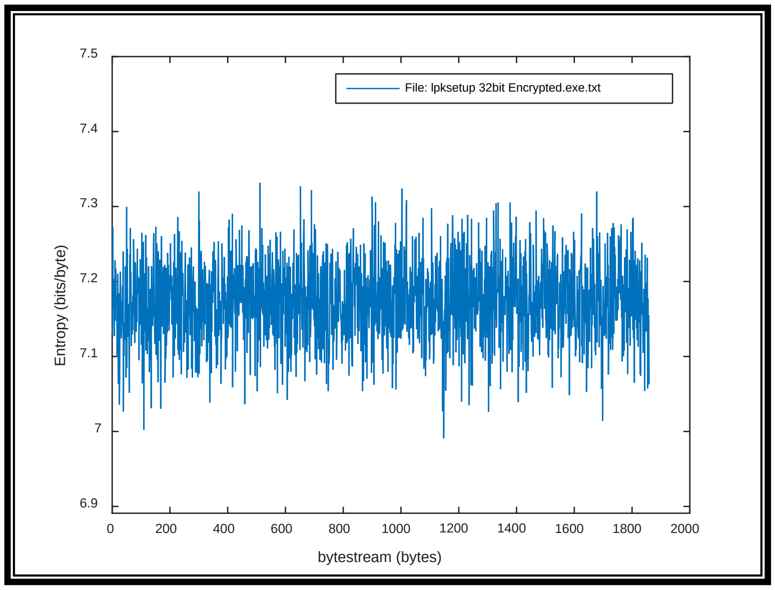

Figure 77. File: ‘lpksetup 32bit Encrypted.exe.txt’ Entropy Structure

G. ENCRYPTED 64-BIT NATIVE PE FILES

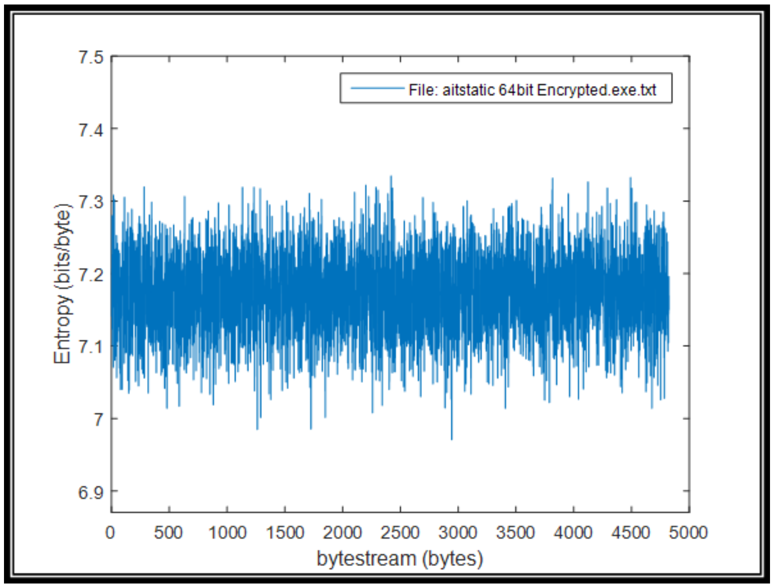

Figure 78. File: ‘aitstatic 64bit Encrypted.exe.txt’ Entropy Structure

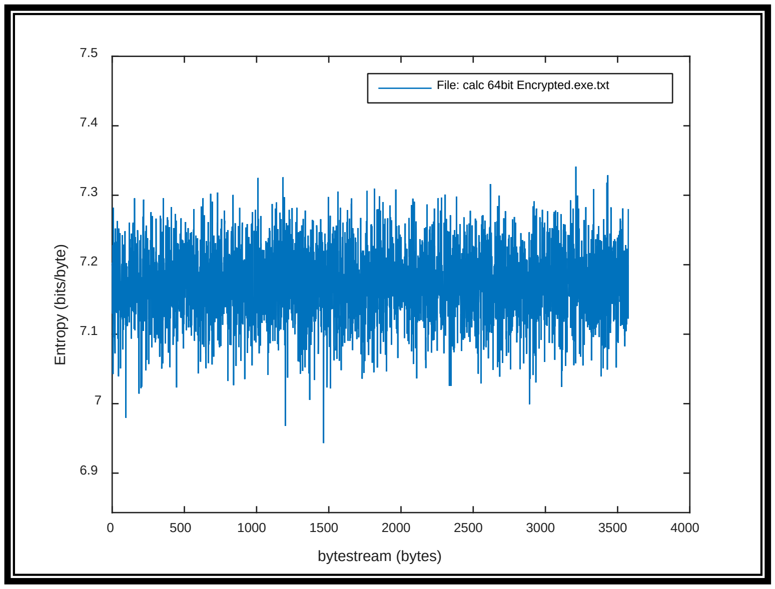

Figure 79. File: ‘calc 64bit Encrypted.exe.txt’ Entropy Structure

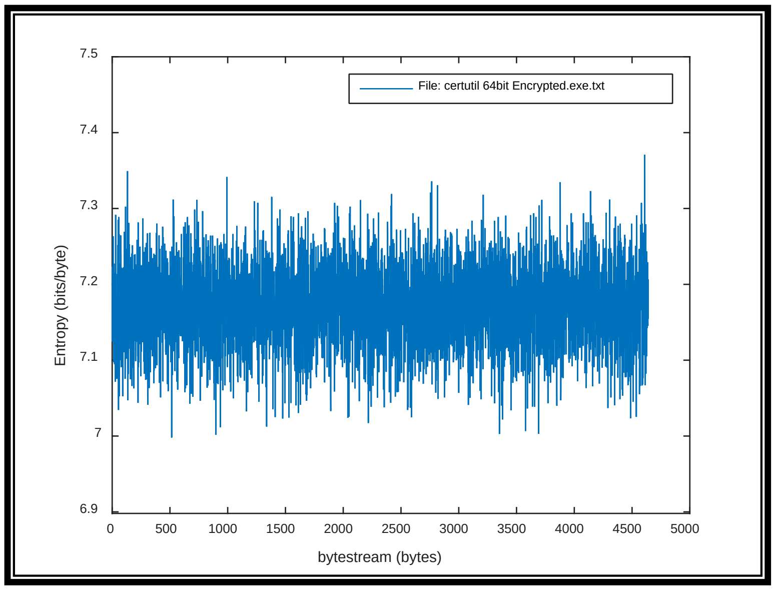

Figure 80. File: ‘certutil 64bit Encrypted.exe.txt’ Entropy Structure

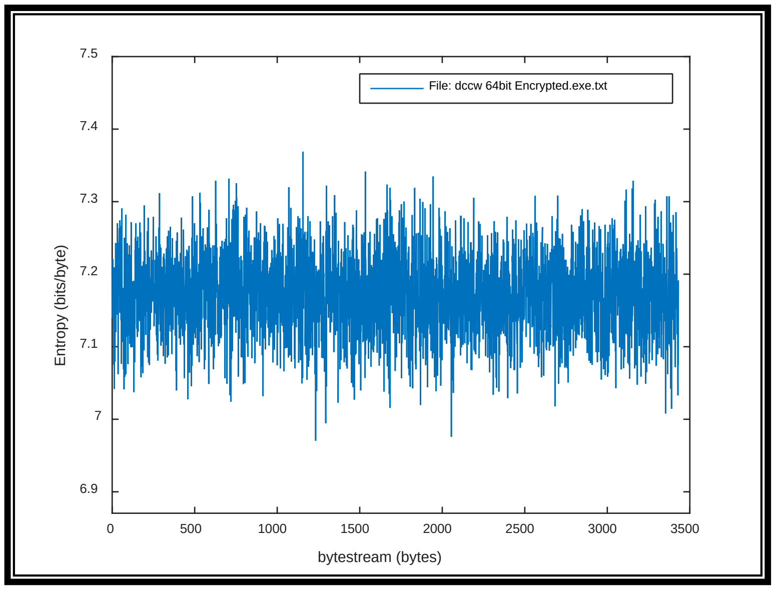

Figure 81. File: ‘dccw 64bit Encrypted.exe.txt’ Entropy Structure

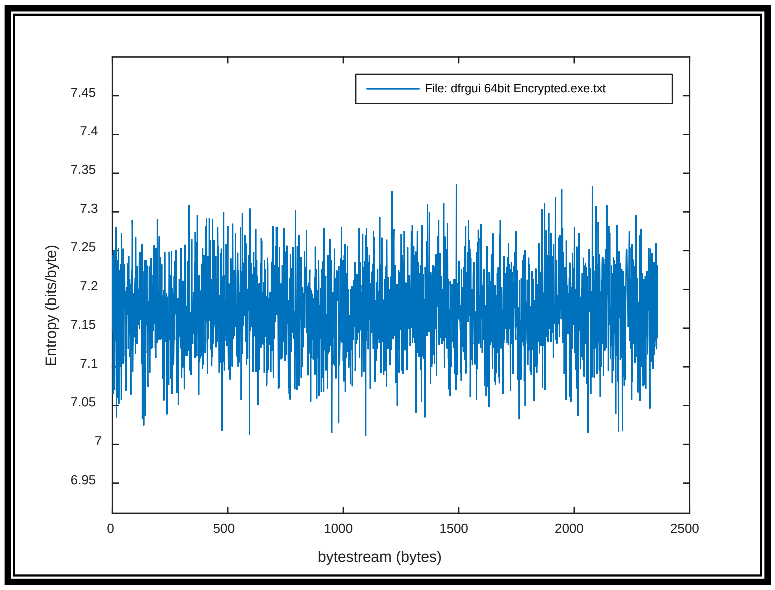

Figure 82. File: 'dfrgui 64bit Encrypted.exe.txt' Entropy Structure

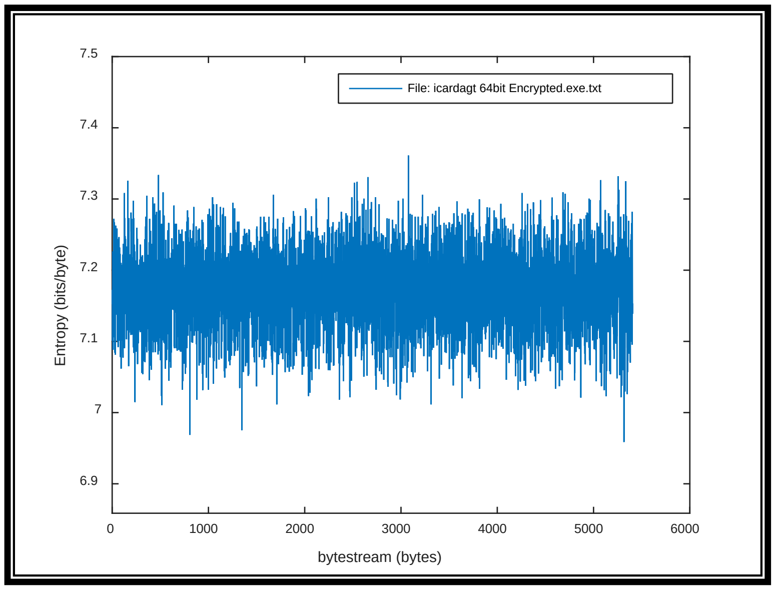

Figure 83. File: 'icardagt 64bit Encrypted.exe.txt' Entropy Structure

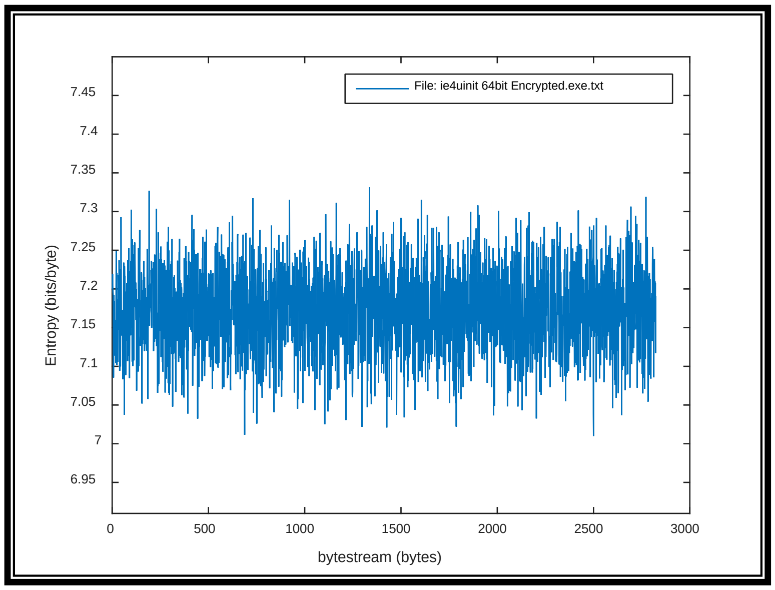

Figure 84. File: ‘ie4uinit 64bit Encrypted.exe.txt’ Entropy Structure

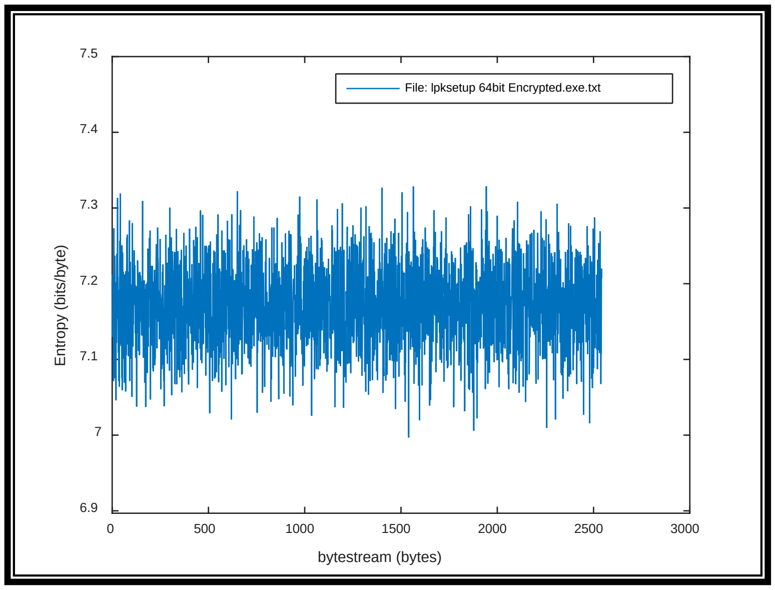

Figure 85. File: ‘lpksetup 64bit Encrypted.exe.txt’ Entropy Structure

LIST OF REFERENCES

[1] L. Nataraj, “A signal processing approach to malware analysis,” Ph.D. dissertation, Dept. Elect. and Comp. Eng., Univ. California, Santa Barbara, CA, 2015.

[2] B. Donabelle, R. M. Low, and M. Stamp, “Structural entropy and metamorphic malware,” *Journal of Computer Virology and Hacking Techniques*, vol. 9, no. 4. pp. 179–192, 2013.

[3] G. Conti, S. Bratus, A. Shubina, B. Sangster, R. Ragsdale, M. Supan, A. Lichtenberg, and R. Perez-Alemany, “Automated mapping of large binary objects using primitive fragment type classification,” *Digital Investigation*, vol. 7, no. Supplement 1, pp. S3-S12, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287610000290

[4] R. Lyda and J. Hamrock, “Using entropy analysis to find encrypted and packed malware,” *IEEE Security and Privacy*, vol. 5, no. 2, pp. 40–45, Apr. 2007.

[5] B. Jochheim, “On the automatic detection of embedded malicious binary code using signal processing techniques,” Project report, Dept. Info. and Elect. Eng., Univ. Hamburg University of Applied Science, 2012.

[6] S. Fitzgerald, G. Mathews, C. Morris, and O. Zhulyn, “Using nlp techniques for file fragment classification,” *Digital Investigation*, vol. 9, no. Supplement 1, pp. S44–S49, 2012.

[7] G. Jeong, E. Choo, J. Lee, M. Bat-Erdene, and H. Lee, “Generic unpacking using entropy analysis,” *IEEE Xplore Proceedings of 5th International Conference on Malicious and Unwanted Software*, pp. 98–105, 2010.

[8] I. Sorokin, “Comparing files using structural entropy,” *Journal in Computer Virology*, vol. 7, no. 4, pp. 259–265, 2011.

[9] M. Wojnowicz, G. Chisholm, and M. Wolff, “Suspiciously structured entropy: Wavelet decomposition of software entropy reveals symptoms of malware in the energy spectrum,” *Proceedings of the Twenty-Ninth International Florida Artificial Intelligence Research Society Conference*, pp. 288–293, 2016, Irvine, CA: Cylance, Inc.

[10] M. Wojnowicz, G. Chisholm, and M. Wolff, “Structural entropy analysis for automated malware classification,” *RSA Conference 2015*. [Online] Available: https://www.rsaconference.com/writable/presentations/file_upload/hta-t09-structural_entropy_analysis_for_automated_malware_classification_final_v2.pdf

[11] Z. Chen, P.C. Ivanov, K. Hun, and H.E. Stanley, “Effect of nonstationarities on detrended fluctuation analysis,” *Physical Review E*, vol. 65(4), no. 041107, 2002. [Online]. Available: https://journals.aps.org/pre/abstract/10.1103/PhysRevE.65 .041107

[12] *Wavelet Toolbox For Use with Matlab, User Guide*, Apple Hill, MA: The MathWorks, Inc.

[13] S. Naval, V. Laxmi, M. S. Gaur, and Vinod P, “An efficient block-discriminant classification of packed Malware,” *Sadhana*, vol. 40, no. 5, pp 1435–1456, doi: 10.1007/s12046-015-0399-x. [Online]. Available: https://link.springer.com/ article/10.1007/s12046-015-0399-x

[14] P. Thulasiraman, Course Notes, “Principles of reverse engineering: reverse engineering malware,” , EC3740, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 2017.

[15] W. Stallings, *Cryptography and Network Security*, 7th ed. Hoboken, NJ: Pearson Education, Inc., 2017.

[16] M. Fargues, Course Notes, “I-Random concepts: Review & applications to signal & information processing,” EC3410, Dept. of Elec. and Comp. Eng., Naval Postgraduate School, Monterey, CA, 2016.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California