Digital Investigation xxx (2013) 1–13

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Approaches to the classification of high entropy file fragments

Philip Penrose, Richard Macfarlane, William J. Buchanan*

Edinburgh Napier University, Edinburgh, United Kingdom

ARTICLE INFO

Article history:
Received 6 May 2013
Received in revised form 21 August 2013
Accepted 24 August 2013

Keywords:
Digital forensics
File fragments
Encrpyted files
File forensics
Encryption detection

ABSTRACT

In this paper we propose novel approaches to the problem of classifying high entropy file fragments. Although classification of file fragments is central to the science of Digital Forensics, high entropy types have been regarded as a problem. Roussev and Garfinkel (2009) argue that existing methods will not work on high entropy fragments because they have no discernible patterns to exploit. We propose two methods that do not rely on such patterns. The NIST statistical test suite is used to detect randomness in 4 KiB fragments. These test results were analysed using an Artificial Neural Network (ANN). Optimum results were 91% and 82% correct classification rates for encrypted and compressed fragments respectively. We also use the compressibility of a fragment as a measure of its randomness. Correct classification was 76% and 70% for encrypted and compressed fragments respectively. We show that newer more efficient compression formats are more difficult to classify. We have used subsets of the publicly available 'GovDocs1 Million File Corpus' so that any future research may make valid comparisons with the results obtained here.

1. Introduction

In Garfinkel (2010a), Garfinkel claims that much of the progress made in digital forensic tools over the last decade is becoming irrelevant. These tools were designed to help forensic examiners to find evidence, usually from a relatively low capacity hard disk drive, and do not scale to the capacity of digital storage devices commonly available today.

In addition, time may be a critical factor at the initial stage of a forensic investigation. Giordano (Giordano and Macaig, 2002) describes the importance in military and anti-terrorist activities for forensic examiners to get a quick overview of the contents of seized digital media, whereas Rogers and Goldman (2006) relate how criminal investigations may depend on the quick analysis of digital evidence on-site.

Current digital forensic techniques and tools are not suited to such scenarios. They are aimed mainly at '*post crime*' analysis of digital media. In a time critical situation a forensic investigator needs a data analysis tool that can quickly give a summary of the storage device contents. This will inform the decision by the investigator on whether the media deserves prioritisation for deeper analysis or not.

Garfinkel (2010b) puts forward the hypothesis that the content of digital media can be predicted by identifying the content of a number of randomly chosen sectors. The hypothesis is justified by randomly sampling 2000 digital storage devices to create a '*forensic inventory*' of each. It was found that, in general, sampled data gave similar statistics to the media as a whole.

Using this methodology to quickly produce a summary of storage device contents requires that data fragments be identified accurately. Research in the area of fragment classification has advanced over the last few years, so that many standard file types can be identified accurately from a fragment. Methods of classification fall into three broad categories: direct comparison of byte frequency distributions, statistical analysis of byte frequency distributions and specialised approaches which rely on knowledge of particular file formats. However none of these methods has been successful in the classification of high entropy file fragments such as encrypted or compressed data.

* Corresponding author. Tel.: +44 0 1314552759.
E-mail address: w.buchanan@napier.ac.uk (W.J. Buchanan).

1742-2876/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.diin.2013.08.004

1.1. Problem

Recent research has found that although classification of file fragments of many common file types can be done with high accuracy, the classification of high entropy file fragments is difficult and to date accuracy has been poor (Fitzgerald et al., 2012; Li et al., 2010). Indeed Roussev (Roussev and Garfinkel, 2009) suggests that it is not possible using the current statistical and machine learning techniques to differentiate such fragments.

The entropy of a file fragment measures the amount of randomness or disorder in the data within it. Compressed and encrypted files have little pattern or order. A file is compressed by representing any repeating patterns of data with a shorter code. A well compressed file should have no apparent patterns remaining in it otherwise it could be compressed further. An encrypted file should have no patterns otherwise it would be vulnerable to cryptanalysis. Thus these types are classified as high entropy.

Garfinkel (Garfinkel et al., 2010) states that the classification of high entropy file fragments is in its infancy and needs further research and it is suggested that future works should investigate methods to improve classification performance on such fragments. To date no such investigations have been done.

2. Literature review

2.1. Introduction

This chapter presents the current research into file fragment classification. There is a theme running through the research that results in classifying high entropy fragment types has been universally poor. Many investigations have simply excluded these fragment types from their corpora. Roussev (Roussev and Garfinkel, 2009, p.11) questions whether current machine learning or statistical techniques applied to file fragment classification can ever distinguish between these types since these fragments have no discernible patterns to exploit. These findings lead us to develop our approaches to the problem.

It can be observed that there has been a trend towards specialised approaches for each file type. Analysis of byte frequency distributions has often proved insufficient to classify fragment types, and the unique characteristics of particular file formats have increased recognition accuracy. It becomes apparent that in many cases neither the digital corpora used nor the software developed is publicly available. It is therefore not possible to validate the research nor do a direct comparison against the methods which we develop. In addition, many results have been derived from small sample sets and thus the results may not be universally applicable. These observations lead us to design our investigation in a manner which will avoid such criticisms.

2.2. File fragment identification

The idea of using examination of the byte frequency distribution (BFD) of a file to identify a file type was introduced by McDaniel (McDaniel and Heydari, 2003). The BFD is simply a count of the frequency of occurrence of each possible byte value (0–255) giving a 256 element vector. Several of these vectors are averaged by adding corresponding elements and dividing by the number of vectors to give an average vector or centroid. For each byte-value the correlation between byte frequency in each file is also recorded. These vectors were taken to be characteristic of that file type and termed the '*fingerprint*' for that type. The BFD of an unknown file type is subtracted from the fingerprint and the differences averaged to give a measure of closeness. Another fingerprint was developed using a byte frequency cross-correlation (BFC) which measured the average difference in byte pair frequencies. This was a specialised approach which should target certain file types – HTML files, for example, where the characters '<' and '>' occur in pairs regularly. We shall see that developing such specialised approaches to identify file fragments when byte frequencies of different types are similar is a common occurrence through the research corpus. Average classification accuracy is poor with BFD at around 27% and BFC at around 46%. This result is actually poorer than it appears since whole files were used instead of file fragments and so the file headers which contain magic numbers were included. When file headers were considered by themselves they reported an accuracy of over 90% as would be expected. The corpus consisted of 120 test files with four of each of the 30 types considered. There is no indication as to the source of the test files and no encrypted files were included. They noted that the ZIP file format had '*a low assurance level*' and that perhaps other classification methods might be needed to improve the accuracy for this type.

Li (Li et al., 2005) extended the work of McDaniel and Heydari (2003) and developed the methodology that had been introduced by Wang (Wang and Stolfo, 2004) for a payload based intrusion detection system. They used the term n-gram to refer to a collection of n consecutive bytes from a byte stream and based their analysis on 1-g. The terms '*fileprint*' and '*centroid*' were used interchangeably to refer to a pair of vectors. One contained byte-frequencies averaged over a set of similar files from the sample, and, in the other, the variance of these frequencies. In addition, they created a multi-centroid approach by creating several such centroids for each file type since '*files with the same file extension do not always have a distribution similar enough to be represented by a single model*'. The metric used to calculate the distance of an unknown sample from each centroid was that used in Wang and Stolfo (2004). It was proposed originally for computational efficiency in the scenario of a high bandwidth networked environment but this simplification meant that it was no longer suitable as a metric. It was termed a simplified version of the Mahalanobis distance and given as:

$$d(\mathbf{x},\mathbf{y}) = \frac{\sum_0^{n-1}(|x_i - y_i|)}{(\sigma_i + \alpha)}$$

where

- x_i and y_i are the centroid and unknown sample byte frequencies respectively.

- σ_i is the centroid standard deviation for that byte value.
- α is a small positive value added to avoid possible division by zero.

To be classified as a metric, a function should satisfy the following conditions for all points **x**, **y**, **z** in the space (Copson, 1988, p. 21):

1. $d(\mathbf{x},\mathbf{y}) \geq 0$
2. $d(\mathbf{x},\mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$
3. $d(\mathbf{x},\mathbf{y}) = d(\mathbf{y},\mathbf{x})$
4. $d(\mathbf{x},\mathbf{z}) = d(\mathbf{x},\mathbf{y}) + d(\mathbf{y},\mathbf{z})$

In the simplified metric we can ignore the denominator since it is simply a scaling factor. We have:

$$d(\mathbf{x},\mathbf{y}) = \sum_{0}^{n-1}(|x_i - y_i|)$$

Consider the vectors $\mathbf{x} = (4,7,3)$ and $\mathbf{y} = (1,1,2)$.
Using this metric then:

$$d(x,y) = |(4-1)| + |(7-1)| + |(3-2)| = 10$$

If $\mathbf{z} = (0, 0, 0)$ then:

$$d(x,y) = 14 \text{ and } d(z,y) = 4$$

Hence:

$$d(x,y) \neq d(x,z) + d(z,y)$$

Thus their '*simplified metric*' does not meet the criteria for being a metric. Xing (Xing et al., 2003) describes how learning algorithms depend critically on a good metric being chosen. We find that many of the techniques considered below which use machine learning to create centroids give no justification for the metrics used. Unfortunately only 8 file types were considered, and no encrypted or compressed files were used, although they noted that all compressed files may have a similar distribution. To create the test corpus 100 files of each type were collected from the internet using a search on Google.

To create their sample set, the first 20, 200, 500 and 1000 bytes from a file were taken. As expected, since these truncated files all contain the file header, the results were good. The average classification accuracy was around 99% for the truncated files with just the first 20 bytes of the file i.e. the file header. As the file size increased, accuracy decreased. Accuracy was worst when the whole file, rather than a truncated segment, was used. This could be explained by the fact that with just the first 20 bytes of a file the method reduces to the '*magic numbers*' solution referred to previously. As the file size increases the influence of these first magic numbers is diluted and hence accuracy decreases.

The method named '*Oscar*' was introduced by Karresand (Karresand and Shahmehri, 2006a) using the same BFD vectors as (McDaniel and Heydari, 2003) to create centroids. This was soon extended to increase the accuracy of JPEG detection by introducing Rate of Change (RoC) of consecutive bytes (Karresand and Shahmehri, 2006b). The rate of change is maximum for pairs of bytes 0xFF followed by 0x00. As explained earlier the frequency of this byte pair is a unique marker for JPEG files.

They avoid the criticism made of Li et al. (2005) by using a weighted Euclidian metric:

$$d(\mathbf{x},\mathbf{y}) = \frac{\sum_{0}^{n-1}(x_i - y_i)^2}{(\sigma_i + \alpha)}$$

to measure distance between an unknown sample and the centroid. If this distance was below a certain threshold then it was taken to be a fragment of that file type.

There is no indication as to the source of their file corpus. 57 files were first padded with zeros to ensure that the file was a multiple of 4 kB to simulate an unfragmented hard disc of 4 kB clusters. These 57 files were then concatenated to form one large 72 MB file. The file was scanned for each file type separately, and each 4 kB block was examined. For compressed (zip) files a fragment was marked as a hit even if it contained header information. The authors noted that compressed types were difficult to tell apart because of the random nature of their byte distribution.

2.2.1. Entropy

Hall (Hall and Davis, 2006) departed from the BFD approach by suggesting that the entropy and compressibility of file fragments be used as identifiers. They used the idea of a sliding window to make *n* steps through the fragment. Compressibility and entropy values calculated at each step were saved as elements of the characteristic vectors, although the window contents were not actually compressed. The LZW algorithm was used and the number of changes made to the compression dictionary used as an indication of compressibility.

Centroids were calculated as usual by averaging element values in a training set. Two metrics were evaluated – the Pearson rank order correlation coefficient and a simple difference metric similar to that used by McDaniel and Heydari (2003):

$$d(\mathbf{x},\mathbf{y}) = \sum_{0}^{n-1}(|x_i - y_i|).$$

This metric suffers from the same criticism as that of Li et al. (2005) and results were poor. Identification of compressed fragment was only 12% accurate and other results were not given. It was noted that the method might be better at narrowing down possible file types than actually assigning a file type. The initial corpus was a set of files from the authors' personal computer.

2.2.2. Complexity and Kolmogorov complexity

Veenman (2007) combined the BFD together with the calculated entropy and Kolmogorov complexity of the fragment to classify the file fragment. The Kolmogorov complexity is a measure of the information content of a string which makes use of substring order. A large corpus of 13 different file types was used with 35,000 files in the training set and 70,000 in the test set. HTML and JPEG fragments had over 98% accuracy, however compressed files had only 18% accuracy with over 80% false positives.

Results were presented in a confusion matrix which made them easy to interpret.

2.2.3. *Statistical methods*

Another statistical approach was suggested by Erbacher (Erbacher and Mulholland, 2007). Only statistics calculated from the BFD were used rather than the BFD itself. Their analysis used a sliding window of 1 kB to examine each block of a complete file rather than file fragments. By using the sliding window, however, they could identify component data types within the containing file e.g. a JPEG image within a PDF file. They claimed that five calculated statistics were sufficient to classify the seven data types in the corpus of five files of each type but no results were presented.

This idea was developed by Moody (Moody and Erbacher, 2008), where the corpus consisted of 25 files of each type examined, and no compressed or encrypted files were included. It was found that several data types could not be classified because of the similarity of their structure. These files were processed through a secondary pattern matching algorithm where unique identifying characteristics such as the high rate of occurrence of '<' and '>' in html files was used for identification. Here again we see the use of specialised functions for different file types.

2.2.4. *Linear discriminant analysis*

Calhoun (Calhoun and Coles, 2008) followed the approach of Veenman (2007) by using the linear discriminant analysis for classification but used a selection of different statistics. Linear discriminant analysis is used to develop a linear combination of these statistics by modifying the weight given to each so that the classification is optimised. A statistic that discriminates well between classes will be given a bigger weight than one that discriminates less well. They also included a number of tests that could be classified as specialised, such as their ASCII test where the frequency of ASCII character codes 32–127 can be used to identify text files such as HTML or TXT. The authors noted that the data did not conform to the requirements of the Fisher linear discriminant that data should come from a multi-variate normal distribution with identical covariance matrices and this may explain some '*sub-optimal*' results. Overall only four file types were included in the corpus with 50 fragments of each type, and no compressed or encrypted file fragments were included. Fragments were compared in a pairwise fashion. For example JPEG fragments were tested against BMP fragments and the results of classification noted. JPEG was then tested against PDF and so on. There was no attempt at multi-type classification. Testing in such a way gives less chance of misclassification and the results should be interpreted with this in mind. Extensive tables of result for accuracy are given but again there is no data about false positives or true negatives. They noted that a modification to their methodology would be required to avoid the situation where the method fails and all fragments are classified as one type but which gives high accuracy.

2.2.5. *Multi-centroid model*

Ahmed (Ahmed et al., 2009) developed the methods of Li et al. (2005) but introduced some novel ideas. In creating their multi-centroid model they clustered files with similar BFD regardless of type. This implements their assumptions:

1. Different file types may have similar byte frequency distributions.
2. Files of the same type may have different byte frequency distributions.

Within each such cluster, linear discriminant analysis was used to create a discriminant function for its fragment types. Cosine similarity was used as a metric and was shown to give better results than the simplified Mahalanobis distance. The cosine similarity is defined as the cosine of the angle between the centroid, **x**, and fragment, **y**, BFD vectors:

$$\mathit{Similarity} = \cos(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}\cdot\mathbf{y}}{|\mathbf{x}||\mathbf{y}|}$$

Since all byte frequencies are non-negative the dot product is positive and therefore the cosine similarity lies in the closed interval $[0,1]$. If the cosine similarity is 1 then the angle between the vectors is 0° and they are identical other than magnitude. As the cosine similarity approaches 0, the vectors are increasingly dissimilar.

An unknown fragment was first assigned to a cluster using cosine similarity. If all file types in a cluster were of the same type then the fragment would be classified as that type. If not, then linear discriminant analysis was used to find the closest type match in the cluster. Ten different file types were used although compressed and encrypted types were excluded, and 100 files of each type were included in the training set and the test set. Whole files rather than file fragments were used and so header information was included, achieving 77% accuracy.

2.2.6. *Support vector machine*

Q. Li (Li et al., 2010) used the BFD only and took a novel approach using a Support Vector Machine (SVM) for data fragment classification. Only four file types – JPEG, MP3, DLL and PDF were used. There were no compressed or encrypted file types. The inclusion of the PDF file type may have affected their results since it is a container type – it can embed a variety of other formats within itself such as JPEG, Microsoft Office or ZIP. Thus it might be difficult to differentiate a fragment of a file labelled PDF from some of the other types. Their training corpus was 800 of each file type downloaded from the internet. Each file was split into 4096 byte fragments and the first and last fragment discarded to ensure that header data and any possible file padding was excluded. The test set was created by downloading a further 80 files of each type from the internet and selecting 200 fragments of each type. Accuracy for classification of the four file types was 81%.

By contrast to previous researchers, Axelsson (2010) used the publicly available data set govDocs1 (Garfinkel et al., 2009), making it easier for others to reproduce the results. The normalised compression distance (NCD) was used as a metric. NCD was introduced by Cilibrasi and Vitanyi (2004). Their idea was that two objects (not necessarily file fragments) are 'close' if one can be compressed further by using the information from the other. If C(x) is the compressed length of fragment x and C(x,y) is the compressed length of fragment x concatenated with fragment y then

$$NCD(x,y) = \frac{C(x,y) - \min(C(x),C(y))}{\max(C(x),C(y))}$$

The k-Nearest-Neighbour algorithm (kNN) was used for classification. The kNN approach to classification is simpler than the SVM. No classification function is computed. The training set is plotted in n-dimensional space. An unknown example is classified by being plotted and its k nearest neighbouring points from the training set determined. The unknown example will be assigned to the class which is most common among these k nearest neighbours. The results were poor and average accuracy was 35%.

In Gopal et al. (2011), commercial '*off-the-shelf*' software was evaluated against several statistical fragment analysis methods. SVM and kNN using the cosine similarity metric methods were used. The test corpus was created from the publically available RealisticDC dataset (Garfinkel et al., 2009) and consisted over 36,000 files of 316 file types. The file types are not listed so we do not know if any compressed or encrypted files were included. The true file types were taken to be those reported by Libmagic using the Linux 'file' command and so no account was taken of possible anti-forensic techniques such as file header modification which may have resulted in bias in the results. It was not reported if the first fragment of each file was included. This would have contained header information. The results reported on the performance on file fragments was for the SVM only and it is shown that they achieved 40% accuracy measured using the Macro-F1 measure (Özgür et al., 2005) with a 4096 byte fragment size. There is no breakdown of the results by file type and so we cannot gauge if some high accuracy file types are masking some with very low accuracy.

2.2.7. Lempel–Ziv complexity

Sportiello (Sportiello and Zanero, 2011) tested a range of fragment features including the BFD, entropy, Lempel–Ziv complexity and some specialised classifiers such as the distribution of ASCII character codes 32–127 which would characterise text based file types, and Rate of Change which we have seen is a good classifier for JPEG files. The corpus consisted of nine file types downloaded from the Internet and these were decomposed into a total of 28,000 blocks of 512 bytes for each file type. There is no indication if file header blocks were included. No compressed or encrypted data was included.

SVM was used but no multi-class classification was attempted. For each file type a separate SVM model was created to classify a fragment type against each of the other types individually. The experiment was actually run using a 4096 byte fragment size and no indication of how these fragments were created is given. There is no confusion matrix of the results and there is no mention of false negative results. The table of results is arranged by fragment type, feature and feature parameter (the C and γ for the SVM). These parameters vary by file type and it appears that only the results for the best parameter values for individual fragment types is given. It is therefore difficult to compare results with others.

Fitzgerald (Fitzgerald et al., 2012) used the publicly available govDocs1 data set. They created 9000 fragments of 512 bytes for each of 24 file types. The first and last fragment of each file was omitted. There were an equal number of fragments of each file type. They used an SVM using standard parameters. Both unigram and bigram (pairs of consecutive bytes) along with various statistical values were used for the fileprint. They selected up to 4000 fragments for the data set and apportioned these approximately in the ratio 9:1 as training and test sets respectively for the SVM. There is no mention of whether the 24 file types were represented equally in the selection. An overall accuracy of 47.5% was achieved but correct compressed file prediction averaged 21.8%. It was noted, as did Garfinkel et al. (2010), Li et al. (2010), Conti et al. (2010) that the classification of high entropy file fragments was challenging.

2.2.8. Specialised approaches

It is argued by Roussev (Roussev and Garfinkel, 2009) that using BFD or statistical methods is too simplistic. They advocate using all tools available for file fragment discrimination. If distinct markers are present within a fragment then these should be used. If a byte pattern suggests a particular file type then knowledge of that file format can be used to check the classification. Thus they would be using purpose-built functions for each file type. They suggest a variety of approaches. As well as the header recognition and characteristic byte sequences as explained in the introductory section they use frame recognition. Many multimedia formats use repeating frames. If the characteristic byte pattern for a frame marker is found then it can be checked if another frame begins at the appropriate offset. If it does then it is likely that the fragment will be of that media type. It should be noted that, unlike other methods, they may require previous or subsequent file fragments. If a fragment type cannot be classified, then they use '*Context Recognition*' where these adjacent fragments are also analysed. Although the govDocs1 file corpus was used, this was supplemented by a variety of MP3 and other files that were specially developed. A discriminator for Huffman coded files was developed. However it's true positive rate was 21%. The need for further research in this area was stressed in the paper.

2.2.9. Genetic programming

A novel approach using Genetic Programming was tested by Kattan (Kattan et al., 2010). 120 examples of each six file types were downloaded at random from the Internet and no compressed or encrypted files were included. Analysis was done on whole files rather than fragments and so file headers may have been included. Features were first extracted from the BFD using Principal Component Analysis (PCA) and passed to a multi-layer artificial neural network (ANN) to produce fileprints. PCA removes redundancy by reducing correlated features to a feature set of uncorrelated '*principal components*' which account for most of the structure in the data. This removal of features is generally accompanied by a loss of information (Geiger and Kubin, 2012, p.562). PCA was used here to reduce the number of inputs to the next stage – a multilayer auto-associative neural network (ANN) which creates the fileprint. It is mentioned that file headers in themselves were not used, but would be part of the whole file. A 3 layer ANN was used with these fileprints as a classifier for unknown file types. Only 30 files of each type were used for testing. Results were reported in a confusion matrix and averaged 98% true positives. It is not clear whether the PCA would have extracted file headers as the most prominent component of the test data as a classifier. If this was so then the high detection rate would be explained.

This work was extended by Amirani (Amirani et al., 2013) to include detection of file fragments. The original version using an ANN as the final stage classifier was compared with classification using an SVM. 200 files of each of the 6 file types were collected randomly from the internet. Half were used as the training set and half as the testing set. For the fragment analysis a random starting point was selected in each file and a fragment of 1000 or 1500 bytes was taken. Results showed that the SVM classifier gave better results than the ANN for file fragments of both 1000 and 1500 bytes with extremely good results. It is puzzling that PDF detection gives 89% true positives with the 1500 byte fragments. The PDF format is a container format and might contain any of the other file types examined – doc, gif, htm, jpg and exe – as an embedded object. The random selection of 1500 bytes from within such a file could be misclassified as the embedded object type. The high detection rate for the PDF type itself means that this must have rarely happened. Perhaps it is an indication that the sample set of 100 files is too small, or perhaps the file header has an undue influence on the PCA.

2.3. *High entropy fragment classification*

In Garfinkel et al. (2010, p. S22) it is noted that "The technique for discriminating encrypted data from compressed data is in its infancy and needs refinement". This is supported by our observation that, in the literature considered so far, there has been little mention of classification of high entropy types. Where compressed fragments have been included in the test corpus, results have been poor. There is no source that deals with classification of encrypted or random fragments. This area of research has been recognised as difficult (Garfinkel et al., 2010; Fitzgerald et al., 2012; Li et al., 2010; Conti et al., 2010). Most results rely on patterns within the data. However Roussev and Garfinkel (2009) argues that compressed and encrypted file types have no such patterns. If a compressed file has patterns then it could be compressed further. If an encrypted file has patterns then it would be vulnerable to cryptanalysis. We need therefore to investigate methods of fragment identification that do not rely on patterns within the data.

2.3.1. *Randomness*

In Chang (Chang et al., 2010), the output from a number of compression algorithms and compression programs was tested for randomness. The NIST Statistical Test Suite (Rukhin et al., 2010) was used. It was found that every compression method failed the NIST tests. In the testing of the candidate algorithms for AES it was expected that encrypted files should be computationally indistinguishable from a true random source (Soto, 1999).

Zhao (Zhao et al., 2011) used 7 large (100 MB) test files and compressed and encrypted them by different methods. The whole 188 NIST tests were run against each file. At this scale they achieved good discrimination of encrypted files.

These observations lead us to our first hypothesis – that we can distinguish between compressed and encrypted fragments by testing for randomness.

2.3.2. *Compressibility*

Ziv (1990) stated that a random sequence can only be considered such if it cannot be compressed significantly. Schneier (1995, Ch. 10.7) noted that "*Any file that cannot be compressed and is not already compressed is probably ciphertext*". Mahoney (2012, Ch. 3.5.2) states that encrypted data cannot be compressed.

By contrast, compression algorithms always have to compromise between speed and compression (Zhao et al., 2011, p.5). It is unlikely that a compressed fragment is optimally compressed and therefore can be compressed further.

Our second hypothesis, therefore, is that we can differentiate between compressed and encrypted fragments by applying an efficient compression algorithm. A fragment of a compressed file should compress more than a fragment of an encrypted file.

3. Model design

To test our hypotheses we need to create a test corpus to test the classification methods that we develop to distinguish between encrypted and compressed file fragments. In this section we describe how we use publically available corpora so that our experimental results are repeatable by others. We use randomisation so that we can avoid bias. We also describe how we devised our own methodology to test our hypotheses.

3.1. *Building the corpus*

In the scientific method it is important that results be reproducible. An independent researcher should be able to repeat the experiment and achieve the same results. We have seen in our review of related work that this is not generally the case. Most research has been done with private or irreproducible corpora generated by random searches on the WWW. Garfinkel (Garfinkel et al., 2009) argues that the use of standardised digital corpora not only allows researchers to validate each other's results, but also to build upon them. By reproducing the work they have shown that they have mastered the process involved and are then better able to advance the research.

Such standardised corpora are now available. The Govdocs1 corpus contains a set of 1000 directories each containing 1000 files. Garfinkel et al. (2009, p.S6). We selected 5 of these directories at random to create our training set and 10 for our testing set. We need to create file fragments of representative compressed and encrypted types from these subsets. We can assume that multimedia types which use lossy compression have been classified by the techniques used in our review of related work. We will therefore consider only lossless compression methods in the remainder of our research.

3.1.1. Compression methods

There are a number of lossless compression methods. In order that our initial corpus is representative of compressed files '*in the wild*' we shall create it using four of the most common. Archivers and file compression programs can be categorised as either stream based, like zip, gzip, or use predictive compressors based on prediction by partial matching (PPM), or be block based, like bzip2, where a whole input block is analysed at once (Cilibrasi and Vitanyi, 2004, p.7).

The commonly used Deflate compressed data format is defined in RFC 1951. It uses the LZ77 algorithm followed by Huffman coding. It was originally designed by Phil Katz for the compression program PKZIP (Deutsch, 1996). It uses LZ77 which achieves compression by replacing a repeated string in the data by a pointer to the previous occurrence within the data along with the length of the repeated string. This is followed by Huffman coding which replaces common symbols within the compressed stream by short codes and less common symbols with longer codes.

This method is used by programs using the zip and gzip file formats. Although zip and gzip use similar methods, gzip is an application for compressing single files whereas the zip format is used by archivers. An archiver can compress multiple files into an archive and decompress single files from within the archive. Zip is a common format on Microsoft Windows platforms, but gzip is primarily a Unix/Linux compressor. We will use zip and gzip as representative of common file archivers and compressors in the creation of our corpus.

Bzip2 is a block coding compressor which uses run length encoding (RLE) and the Burrows–Wheeler transform. The B–W transform does not itself compress. It uses a block sort method to transform the data so that the output has long runs of identical symbols which can be compressed efficiently by RLE. The final output is again Huffman coded. Bzip2 is a file compressor rather than an archiver in that it compresses single files only. We will use bzip2 as representative of a block based Unix/Linux compressor.

There are also several proprietary compression implementations that are commonly used. Winrar is one such. Its own archiving format is proprietary but it is based on LZ and PPM. PPM is another stream based method which uses an adaptive data compression technique using context modelling and prediction to create probabilities for symbols in the data stream. It uses previous bytes (bits) in the stream to predict the next byte (bit). They are adaptive in that they adapt the algorithm automatically according to the data being compressed. The output is arithmetic rather than Huffman coded. Whereas Huffman coding is restricted to a whole number of bits, many modern data compressors use arithmetic coding which is not restricted by this limitation (Mahoney, 2012, Ch. 3.2). It can work with all the compression methods above. Winrar is used in our initial corpus as an example of proprietary compression software.

3.1.2. Encryption methods

In Microsoft Windows operating systems AES has been the default file and BitLocker drive encryption method since Windows XP. Triple DES has been available as an alternative (Microsoft, 2013). AES is also used by popular open source encryption software such as Axcrypt and TrueCrypt.

PGP, together with the open source GnuPG conforming to the OpenPGP standard in RFC4880 is the most widely used cryptographic system (Microsoft, 2013). It uses AES, Triple DES, Twofish and Blowfish.

We will use AES, Triple DES and Twofish as representative of encryption methods while creating our corpus.

3.1.3. Corpus creation

Hard discs store data in sectors. Since 2011, all hard disk drive manufacturers have standardised on a 4 KiB sector size. By emulation, these drives are backward compatible with the older 512 byte sector drives (The Advent of Advanced Format, 2013). In addition, regardless of sector size, operating systems store data in clusters. The usual cluster size is 4 KiB (Karresand and Shahmehr, 2007). We will therefore use 4096 bytes as our fragment size.

Most file systems store files so that the beginning of a file is physically aligned with a sector boundary (Garfinkel et al., 2010, p.S15). To emulate randomly sampled disc sectors we will therefore assume that each file begins on a sector boundary and consists of 4 K blocks. The first sector of any file will contain header information which may be used to identify a fragment type. If the file does not fill the last sector then this sector may contain padding or undefined content. For this reason we will exclude the first and last sector of any file from our corpus.

Each file in the training corpus was compressed individually by each compression or archiving program. Each file was also encrypted by each encryption method. A fragment beginning on a 4 K boundary and excluding the first and last fragments was randomly chosen from each file. Files which were less than 12 KiB after compression or encryption were excluded. It is not possible to select a random 4 KiB fragment from such files after first and last 4 KiB fragments are excluded. This generated a total of 25,000 fragments in our training corpus. Exactly the same procedure was used on each of the ten randomly chosen folders in the testing corpus and this generated a total of 49,000 fragments.

3.2. Fragment analysis tools

In this section we consider methods available to test our hypotheses:

1. We can distinguish between compressed and encrypted fragments by testing for randomness.
2. We can differentiate between compressed and encrypted fragments by applying an efficient compression

algorithm. A compressed file should compress more than an encrypted file.

No published work has been done in this area and so we will devise our own methodology to test our hypotheses.

3.2.1. *Testing randomness – the NIST statistical test suite*

It is important that the output of an encryption algorithm is random, otherwise it would be subject to cryptanalysis. The NIST Statistical Test Suite (Rukhin et al., 2010) was used in randomness testing of the AES candidate algorithms to test if their output was truly random (Soto, 1999). However Chang et al. (2010) used the NIST tests and found that compressed data tended to fail randomness tests. We used this finding to create our classifier. A compressed fragment should display poorer randomness than an encrypted one. We modified the NIST test suite so that it can operate on multiple files and output the results in the correct format for our ANN analysis.

The NIST Statistical Test Suite consists of a set of 15 tests. These tests are summarised in the table below. Note that all tests are done on binary data and not bytes. We use n to denote the number of bits in a sequence (Table 1).

We did not use the binary matrix rank test, overlapping template matching, Maurer's Universal Statistical test, linear complexity or the random excursions tests. Our bit sequence was not long enough to make the results of these tests statistically valid. However, since we were using 4096 bytes = 32768 bits, our fragments allowed us to use 64 binary sequences of 512 bits which satisfies the size requirements for all other tests.

Soto (1999) suggests that the results of the tests are first analysed in terms of the number of our sequences passing each test. We are using 64 sequences of 512 bits. For each sequence a P-value is calculated. H_0 is accepted if the $P\text{-value} \geq \alpha$. At our 0.01 level of significance it is expected that at least 60 of our 64 binary sequences making up the fragment will pass the test if the fragment is truly random. Secondly, if the fragment is random then the P-values calculated in the 64 sequence tests should be uniformly distributed. A P-value of P-values is calculated and if this P-value is greater than 0.0001 then the sequence of P values is taken as uniformly distributed.

We run a total of nine statistical tests from the test suite on each file fragment. Each test generates two values – the number of the 64 sub-sequences of 512 bits passing the test and a uniformity value. Two of the tests report two results each and so we will have a total of 11 pairs of values generated to form our characteristic vector for each fragment. Thus our characteristic vector v_f for file fragment f is defined as the sequence $v_f = (n_1, u_1, n_2, u_2, \ldots\ldots\ldots, n_{11}, u_{11})$ where n_i is the number of the 64 sequences passing test i and u_i the uniformity P-value for those 64 tests.

3.2.2. *Testing compressibility*

The probability that an encrypted (random) fragment will losslessly compress even by a small amount is low. Consider a random fragment of n bits. There are 2^n possible

Table 1
The NIST statistical test suite.

Test name	Description	Sequence size recommendation
Frequency (Monobit)	Proportion of zeroes and ones	$n \geq 100$
Frequency within a block	Splits sequence into N blocks of size M and applies the monobit test on each block	$M \geq 20$, $M > 0.1n$ $N < 100$
Runs test	Checks if total number of runs of length k, which are sequences of identical bits, is consistent with random data	$n \geq 100$
Block runs test	Runs test on data split into blocks of length M	For $n < 6272$, $M = 8$ For $n < 750$ K, $M = 128$
Binary matrix rank	Checks for linear dependence between fixed length substrings	$n \geq 38912$
Discrete Fourier test	Detects any periodic features in the sequence	$n \geq 1000$
Non-overlapping template matching	Checks number of occurrences of a target string of length m in N blocks of length M bits. Skips m bits when pattern found	$n \geq 10^6$
Overlapping template matching	As non-overlapping but does not skip	$n \geq 10^6$
Maurer's universal statistical test	Detects if a sequence is significantly compressible	$n \geq 387840$
Linear complexity	Determines if the complexity of a sequence is such that it can be considered random	$n \geq 10^6$
Serial test	Checks for uniformity – every m bit sequence should have the same chance of occurring	$m < \lfloor \log_2 n \rfloor - 2$ i.e. floor($\log_2 n$) − 2
Approximate entropy	Compares the frequency of all overlapping patterns of size m and $m+1$. These frequencies are compared against what would be expected of a random sequence	$m < \lfloor \log_2 n \rfloor - 5$
Cumulative sums	Calculates the maximum distance from zero a random walk (the cumulative sum adjusted so 0 is represented by −1, and 1 by 1) achieves.	$n \geq 100$
Random excursions	Measures deviation from that expected of a random walk (as above) to certain states	$n \geq 10^6$
Random excursions variant	Calculates the number of times a given distance from origin is visited in a random walk. Detects deviations from that expected of random sequence	$n \geq 10^6$

different fragments. Let P be the probability that the fragment will compress by 4 bytes (32 bits) or less.

Then:

$$p = \frac{Number\ of\ fragments\ \ that\ compress\ by\ 32\ bits\ or\ less}{Total\ number\ of\ possible\ fragments}$$

If the fragment compresses by 32 bits or less then the fragment of size n must map to one of the fragments of size $n-1, n-2, \ldots, n-32$. There are only $2^{n-1} + 2^{n-2} + 2^{n-3} + \ldots + 2^{n-32} = \sum_{m=n-32}^{n-1} 2^m$ such fragments.

Since the compression is lossless, the decompression must map back to a unique original fragment. Thus the mapping must be 1–1. Thus there are only this many fragments which will compress by 4 bytes or less.

Thus:

$$p = \frac{\sum_{m=n-32}^{n-1} 2^m}{2^n}$$
$$= \frac{\sum_{m=0}^{n-1} 2^m - \sum_{m=0}^{n-31} 2^m}{2^n}$$
$$= \frac{(2^n-1)-\left(2^{n-30}-1\right)}{2^n}$$
$$= \frac{2^n-2^{n-30}}{2^n}$$
$$= 1-2^{-30}$$

The probability that a random fragment compresses by 32 bits or less is $1 - 2^{-30}$ which is, for practical purposes, indistinguishable from 1. Also the probability that a random fragment compresses by more than four bytes is therefore $1 - P$(compresses by 4 bytes or less) which is for practical purposes equal to zero. Thus if a fragment compresses by more than 4 bytes we can assume that it is not encrypted with high confidence. We will use this fact to classify our fragments.

We need to use a compression algorithm which will meet several requirements. Firstly we need an algorithm which will compress more optimally than standard algorithms such as ‘deflate’. Secondly, ‘**deflate**’ uses <length, distance> pairs and literals which are then Huffman coded. There is a chance that literal bytes will align on a byte boundary and so a bytewise compressor might see them. However in Huffman coding the data is packed as bits and the three bit header will throw out this alignment. Also literals are likely to become rare further into the stream. The situation is worse with dynamic Huffman coding as codes can be nearly any length. A bitwise compression algorithm, however, is not constrained by lack of byte alignment. It will be able to see repeating literals or <length, distance> pairs (Kumar et al., 2010). For these reasons we are going to use zpaq as our for compressing our file fragments. In addition to being a suitable bitwise compression program it uses bit prediction. It maintains a set of context models which independently create probabilities for the next bit. The probabilities are combined to make the prediction. We would expect that, by definition, it would not be possible to predict the next bit in a random fragment. In a fragment which has not been optimally compressed, however, there must be some remaining pattern and hence predictability otherwise it would be optimally compressed. Zpaq should be able to detect this and give a more optimal compression. Our classifier in this instance will be simply the compressed size of the fragment. Using the archive program ZPaq to compress our fragments of course leads to the archive file structure and metadata being added to the compressed data. However to mitigate the possibly confounding the results we use a constant fixed size of filename for the fragments and also the fragment length is fixed.

4. Implementation and results

4.1. Classification by statistical analysis of randomness

The NIST test suite was run against our training and our testing corpora separately. RapidMiner (RapidMiner Data Mining Software, 2012) was used to create an artificial neural network (ANN) using 10-fold cross validation with default parameters using the training corpus. The model was then used to classify our test data. Rather than treat the 49,000 files as a block we retained them in their original folders of approximately 5000 fragments each in order to test if our method was performing consistently on all sets. The results are shown in Table 2.

Table 2
Classification results from 4 KiB fragments.

GovDocs1 folder	Predicted type			
	Actual type	Encrypted	Compressed	Accuracy
027	Encrypted	2270	207	92%
	Compressed	722	1600	69%
050	Encrypted	2231	220	91%
	Compressed	718	1635	69%
158	Encrypted	2012	192	91%
	Compressed	727	1355	65%
220	Encrypted	2140	191	92%
	Compressed	814	1473	64%
374	Encrypted	2253	201	92%
	Compressed	741	1677	69%
410	Encrypted	2212	197	92%
	Compressed	740	1457	66%
679	Encrypted	2423	235	91%
	Compressed	875	1848	68%
869	Encrypted	2251	203	92%
	Compressed	862	1551	64%
891	Encrypted	2266	214	91%
	Compressed	951	1582	62%
922	Encrypted	2411	201	92%
	Compressed	833	2018	71%

Encrypted file fragments have been identified with good accuracy. Type I errors are low. However it is obvious from the Type II errors that approximately 35% of compressed files are being identified as encrypted.

4.1.1. *8 KiB fragments*

We retrained the ANN with 8 KiB fragments and ran the same tests but used an 8 KiB fragment size to see if there would be any improvement in classification accuracy (Table 3).

It can be seen that there is a marked improvement over the 4 KiB fragment size. The most notable is that compressed fragment detection has risen from an average of 67% to 82%.

4.2. *Effect of corpus compression method*

The compressed fragments in our initial corpora consist of fragments of files compressed by popular compression and archiving programs using default settings. A new corpus was created using both default and maximum compression using both popular and efficient programs. The formats created were bzip2, zip and 7zip. To ascertain if the degree of compression affected results we created the compressed fragments in both default and maximum compression modes for each compressor. The encrypted fragments were AES, 3DES and TwoFish as before. The ANN was retrained using a training corpus of 30,000 fragments of these new types. The testing corpus contained a total of over 17,000 fragments. The results are shown in Table 4.

It can be seen that although fragments of encrypted files are still detected with good accuracy, the misclassification of fragments of compressed files has risen considerably. In order to clarify the reason for this increase in misclassification we did an analysis of the classification of the fragments of compressed files by their original file compression method. The results are given in Table 5.

The average compression ratio achieved on the corpus by each method is included. It can be seen that there is a correlation between the degree of compression and the accuracy of the classification. There is a trend for misclassification to increase as the degree of compression rises. Fragments of highly compressed files account for most of the increase in misclassification. Whether the standard or maximum compression option is chosen with the various compressors seems to make no significant difference to compression ratio and hence the classification.

Table 3
ANN results with 8 KiB fragment size.

Actual type	Predicted type		
	Encrypted	Compressed	Accuracy
Encrypted	2108	197	91%
Compressed	320	1432	82%

Table 4
4 KiB fragments.

Actual type	Predicted type		
	Encrypted	Compressed	Accuracy
Encrypted	6366	766	89%
Compressed	5145	4869	49%

Table 5
Analysis of compressed fragment classification (m = maximum compression).

Actual type	Predicted type			
	Compression ratio	Compressed	Encrypted	Accuracy
zip	1.91	1199	502	70%
mzip	1.91	1198	493	71%
bz2	1.96	1031	639	62%
mbz2	1.97	1080	584	65%
7z	2.13	171	1472	10%
m7z	2.14	189	1452	12%

4.3. *Fragment size*

To determine if increased fragment size would improve classification, corpora of 8 KiB and 16 KiB fragments were created including fragments of zip, bzip and 7z at both default and maximum compression settings and the ANN retrained on these fragment sizes.

4.3.1. *8 KiB fragment size*

The results are given in Table 6.

It can be seen that overall accuracy increases over the 4 KiB fragment size but misclassification of fragments of compressed files is still high. An analysis of the classification of 8 KiB fragments of compressed fragments by their original file compression method is given in Table 7.

Since these fragments were extracted from the same compressed corpus as the 4 KiB fragments the compression ratios are as in Table 7. Although the accuracy of prediction of zip and bz2 fragment types has increased with the increased fragment size, it is anomalous that the accuracy of prediction of 7z fragments has decreased.

4.3.2. *16 KiB fragment size*

The corpus for 16 KiB fragments is slightly smaller since some files were not large enough to produce a 16 KiB fragment. The overall accuracy has increased over the 8 KiB fragments as shown in Table 8.

The classification of fragments of compressed files was again analysed by their original compression method. The compression ratios are as before. The results are given in Table 9.

The accuracy of prediction of zip and bz2 fragments has increased with the increased fragment size as would be expected. However we see again that the accuracy of

Table 6
8 KiB fragment classification.

Actual type	Predicted type		
	Encrypted	Compressed	Accuracy
Encrypted	5984	471	93%
Compressed	3808	4657	55%

prediction of 7z fragments has decreased markedly with increasing sample size. This anomalous decrease in accuracy with increasing fragment size is worthy of investigation in future work.

4.4. *Classification by compression of fragments*

Analysis of the results of compressing the fragments in the corpus was simpler. We have one figure produced for each fragment – the size of the fragment after it is compressed. Our hypothesis was that a fragment that was originally compressed will compress more than an encrypted fragment. We classified the fragments by their compressed file size. If it was bigger than a given size then we classified it as encrypted, otherwise it was classified as compressed. The confusion matrix below shows results for classifying our original 4 KiB test corpus fragments using compression. The classification of compressed files is better than the NIST statistical tests. As shown in Table 10, the overall correct detection rate is over 70%.

4.4.1. *Effect of corpus compression method*

The 8 KiB and 16 KiB fragment corpora including the maximally compressed fragments were now analysed to test the effect of the degree of compression on the classification. The results are given in the tables below.

4.4.1.1. Analysis of 8 KiB fragments. Tables 11 and 12 show the effect of increasing the fragment size to 8 KiB.

The detection of encrypted fragments has increased in accuracy but the introduction of the more highly compressed types has reduced the overall accuracy of classification of the fragments of compressed files. An analysis of the fragments of compressed files by the original compression method is given in Table 12.

The correlation between the degree of compression and classification accuracy is more marked than with the statistical analysis. This could be expected since a fragment that was originally highly compressed will compress less and so be more difficult to differentiate from an encrypted fragment by compressed size. For the same reason the fragments maximally compressed by each method are also less accurately classified than those with the default compression setting which was not generally the case with the statistical analysis. Although the classification accuracy of the more highly compressed files (7zip) is poor, it is markedly better than the classification of these fragment types by statistical analysis.

Table 7
8 KiB compressed fragment classification by type (m = maximum compression).

Actual type	Predicted type			
	Compression ratio	Compressed	Encrypted	Accuracy
zip	1.91	1235	199	86%
mzip	1.91	1227	201	86%
bz2	1.96	972	448	68%
mbz2	1.97	1006	404	71%
7z	2.13	104	1278	8%
m7z	2.14	112	1274	8%

Table 8
16 KiB fragment classification.

Actual type	Predicted type		
	Encrypted	Compressed	Accuracy
Encrypted	5817	189	97%
Compressed	3473	4365	56%

Table 9
16 KiB compressed fragment classification by type (m = maximum compression).

Actual type	Predicted type			
	Compression ratio	Compressed	Encrypted	Accuracy
zip	1.91	1223	113	92%
mzip	1.91	1208	118	91%
bz2	1.96	925	386	71%
mbz2	1.97	935	367	72%
7z	2.13	40	1243	3%
m7z	2.14	34	1244	3%

4.4.1.2. Analysis of 16 KiB fragments. Table 13 shows the analysis of the results for 16 KiB fragments.

Again the encrypted fragment classification accuracy has increased with increasing fragment length. However the overall accuracy of classification of fragments of files that were originally compressed has not improved. An analysis of the fragments of compressed files by the original compression method is given in Table 14.

Although the accuracy of classification has improved for most compression methods it is noticeable that again the classification of the 7zip type is poorer with the larger fragment size. This mirrors the anomalous behaviour of these fragment types when subjected to statistical analysis. There is no obvious reason as to why these fragments of highly compressed files should be less accurately classified as fragment size increases and the anomaly is worthy of further investigation.

5. Conclusions and future work

The aim of this work was to advance the research in the area of file fragment classification. We focussed on fragments of high entropy file types, specifically fragments of compressed and encrypted files. We make the following main contributions:

- We critically reviewed the literature in this research area and categorised methods of detection and analysis.

Table 10
Analysis of fragment compression classification by category.

Actual type	Predicted type		
	Encrypted	Compressed	Accuracy
Encrypted	18643	5887	76%
Compressed	7254	16925	70%

Table 11
8 KiB fragment classification.

Actual type	Predicted type		
	Encrypted	Compressed	Accuracy
Encrypted	5286	1506	78%
Compressed	14611	18513	56%

Table 12
8 KiB compressed fragment classification by type (m = maximum compression).

Actual type	Predicted type			
	Compression ratio	Compressed	Encrypted	Accuracy
zip	1.91	4504	1113	80%
mzip	1.91	3734	1845	67%
bz2	1.96	3414	2136	62%
mbz2	1.97	2689	2844	49%
7z	2.13	2667	2758	49%
m7z	2.14	1502	3918	27%

This led us to propose two approaches to the classification of high entropy file fragments.

- By using a subset of the NIST statistical tests for randomness we created a classifier that achieved 91% correct classification of encrypted and 82% correct classification of compressed fragments. The classifier used an Artificial Neural Network to analyse the results of the statistical tests.
- We achieved 76% correct classification of encrypted and 70% correct classification of compressed fragments by using the size of a fragment once it was (re)compressed.
- By compressing the corpus with compression methods which achieved higher compression ratios we showed that there is a clear correlation between the degree of compression and misclassification of fragments.
- Using different fragment sizes we showed that classification accuracy generally improved with fragment size but noted that there is an anomaly in that the fragments that were originally more highly compressed appear to decrease in classification accuracy as fragment size increased.

Future work will include improved compression of fragments. It was noted that during the statistical analysis we already have an accurate frequency count of byte values within the fragment. Using an accurate frequency count together with arithmetic coding will deliver near optimal compression of each fragment (Howard and Vitter, 1994) and so should provide more discrimination of the small differences in compressed size that we are analysing. It also avoids the overheads of using an archiver to compress a fragment. In a production environment this analysis would be done on the byte array in memory which would be read directly from the storage device.

If the anomalous behaviour of our tests on file fragments from files that have been highly compressed proves to be consistent across further corpora then this needs to be investigated. It may be that the reasons for this anomalous behaviour might inform a methodology for classification of such fragments.

Table 13
16 KiB fragment classification.

Actual type	Predicted type		
	Encrypted	Compressed	Accuracy
Encrypted	4941	1151	81%
Compressed	12976	16543	56%

Table 14
16 KiB compressed fragment classification by type (m = maximum compression).

Actual type	Predicted type			
	Compression ratio	Compressed	Encrypted	Accuracy
zip	1.91	4257	749	85%
mzip	1.91	3944	1033	79%
bz2	1.96	2906	2027	59%
mbz2	1.97	2662	2265	54%
7z	2.13	1726	3115	36%
m7z	2.14	1050	3785	22%

References

Ahmed I, Lhee K, Shin H, Hong M. On improving the accuracy and performance of content-based file type identification; 2009. p. 44–59.

Amirani M, Toorani M, Mihandoost S. Feature-based type identification of file fragments. Security and Communication Networks 2013;6(April 2012):115–28.

Axelsson S. The normalised compression distance as a file fragment classifier. Digital Investigation Aug. 2010;7:S24–31.

Calhoun WC, Coles D. Predicting the types of file fragments. Digital Investigation Sep. 2008;5:S14–20.

Chang W, Fang B, Yun X, Wang S, Yu X, Ethodology M. Randomness testing of compressed data. Journal of Computing 2010;2(1):44–52.

Cilibrasi R, Vitanyi P. Clustering by compression. IEEE Transactions on Information Theory 2004;51(4):1–28.

Conti G, Bratus S, Shubina A, Sangster B, Ragsdale R, Supan M, et al. Automated mapping of large binary objects using primitive fragment type classification. Digital Investigation Aug. 2010;7:S3–12.

Copson ET. Metric spaces, paperback. Cambridge, England: Cambridge University Press; 1988.

Deutsch P. RFC 1951-DEFLATE compressed data format specification version 1. 3 IESG. IETF 1996;RFC 1951:1–15.

Erbacher RF, Mulholland J. Identification and localization of data types within large-scale file systems. In: Second international workshop on systematic approaches to digital forensic engineering SADFE07 2007. p. 55–70.

Fitzgerald S, Mathews G, Morris C, Zhulyn O. Using NLP techniques for file fragment classification. Digital Investigation Aug. 2012;9:S44–9.

Garfinkel SL. Random sampling with sector identification. In: Naval postgraduate school presentation 2010.

Garfinkel S. Digital forensics research: the next 10 years. Digital Investigation Aug. 2010;7:S64–73.

Garfinkel SL, Nelson A, White D, Roussev V. Using purpose-built functions and block hashes to enable small block and sub-file forensics. Digital Investigation Aug. 2010;7:S13–23.

Garfinkel S, Farrell P, Roussev V, Dinolt G. Bringing science to digital forensics with standardized forensic corpora. Digital Investigation Sep. 2009;6:S2–11.

Geiger BC, Kubin G. Relative information loss in the PCA. In: Proceedings IEEE information theory workshop 2012. p. 562–6.

Giordano J, Macaig C. Cyber forensics: a military operations perspective. International Journal of Digital Evidence 2002;1(2):1–13.

Gopal S, Yang Y, Salomatin K, Carbonell J. Statistical learning for file-type identification. In: 2011 10th international conference on machine learning and applications and workshops Dec. 2011. p. 68–73 no. DiiD.

Hall G, Davis WP. Sliding window measurement for file type identification. In: IEEE information assurance workshop 2006.

Howard PG, Vitter JS. Arithmetic coding for data compression. Proceedings of the IEEE June 1994;82(6):857.

Karresand M, Shahmehr N. Oscar—using byte pairs to find file type and camera make of data fragments. EC2ND 2006; 2007.

Karresand M, Shahmehri N. Oscar – file type identification of binary data in disk clusters and RAM pages. In: Proceedings of the 2006 IEEE workshop on information assurance, vol. 201; 2006. p. 140–7.

Karresand M, Shahmehri N. File type identification of data fragments by their binary structure. In: Information assurance workshop 2006. p. 140–7.

Kattan A, Galván-López E, Poli R, O'Neill M. GP-fileprints: file types detection using genetic programming. Genetic Programming 2010:134–45.

Kumar D, Kashyap D, Mishra KK, Misra AK. Security vs cost: an issue of multi-objective optimization for choosing PGP algorithms. In: 2010 International conference on computer and communication technology (ICCCT), vol. 1; Sep. 2010. p. 532–5.

Li W, Wang K, Stolfo S, Herzog B. Fileprints: Identifying file types by n-gram analysis. In: Proceedings of the 2005 IEEE workshop on information assurance and security 2005. p. 64–71.

Li Q, Ong A, Suganthan P, Thing V. A novel support vector machine approach to high entropy data fragment classification. In: Proceedings of the South African information security multi-conference 2010.

Mahoney M. Data compression explained. Data Compression Explained – Dell Inc.; 2012 [Online]. Available: http://mattmahoney.net/dc/dce.html [accessed 14.07.12].

McDaniel M, Heydari M. Content based file type detection algorithms. In: Proceedings of the 36th annual Hawaii international conference on system sciences 2003.

Microsoft. Cryptography, Crypto API and CAPICOM. Windows Dev Centre; 2013 [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/aa380251(v=vs.85).aspx.

Moody SJ, Erbacher RF. SÁDI – statistical analysis for data type identification. In: 2008 Third international workshop on systematic approaches to digital forensic engineering May 2008. p. 41–54.

Özgür A, Özgür L, Güngör T. Text categorization with class-based and corpus-based keyword selection. In: Proceedings of the 20thInternational conference on computer and information sciences 2005. p. 606–15.

RapidMiner Data Mining Software [Online]. Open Source – Available: http://rapid-i.com/content/view/181/190/; 2012 [Accessed. 05.08.12].

Rogers M, Goldman J. Computer forensics field triage process model. Digital Forensics 2006:27–40.

Roussev V, Garfinkel SL. File fragment classification-the case for specialized approaches. In: 2009 Fourth international IEEE workshop on systematic approaches to digital forensic engineering May 2009. p. 3–14.

Rukhin A, Soto J, Nechvatal J. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST Special Publication 2010;22(April, 2010).

Schneier B. Applied cryptography: protocols, algorithms, and source code in C. 2nd ed. New York: John Wiley & Sons. Inc.; 1995. p. 784.

Soto J. Randomness testing of the AES candidate algorithms. NIST; 1999. p. 14. Available via csrc. nist. gov.

Sportiello L, Zanero S. File block classification by support vector machine. In: 2011 Sixth international conference on availability, reliability and security Aug. 2011. p. 307–12.

The Advent of advanced format. International Disk Drive Equipment and Materials Association; 2013 [Online]. Available: http://www.idema.org/?page_id=2369 [accessed 27.08.13].

Veenman CJ. Statistical disk cluster classification for file carving. In: Third international symposium on information assurance and security Aug. 2007. p. 393–8.

Wang K, Stolfo SJ. Anomalous payload-based network intrusion detection. Recent Advances in Intrusion Detection 2004;3224:203–22.

Xing EP, Ng AY, Jordan MI, Russell S. Distance metric learning, with application to clustering with side-information. Learning 2003;15(2):505–12.

Zhao B, Liu Q, Liu X. Evaluation of encrypted data identification methods based on randomness test. In: 2011 IEEE/ACM international conference on green computing and communications Aug. 2011. p. 200–5.

Ziv J. Compression, tests for randomness and estimating the statistical model of an individual sequence. Sequences; 1990.