HEDGE: Efficient Traffic Classification of Encrypted and Compressed Packets

Fran Casino
Department of Informatics
University of Piraeus

Kim-Kwang Raymond Choo
Department of Information Systems and Cyber Security
University of Texas at San Antonio

Constantinos Patsakis
Department of Informatics
University of Piraeus

May 29, 2019

Abstract

As the size and source of network traffic increase, so does the challenge of monitoring and analysing network traffic. Therefore, sampling algorithms are often used to alleviate these scalability issues. However, the use of high entropy data streams, through the use of either encryption or compression, further compounds the challenge as current state of the art algorithms cannot accurately and efficiently differentiate between encrypted and compressed packets. In this work, we propose a novel traffic classification method named HEDGE (High Entropy DistinGuishEr) to distinguish between compressed and encrypted traffic. HEDGE is based on the evaluation of the randomness of the data streams and can be applied to individual packets without the need to have access to the entire stream. Findings from the evaluation show that our approach outperforms current state of the art. We also make available our statistically sound dataset, based on known benchmarks, to the wider research community.

1 Introduction

Network traffic is increasing at an unprecedented race. For example, studies from IBM[1] and CISCO[2] reportedly estimated that significant amount of today's

---

[1] https://www.ibm.com/blogs/insights-on-business/consumer-products/2-5-quintillion-bytes-of-data-created-every-day-how-does-cpg-retail-manage-it/

[2] https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html

data were created in the past few years. In our current landscape, a significant amount of data is generated and exchanged through mobile and Internet of Things (IoT) devices, such as those found in smart homes and smart cities. There are also corresponding security and privacy risks and challenges associated with such a trend. A typical measure is to perform network traffic monitoring to prioritise and balance traffic load (e.g. to ensure quality of service – QoS), as well as guaranteeing security and privacy for user data and systems (e.g. by detecting and preventing malicious behaviours).

Currently, network data (data-in-transit) are generally encrypted, as non-encrypted traffic can be subject to a broad range of attacks (e.g., impersonation attack, man-in-the-middle attacks and false data injection attacks), as well as surveillance by both honest-but-curious and malicious threat actors (e.g. violating user anonymity). By analysing the features and characteristics of traffic flows, including encrypted traffic flows, one may be able to identify specific user behaviours [7, 48]. For example, the authors in [57] presented a machine learning-based approach to identify WeChat (a widely used social and payment mobile application) packets and identified transfer transactions by simply analysing the intercepted TLS/SSL traffic. However, current state-of-the-art approaches are generally not capable of differentiating high entropy streams with high accuracy. As later discussed in Section 2, only few works focus on distinguishing between encrypted and compressed data, being the work presented in [17] the most similar to ours as other methods exploit additional features of the collected packages.

Most security and privacy mechanisms rely on (strong) encryption algorithms to protect the communications. However, flawed implementations often use cleartext communication [55] to transfer sensitive information. The latter is often in IoT devices due to their lack of processing resources. The problem is amplified by the fact that IoT firmware is closed source and cannot be easily extracted since access to the storage and processing units can be concealed or access-protected [58]. The use of compression instead of encryption [17] from some devices may perplex the problem even more as there are no efficient and accurate methods to distinguish high entropy sources, e.g. encryption from compression. Therefore, a security investigator cannot easily determine whether a device is using a custom compression algorithm or encryption when evaluating its security.

On the contrary, an adversary may penetrate an internal network. To cover his/her actions, the adversary may choose to use compression instead of encryption when exfiltrating data from hosts as encrypted channels may not be available and are not always easy to establish, e.g when pivoting from one machine to another. Note that the compression of data would render many signature-based detection methods useless in network level. The above should also be considered in the context of big data, which may prevent many mechanisms to examine all packages in real-time.

It is, therefore, mandatory to be able to detect encrypted and unencrypted data in network traffic to ensure proper management, auditing and abnormal behaviour detection [20, 49]. Most of state-of-the-art methods are able to detect

2

cleartext/plaintext communications or low entropy file-types (e.g. images and binaries), but are not able to discern between high-entropy files, especially in the case of compressed and encrypted data [55, 17]. This hinders proper detection of unprotected data being transmitted, which in the case of compressed files can be easily recovered. Therefore, tasks such as detecting devices transmitting unprotected information from individuals, as well as the proper application of privacy policies are difficult to accomplish.

Thus, in this paper we propose a novel traffic classification method named HEDGE, which stands for **H**igh **E**ntropy **D**istin**G**uish**E**r. HEDGE is based on the randomness evaluation of the payload information in randomly selected network traffic packets. Unlike majority of existing approaches, we do not analyse all network traffic. Instead, we only analyse random subsets to enable real-time detection. We select a set of randomness tests from the literature according to their accuracy and efficiency, and implement a threshold-based approach to distinguish between encrypted and non-encrypted high-entropy data streams in real-time.

We also develop a statistically sound dataset using well-known benchmarks and propose a set of strategies to evaluate the utility of our proposal. The dataset has an equally distributed number of file-types; thus, avoiding biases and enhancing the reliability of the outcomes. The code and implementations are publicly available in GitHub[3].

Based on the evaluation using this dataset, our proposed method correctly classifies 94.72% of random packets of 64 KB, outperforming current state of the art. Notably, our worst classification results are with small packets of 1 KB each, where the accuracy is slightly above 68.68%. For instance, the most accurate state-of-the-art work up-to-date that considers random packet inspection [17] achieves an accuracy of 66.9% using another sound benchmark, whilst our method achieves an accuracy above 70.6% using the same dataset. Note that a 4% is a significant enhancement in this context, especially for small size files, without the use of training or heavy computations. Similar works that study compressed and encrypted packet classification can be found in [52] and [19]. More concretely, Lin [52] proposes an SVM classifier according to a set of features to identify the type of traffic, including compressed and encrypted files. In the case of encrypted classification, they compute a set of features and train a model using SVM as well as Jensen Shannon divergence (JSD) [27]. Contrary to our proposal, this approach uses other traffic features to train their model achieving at most 79.8% accuracy with variable-sized packets, in all cases bigger than 1KB. In [19] authors use a similar approach to the one presented in [52] and compare it with a CART decision tree classifier. Nevertheless, they utilise traffic and packet information, as well as the first bits of each packet (i.e. *magic header bytes*) where characteristic information of the protocol/method used can be obtained. Although this approach has a series of constraints (e.g. scalability) to be used in real-time classification and thus differs from our approach, authors are able to obtain accuracies near 75% with SVM and CART and between 76%

---

[3]https://github.com/francasino/traffic_analysis

and 85% after testing several feature configurations with different file sizes (their tests consider sizes up to 64KB).

A comparative summary of the proposed method with the approaches in [17], [52] and [19] is presented in Figure 1. Moreover, our method is efficient and easy-to-adopt, avoiding costly training procedures and similarity computations. Further relevant findings when analysing behaviour of compressed and encrypted data depending on the input file-type are also discovered and documented in the paper. For instance, our method manages to distinguish with high accuracy between the file-types from single random packages.
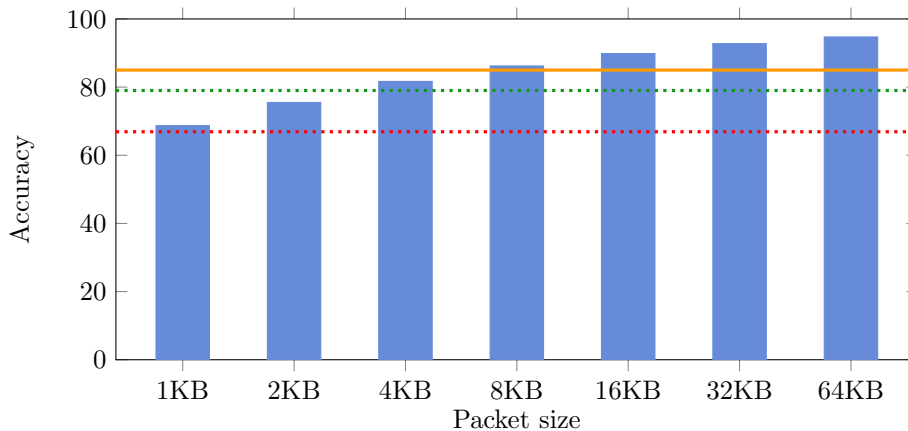

Figure 1: A comparative summary of the classification accuracy of the proposed method for each packet size and that of [17] (in red), which was only tested for 1KB files. Note that, contrary to us, in the case of [52] (in green) and [19] (in yellow), authors use additional traffic information as well as the first *magic header bytes* to train their classifiers.

The rest of this paper is organised as follows: Section 2 provides the reader with relevant background on network traffic analysis and existing approaches, and widely-used randomness tests. Section 3 is devoted to explaining HEDGE, our proposed method. Section 4 describes the experimental setup, the benchmarks and the tests performed to evaluate our proposal. The evaluation findings are later discussed in Section 5. Finally, Section 6 concludes the paper and identifies possible directions for future research.

2   Related Work

In network traffic analysis approaches, data streams are analysed and classified according to flow and packet information, payload content and statistics (including among others packet size and average times). Based on such information, we can distinguish between: (i) *feature-based* techniques, which determine a set of characteristics/features regarding the traffic flow as well as internal packet

information and its structure (i.e. it differs depending on the protocol used), and (ii) *payload-based* techniques, which inspect the data payload content and apply relevant methods to extract statistical properties and identify, for example, the type of information being transmitted (e.g. text, image, video, music, compressed or encrypted). We refer the interested reader to [20, 49] for a classification of network traffic analysis models based on input data, applied techniques applied and their corresponding outcomes.

Most feature-based techniques create a structure or vector containing the features of data flows and then classify them using statistical or machine learning methods [21, 35]. For instance, in [33] close to 250 discriminators/features are identified, which can be used for flow record classification. In the case of payload-based methods, different methodologies to accurately classify data streams according to their payload characteristics [54] have been studied in the last decade. Machine learning-based techniques generally focus on identifying the application associated with the stream or whether a flow contains some code. Signature-based approaches include hybrid schemes and usually try to exploit characteristics of protocols in some of the first bytes of the packet payloads. In the literature they have been used for identifying widely used application protocols (HTTP, FTP, SMTP) using features such as packet size, port, timing characteristics, and packet payload [13, 24, 9, 12, 61]. Another widely used method is byte frequency distribution, in which each file-type is classified according to its signature [50, 59]. In [2], the authors evaluated state-of-the art classifiers and compared them with a genetic algorithm. Experiments performed using two benchmarks (SSH and non-SSH samples) show a similar level of accuracy, but genetic algorithm achieves higher efficiency. In [5], the authors used k-nearest neighbours (k-NN) and k-means for flow classification. Their implementation enables efficient similarity computation, since only c comparisons are required (considering that files have been classified in c clusters) to find the closest flows and classify them accordingly.

Entropy-based classification methods have also been the focus of ongoing research [47, 53, 45, 30]. In addition to the use of Shannon entropy [44] or the chi-square test [8] to classify data streams, approaches such as those reported in [41, 16, 25] have been used to model the structure of files. Such techniques, however, incur high computational costs for outcome validation. The costs also increase with the amount of traffic analysed.

Although encrypted traffic hides all content from a potential eavesdropper, the first steps (initial handshake, connection parameter negotiation, protocol version) are performed in plaintext and therefore have distinguishable features. Exploiting the latter, machine learning and other classification methods can infer the underlying application protocols from encrypted traffic [3, 10, 56, 36], for example in SSH and web browsing traffic analysis [26, 46]. Specifically, classification algorithms used to detect encrypted web traffic rely on static object lengths and the previous creation feature-based libraries (i.e. training models). An associated limitation is that these methods are not reliable for dynamic encryption schemes applied to different file sizes and lengths, padding or other techniques. In the work of Zhang et al. [60], the authors analysed browsing traffic to detect

self-decrypting exploit code. In [4], the authors used three techniques (i.e. a multi-objective genetic algorithm (MOGA) [34], the C4.5 algorithm [42] and a semi-supervised k-means [14]) on different benchmark datasets to classify encrypted data streams on SSH. They performed binary identification of SSH traffic in the following two settings: (i) using test data from the training model, and (ii) using test data from a different network. Findings from their evaluation showed that MOGA performs better in the first case, whilst C4.5 outperforms the rest in the second. In [22], the authors characterised the differences between encrypted and non-encrypted traffic and designed a function to correlate the encrypted and non-encrypted traffic according to their features, as well as the minimum number of packets to guarantee an accurate classification.

Encrypted data is known to be more uniformly distributed than unencrypted data [31, 44]. Therefore, a number of approaches rely on this characteristic to locate cryptographic keys stored in memory and file system dumps [43]. The authors generally analysed big data streams, divided them into small blocks, and computed their entropy. Therefore, high entropy blocks may indicate the presence of encrypted data. Similarly, the authors in [37] and [11] first computed the entropy of packet payloads and then compared it with the entropy of uniformly randomly distributed sequences of the same length. However, the entropy estimation approach is not effective when the number of samples is small [38, 39]. Moreover, entropy measures are not reliable when other data which has high entropy is present such as compressed, MP3 or PDF files.

There has been renewed interest in distinguishing between compressed and encrypted data streams, partly due to the increasing use of end-to-end encryption in online communications. As discussed earlier, one of the most prevalent strategies is to use Shannon entropy or the chi-square test, among other measures over fixed-size sliding data streams (1KB, 2KB and so on) to distinguish between different types of data, as well as compressed and encrypted. However, one of the most challenging aspects, beyond distinguishing between high entropy files, is the time required to develop this classification in continuous traffic monitoring; thus, limiting its applicability in real-time scenarios.

As claimed by Malhotra et. al. [31], most traffic analysis tools analyse the contents of the data packets but they are not capable of determining whether data are encrypted. According to their research findings, Shannon entropy can be used to accurately classify low entropy streams. However, other techniques such as the chi-square test are needed to differentiate high entropy streams. Moreover, the authors noted that many compression algorithms have a fingerprint as their first bytes (*magic header bytes*) imply their format and thus, such information can enhance classification's accuracy. In [52], the authors used entropy and frequencies of characters to distinguish the type of payload content (e.g. text, picture, audio, video, compressed, base64-encoded image, base64-encoded text and encrypted), with small computing space and notable accuracy. In [19], the authors claimed that if the probability distribution of each subset of a file is similar, we can distinguish such file from others. However, such differences may be too small in the case of high-entropy compressed files and encrypted files, especially if we consider small files (< 4KB). In another work

6

[6], the authors performed k-NN classification to distinguish between random, compressed and encrypted data streams. In [51], authors use chi-square test to distinguish between encrypted and encoded (compressed) data to bypass DRM protection in streaming media services. Nevertheless, authors need to use a considerable amount of data (i.e. more than 380KB) to ensure reliable classification outcomes.

Existing methods generally rely on continuous traffic information to enhance their accuracy by collecting information about *complete* packet transmission, the beginning and the end of a connection or of a file and etc. Therefore, real-time monitoring is inefficient using such schemes, since they require the analysis of huge volumes of data. However, these schemes can be useful in studying past-events or analysing only specific connections. Hence, to enable real-time monitoring, our aim is to analyse the payload of a random subset of packets and infer as much information as possible. Similar to our approach, the work of Hahn et al. [17] presents the first, to the best of our knowledge, technique to distinguish encrypted from compressed unencrypted network transmissions by analysing random packets. The authors applied three machine learning models, with the convolutional neural network (CNN) model achieving the best results. However, more efficient solutions are needed to perform practical real-time traffic classification even in the case of random data stream analysis, as the methods applied in [17] (i.e. CNN and k-NN) are computationally expensive and require proper training.

3   Proposed Approach

In this section, first we state the assumptions that we make for our approach e.g. capabilities of the entities involved and the requirements that are needed from any such a solution. Then, we discuss how we perform the feature selection and the steps of our methodology.

3.1   Basic assumptions and requirements

In our model, we assume *Alice* and *Bob* are exchanging messages over a public channel and the exchanged message can either be plaintext, encrypted or compressed. Therefore, messages where one part is encrypted and the other part plaintext are beyond the scope of this study. Moreover, we assume that another entity, *Eve* (i.e. the adversary), can passively intercept the traffic between Alice and Bob. Typically in the literature, Eve is considered malicious, however, we do not assume malicious motives for her actions. That is because Eve might be a firewall, intercepting the traffic only to determine its content and block malicious requests or packets that may expose the nodes of the internal network to threats. Additionally, we assume that Eve cannot determine whether two or more packets belong in the same message or session, therefore, each packet is studied *individually*. The latter enables a more generic scenario with a weak assumption, compared to the bulk of the literature [52, 19]. Therefore, we as-

sume that Eve can analyse only a fragment of packets due to e.g. high volume of network traffic or processing constraints. Figure 2 shows an overview of the proposed scenario.

Furthermore, in order to enhance the applicability of our solution, we require the following properties:

- **Accurate:** The proposed solution should be able to distinguish between plaintext, compressed and encrypted files with high accuracy. If possible, the solution should be able to differentiate file-types (image, binary, etc.) as well.

- **Efficient:** The performed tests or methods must provide fast and robust responses. In addition to method efficiency, we want to implement a tree-based threshold system, so that in most cases, we do not need to run all tests to discard or classify an element (message); thus, improving the efficiency of the method.

- **Adaptable:** The proposed solution must allow fine-tuning of parameters, depending on the network traffic, to achieve better classification accuracy than static methods.

- **Reproducible:** The proposed methodology has to be easy to implement and reproduce, in order to facilitate adoption and verification. To address this, we will use state-of-the-art methods and a set of predefined strategies to automate the database creation and randomness tests.
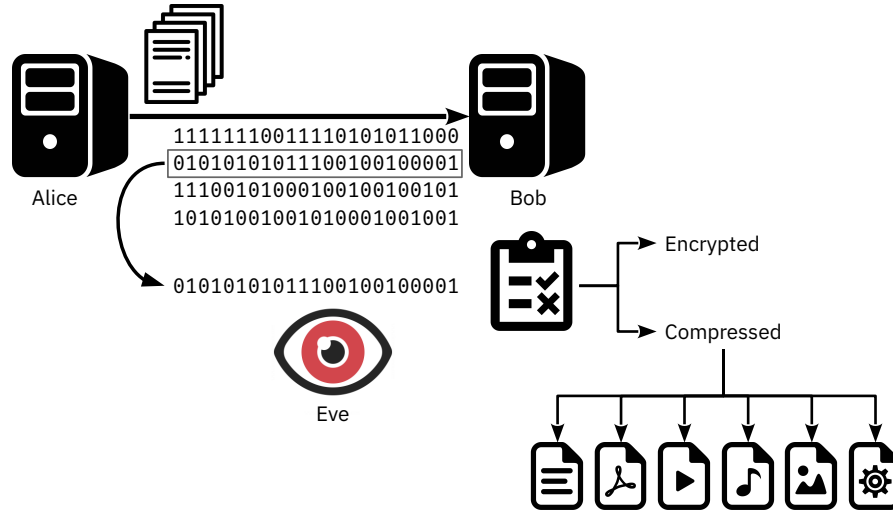
3.2 Feature Selection

Figure 2: An overview of the proposed approach.

Table 1: Features selected from each randomness test. Therefore, a total of 3 features will be used to define the threshold. Note that we aggregate the outcomes of NIST tests.

Method	Features
Chi square test	Absolute value
Chi square test	$\chi\%$ of confidence
NIST SP 800-22	Aggregate number of failed blocks

Table 2: Threshold values for each selected feature. The value x denotes the outcome of the randomness test for a single bit stream.

Chi abs. val.	$\chi\%$	NIST SP 800-22
$x \in AVG \pm \sigma$	$\chi\% > 99\% \ \| \ \chi\% < 1\%$	0 fails

After evaluating the performance and the computational cost of the methods described in Appendix .1 using random data from well-known benchmark datasets (later described in Section 4), only a subset of them met the requirements to be used in HEDGE. More precisely, the chi-square test can efficiently recognise true random bit streams from non-random ones and the % of confidence is a reliable indicator, exhibiting (according to our architecture) a better trade-off between computational time and accuracy than other distribution-based approaches such as Kolmogorov-Smirnov and Anderson-Darlong. FIPS-2-140 tests are very stable in the case of encrypted bit streams, that is passing the FIPS-2-140 test without fails. After discarding the common tests of SP 800-22 and FIPS-2-140, as they were already performed in the latter, we selected a small subset of SP 800-22 tests that could efficiently distinguish between compressed and encrypted files, without the 20.000 bits limitation though to accommodate for small packets. After experimenting with all tests of this suite, we identified that the ones that fit the requirements were the frequency within block test, the cumulative sums test, and the approximate entropy test.

The average and correlation tests, although having slightly more accuracy and less variability in encrypted than in compressed streams, do not provide reliable outcomes for small size files with high entropy. The other remaining methods could not be used as either (1) they did not provide a good indicator – both encrypted and compressed files had indistinguishable results, considering the standard deviation of the outcomes– or (2) they are computationally costly. For instance, the entropy is a good indicator only between raw files and the rest, according to our tests and [55, 17], and the Diehard most common tests (e.g. birthday spacing, parking lot or random spheres) are usually passed by both compressed and encrypted files. Moreover, Diehard tests, as well as TestU01 tests, require a considerable amount of time to be computed and thus, they are not suitable for real-time traffic analysis. A summary of the selected features is described in Table 1.
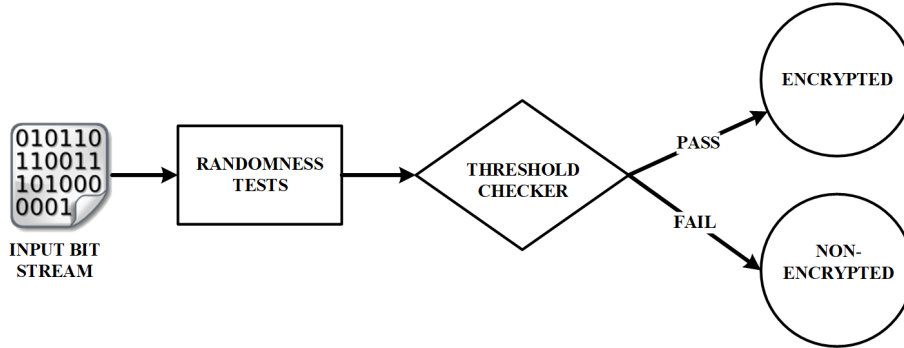
Figure 3: Step by step of our methodology.

To enable data payload classification, HEDGE implements a threshold-based method that takes into account the set of features selected from Table 1. The threshold values selected for each feature are described in Table 2. More concretely, we consider the average and its standard deviation σ for the chi square test. The $\chi\%$ most relaxed confidence (i.e. $\chi\% > 99\% \parallel \chi\% < 1\%$) is also considered (see Section .1). Finally, we set the threshold of the number of NIST SP 800-22 tests failed to 0. The values for each category have been selected after analysing the outcomes of a huge set of random encrypted files for each test. The threshold values selected reflect the stability of their outcomes regardless of the input file-type (e.g. text, image, binary and etc). Therefore, the results of the tests are almost indistinguishable between encrypted files, a feature which is discussed with more details in Section 4.

To introduce some variability and provide a more adaptable threshold to the input bit streams, HEDGE adds a *gain* factor γ to the threshold values. Therefore, we can apply relaxed or strict policies in our detection system (as later described in Figure 5 and Table 7), observe its behaviour and select the one that provides better outcomes according to the input data. In this regard, γ is added as a factor of the standard deviation of the threshold values in the case of the chi-square absolute value. The remaining features will not be affected by γ. For instance, the chi-square % confidence should be a fixed factor, as well as the NIST SP 800-22 test fails. Note that accepting one fail in NIST SP 800-22 test implies a considerable amount of false positives.

An overview of our method is depicted in Figure 3. First, we apply the randomness tests to the input bit stream and check whether the outcomes pass our threshold (according to a desired level and γ). We consider the data stream encrypted if, and only if, it passes all thresholds. Note that our approach can be easily parallelised, since it can be executed in independent machines regardless of the input data. Therefore, there is no need for a centralised database to compute distances/similarities, such as in the case of k-NN or clustering methods.

10

Table 3: Set of encryption and compression methods. Considering all encryption and compression variants, and each input generates a total of 10 different new files.

Encryption	Compression
AES (128 / 192 / 256)	**ZIP RAR BZIP2 GZIP**
Camelia (128 / 192 / 256)	

Table 4: Datasets used to obtain raw files for our experiments. All data have been randomly selected.

File-type	Benchmark
IMG	**COCO Dataset** [4]
IMG	**Microsoft Research** [5]
PDF	**ArXiv** [6]
TXT	**Project Gutenberg** [7]
MP3	**9th Symphony of Beethoven**
VIDEO	**Cambridge-driving Labeled Video Database** [8]
BIN/EXEC	**/System32 in Win10 x64**
	/sbin in Ubuntu 16.04

4 Experiments

In this section, we detail the experimental setup and the tests performed to validate the efficacy of HEDGE. In Section 5, we present the results.

As stated in Table 1, we use the chi-square and a subset of NIST SP 800-22 tests. However, since the outcomes of some tests vary depending on the file size, we apply our method to different bit stream sizes.

The first step of the experiment is to create a statistically sound dataset. Therefore, we use a random set of files retrieved from well-known benchmarks, summarised in Table 4, to create a heterogeneous dataset. Our dataset has the same number of files for each file-type (i.e. image, video, binary, audio, text and PDF) to avoid biases, since the randomness of compressed files is input-dependant (see Section 5). Note that we selected the most representative file-type according to the state-of-the-art [54, 17, 19].

Next, since the aim of the paper is to classify high entropy files, we generate a set of compressed and encrypted bit streams of fixed size which covers all powers of 2, starting from 1KB and up to 64KB. To cover the variability of real-life scenarios, we use a set of prevalent encryption and compression methods, which are summarised in Table 3. The dataset creation step is described in Algorithm 1. Both arrays *Sizes* and *Methods* used in Algorithm 1 *line 1* correspond to

---

[4]http://cocodataset.org/home

[5]https://www.microsoft.com/en-us/research/project/rgb-d-dataset-7-scenes/

[6]https://arxiv.org/

[7]https://www.gutenberg.org/

[8]http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/

Algorithm 1 Database Creation

1: **function** GENERATE DATASET(DataSet D, Array Sizes, Array Methods)▷
   The possible bit stream sizes and the compression and encryption methods
2:
3:     **while** (DatasetHasFiles) **do**
4:         f_i = SelectTheNextFile (D);   ▷ Next raw file in dataset
5:         V = CreateFileVariants (f_i, Array Methods); ▷ Creates a set of files
   for each f_i
6:         S = SplitFiles (V, Array Sizes);   ▷ Each file in V is split into
   different sizes
7:     **end while**
8: **end function**

Table 5: Threshold values for each selected feature. Values correspond to the values obtained considering all training sets of our experiments.

KB	Chi-square abs. val.	$\chi\%$	NIST SP 800-22
64	255.37 ± 22.82	$x > 99 \parallel x < 1$	0 fails
32	255.08 ± 22.68	$x > 99 \parallel x < 1$	0 fails
16	254.96 ± 22.76	$x > 99 \parallel x < 1$	0 fails
8	255.09 ± 22.54	$x > 99 \parallel x < 1$	0 fails
4	255.04 ± 22.60	$x > 99 \parallel x < 1$	0 fails
2	254.98 ± 22.57	$x > 99 \parallel x < 1$	0 fails
1	255.02 ± 22.57	$x > 99 \parallel x < 1$	0 fails

the different payload sizes (from 1KB to 64KB) and to the contents of Table 3, respectively. After applying Algorithm 1 to the data extracted from the benchmarks described in Table 4, the result is a collection of datasets (one for each file size), where each contains approximately 1GB of compressed and encrypted files. To further guarantee equally distributed amount of files and unbiased experiments, we randomly select a subset with the same number of encrypted and compressed files to perform our experiments. Therefore, our datasets consist of exactly 50% encrypted and 50% compressed files.

Next, we use an inverse 10-fold experiment on each dataset and repeat the experiments 1000 times. More concretely, a 10% of data is randomly selected (5% of compressed and 5% of encrypted files) and the threshold levels are trained using the encrypted data, while the 5% of compressed data is discarded. Hence, the remaining data (90%) is used for testing and validation. The possible outcomes of our classification are summarised in Table 6. Therefore, the accuracy of our method will be computed as the sum of $TP + TN$. In addition to our benchmark, we compute the accuracy of our method using another state-of-the-art dataset [17].

After computing the training dataset, we observed that our method could be applied to any file size, due to the stability of the training threshold levels.
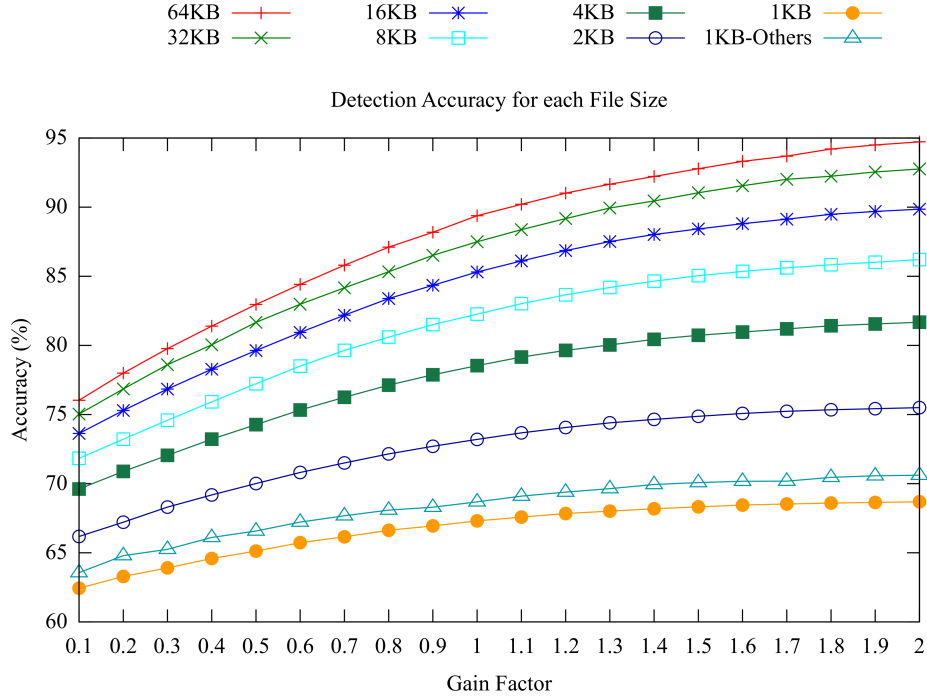
Figure 4: Accuracy detection according to the different file sizes and the parameters of our system.

Table 6: Possible outcomes of our method according to the input file-type.

Input	Output	Result
Encrypted File	PASS	true positive (TP)
Encrypted File	FAIL	false negative (FN)
Compressed File	PASS	false positive (FP)
Compressed File	FAIL	true negative (TN)

As supporting evidence, we compute the average of threshold values of our experiments (see Table 5). One can observe that the average deviation is very low and that the values are very similar for each file size. Therefore, the similarity of the threshold levels obtained from the training tests indicate the stability of the encrypted files' features. The latter means that our threshold values can be applied, regardless of the file-type and file sizes. In addition, we only need to train the method once at the beginning; thus, enabling efficient and adaptable classification.

(a) % of accuracy for encrypted detection in 64KB files.

(b) % of accuracy for compressed detection in 64KB files.

(c) % of accuracy for compressed detection in 32KB files.

(d) % of accuracy for compressed detection in 16KB files.

(e) % of accuracy for compressed detection in 8KB files.

(f) % of accuracy for compressed detection in 4KB files.

(g) % of accuracy for compressed detection in 2KB files.

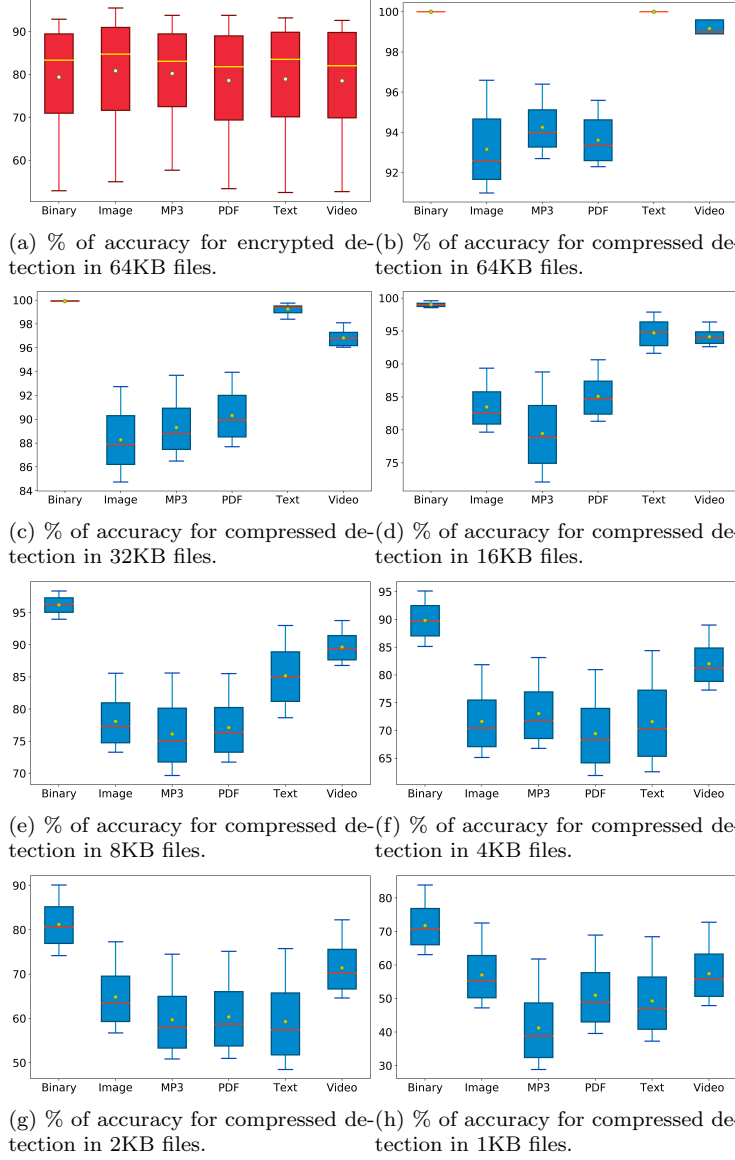(h) % of accuracy for compressed detection in 1KB files.

Figure 5: Detail of the accuracy according to input file-type in compressed and encrypted files. Lines represent the median whilst points represent the average value.

5   Discussion

The average accuracy outcomes for each file size according to γ are depicted in Figure 4. In terms of accuracy, the best outcomes are obtained by large file sizes (i.e. 64KB and 32KB). In general, we observe that with a high γ, we

obtain better results because the trade-off between detected compressed files (true positives) and missed encrypted files (false negatives) increases. This is mainly due to the stability of encrypted files' features. Therefore, the use of γ enhances the dynamism and adaptability of the system. We can also observe that our method achieves better accuracy than the state-of-the-art (see Figure 4, 1KB-Others), since the accuracy achieved by the authors in [17] was 66.9% and ours is 70.61%, using the same benchmark dataset proposed by them.

For the sake of clarity, Table 7 provides the detailed outcomes (as described in Table 6) for the most restrictive and the most relaxed case in each file size (i.e. for $\gamma = 0.1$ and $\gamma = 2$, respectively). We can observe that the values for encrypted files (i.e. TP and FN) are very stable, which means that, as expected, file's randomness properties are similar regardless of the file sizes, according to the tested values (i.e. a study of the randomness properties of files smaller than 1KB is left to future work). In the case of compressed files (i.e. FP and TN) their behaviour is rather variable in the case of small files. Nevertheless, compressed files' outcomes become more stable the higher the file size (i.e. the difference between outcomes for big files is lower than for small files). Moreover, such difference is also reduced when using $\gamma = 0.1$ and $\gamma = 2$ for high file sizes, which means that we can apply more relaxed policies in such cases. In general, the total amount of correctly classified files (i.e. accuracy) is higher when we apply $\gamma = 2$, since the degrowth rate of the TN value is lower than the growth rate of the TP value.

Next, after thoroughly analysing the results, we discovered that the randomness tests applied to compressed files exhibited different behaviours according to the input file-type. More precisely, there is a strong relationship between the randomness of the input file-type and the randomness of the compressed file generated. To document such behaviour, we classify the outcomes of all data streams according to input file-types for all the compressed (see Fig 5) algorithms tested (see Table 3). First, Fig. 5a, shows the accuracy detection of 64KB encrypted data streams, whose behaviour is similar to other file sizes and therefore we omit the results. It is apparent that when we analyse encrypted data streams we cannot distinguish the input file-type which is a result that complies with the theory. On the contrary, one can observe that compressed binary and compressed text files, as well as compressed video files are more easily classified by our method (achieving 100% accuracy with binary and text in 64KB file streams) than the rest. This implies that their randomness is much lower than other files, such as PDF or MP3 (i.e. MP3 is already a compressed file, as well as JPG files). This behaviour applies to all file sizes except for files lower than 4KB, where compressed text files become more indistinguishable, but we still find notable differences between each file-type. Such findings are relevant, especially from a security perspective, because discovering the content of compressed data streams enables the proper management when detecting exchange of unexpected/suspicious files (e.g. executables). Again, the result complies with the theory since the output of compression algorithms like Huffman or members of the Lempel-Ziv family highly depends on the statistical properties of the input stream.

Table 7: Excerpt of the outcomes (percentages) for the worst (most restrictive) and best (most relaxed) strategies.

File size	γ	TP	FN	FP	TN	Accuracy (TP+TN)
1 (Others)	0.1	26.72	23.28	13.16	36.84	63.56
	2	46.70	3.29	26.09	23.90	70.60
1	0.1	26.73	23.27	14.31	35.69	62.42
	2	46.69	3.31	28.01	21.99	68.68
2	0.1	26.58	23.42	10.41	39.59	66.17
	2	46.68	3.32	21.18	28.82	75.50
4	0.1	26.74	23.26	7.13	42.87	69.61
	2	46.77	3.23	15.1	34.90	81.67
8	0.1	26.68	23.32	4.84	45.16	71.84
	2	46.69	3.31	10.49	39.51	86.20
16	0.1	26.74	23.26	3.11	46.89	73.63
	2	46.86	3.14	7.01	42.99	89.85
32	0.1	26.85	23.15	1.81	48.19	75.04
	2	46.66	3.34	3.89	46.11	92.77
64	0.1	27.01	22.90	0.98	49.02	76.03
	2	46.82	3.18	2.09	47.91	94.72

In summary, the results show that we can distinguish compressed from encrypted bit streams accurately in an efficient way, and in the former case we may even determine the content of compressed files with certainty. The accuracy of the proposed methodology is highly dependent on the size of the investigated packets, decreasing as the packet size decreases. Moreover, our threshold-based method achieves higher accuracy (see Figure 1) with more efficiency than the state-of-the-art [17] (k-NN and convolutional Neural networks), since they have a complexity of at least $O(n^2)$ whilst our method has linear cost $O(n)$. Moreover, in most cases we need only to compare one feature (and not n) to classify a bit stream (e.g. if the chi-square test fails, the file is discarded and there is no need to compute the rest of tests). In addition, results are reproducible since the variability of threshold values is almost negligible, so that a universal threshold can be considered regardless of data, avoiding costly training procedures (such in the case of Neural networks) and other data-dependent methods.

6   Conclusions

Distinguishing high entropy sources (e.g. encrypted) from compressed streams is a challenge that has been the focus of the research community. Most solutions in the literature achieve high accuracy only when they are able to access all the

exchanged packets and extract patterns from the intercepted traffic, but are inefficient when trying to distinguish between high entropy data [37, 11, 38, 39, 31, 52, 6]. Recently, Hahn et al. [17] introduced a method based on machine learning that manages to distinguish between encrypted and compressed data using random packets and not their complete sequence.

In this article, we introduced a novel threshold-based methodology that uses efficient randomness tests to classify data streams into either compressed or encrypted by investigating each intercepted packet individually. Our approach was evaluated using a statistically sound benchmark we developed, achieving an accuracy between 68.68% (worst case scenario) and 94.72% (and 70.61% using the reference dataset of [17]). The accuracy rate depends on the packet size. We also determined that the randomness of compressed files has a strong dependence on the input file-type and we analysed their behaviour. We were able to conclude that (compressed) binary, text and video files can be easily detected by our system. Moreover, our method is more efficient than other competing state-of-the-art works and enables real-time traffic classification.

In the future, we plan to: (i) refine this method to further increase its accuracy; especially for low packet sizes, and (ii) enable more accurate content classification of compressed files, to improve pro-active security and specific file-type detection. Evaluation will also be carried in real-time environment, such as those of the authors' institutions.

.1 Randomness Tests

As this work focuses on the randomness analysis of data streams to determine whether they are encrypted, we will now briefly describe widely used randomness evaluation methods in the literature.

Entropy. According to Information Theory [44], the entropy measures the unpredictability of data, given an information source. Therefore, the higher the entropy, the higher the randomness of data.

Chi-square test. This is one of the prevalent tests used to compute the randomness of a data stream. The test computes how much two given data samples differ from each other. It is used to test whether a set of data follows a specific distribution with a degree of confidence. Usually, the chi-square outcomes are represented as an absolute value and a confidence percentage $\chi\%$, which estimates the frequency that a truly random sequence exceeds this value by chance. There are three main possibilities [8]:

1. $1\% > \chi\%$ or $\chi\% > 99\%$, the stream is not random.

2. $1\% < \chi\% < 5\%$ or $95\% < \chi\% < 99\%$, the stream is "suspected" to be random.

3. $5\% < \chi\% < 10\%$ or $90\% < \chi\% < 95\%$, the stream is likely not to be random.

Auto-correlation test. This test computes inner correlations that can reveal a cyclic pattern or periodic behaviours. The randomness of the data is checked

by calculating auto-correlation for the values of the data stream at different time lags [15]. Auto-correlation values close to zero show a highly random pattern, while high variations from zero show non-random behaviour.

Average test. This simple test computes the summation of the values of the bytes. For a random data stream, the value should be close to 127.5.

Kolmogorov-Smirnov test. This test is an alternative to the chi-square test and quantifies the maximum absolute difference between two empirical cumulative distribution functions as a measure of disagreement [29].

Anderson-Darlong test. This test is similar to Kolmogorov-Smirnov test, however its reliability is higher, especially for small data streams, as reported in [40].

Monte Carlo value for π test. This method computes an approximation to the number π using the bit values of the data stream [28]. The closer to π, the more random is the data stream.

Monobit test. This test tries to estimate the balance of ones and zeros in a bit stream. Practically, given a sequence of n bits, we test whether:

$$erfc(\frac{|\#zeroes - \#ones|}{n\sqrt{2}}) < 0.01$$

Poker test. This test splits the bit stream in segments of 4 bits. Each segment, when converted to an integer belongs to $[0, 15]$. If we denote as $f(i)$ the number of occurences of each number i then we evaluate:

$$X = \frac{16}{5000} \sum_{i=0}^{15} f(i)^2 - 5000$$

The test is passed if $2.16 < X < 46.17$.

Runs test. This test evaluates how often we see the same consecutive bits (a run), e.g. all ones in a raw, in a bit stream. To this end, this test we count the length of each possible run up to length 6 and check whether it conforms to a specific pattern.

Long runs test. In this test we try to determine whether there are runs of length more than 25.

Diehard tests. The Diehard tests are a collection of statistical and mathematical tests for measuring the quality of a random number generator [32].

TestU01 tests. Similarly to Diehard, the TestU01 suite is a collection of random number generator methods as well as a collection of tests to measure their quality [23].

FIPS-2-140 test. The National Institute of Standards and Technology (NIST) proposed a set of four empirical tests to analyse the randomness of binary data streams [1]. In our experiments, we use the FIPS-2-140 cryptographic module test with minimum block size (i.e. 20,000 bits) and, hence, tests are applied independently to each data block. The set consists of the monobit, the poker, the runs, and the long runs tests presented above.

SP 800-22 test suite. NIST later extended the FIPS-2-140 test suite for random and pseudorandom number generators with more tests in SP 800-22 [18].

Acknowledgments

References

[1] NIST SP 800-22, 2014.

[2] Riyad Alshammari, Peter I. Lichodzijewski, Malcolm Heywood, and A. Nur Zincir-Heywood. Classifying ssh encrypted traffic with minimum packet header features using genetic programming. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, pages 2539–2546, New York, NY, USA, 2009. ACM.

[3] Riyad Alshammari and A Nur Zincir-Heywood. Can encrypted traffic be identified without port numbers, ip addresses and payload inspection? *Computer networks*, 55(6):1326–1350, 2011.

[4] D. J. Arndt and A. N. Zincir-Heywood. A comparison of three machine learning techniques for encrypted network traffic analysis. In *2011 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, pages 107–114, April 2011.

[5] Roni Bar Yanai, Michael Langberg, David Peleg, and Liam Roditty. Real-time classification for encrypted traffic. In Paola Festa, editor, *Experimental Algorithms*, pages 373–385, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[6] Gregory Conti, Sergey Bratus, Anna Shubina, Benjamin Sangster, Roy Ragsdale, Matthew Supan, Andrew Lichtenberg, and Robert Perez-Alemany. Automated mapping of large binary objects using primitive fragment type classification. *digital investigation*, 7:S3–S12, 2010.

[7] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde. Analyzing android encrypted network traffic to identify user actions. *IEEE Transactions on Information Forensics and Security*, 11(1):114–125, Jan 2016.

[8] RalphB D'Agostino. *Goodness-of-fit-techniques*. Routledge, 2017.

[9] Alberto Dainotti, Walter De Donato, and Antonio Pescapé. Tie: A community-oriented traffic classification platform. In *International Workshop on Traffic Monitoring and Analysis*, pages 64–74. Springer, 2009.

[10] Peter Dorfinger, Georg Panholzer, and Wolfgang John. Entropy estimation for real-time encrypted traffic identification (short paper). In *International Workshop on Traffic Monitoring and Analysis*, pages 164–171. Springer, 2011.

[11] Peter Dorfinger, Georg Panholzer, Brian Trammell, and Teresa Pepe. Entropy-based traffic filtering to support real-time skype detection. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, pages 747–751. ACM, 2010.

[12] Holger Dreger, Anja Feldmann, Michael Mai, Vern Paxson, and Robin Sommer. Dynamic application-layer protocol analysis for network intrusion detection. In *15th USENIX security symposium*, pages 257–272. USENIX Association, 2006.

[13] James P Early, Carla E Brodley, and Catherine Rosenberg. Behavioral authentication of server flows. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 46–55. IEEE, 2003.

[14] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. Offline/realtime traffic classification using semi-supervised learning. *Perform. Eval.*, 64(9-12):1194–1213, October 2007.

[15] Bhaskar Kumar Ghosh and Pranab Kumar Sen. *Handbook of sequential analysis*. CRC Press, 1991.

[16] John Haggerty and Mark Taylor. Forsigs: Forensic signature analysis of the hard drive for multimedia file fingerprints. In *IFIP International Information Security Conference*, pages 1–12. Springer, 2007.

[17] Daniel Hahn, Noah Apthorpe, and Nick Feamster. Detecting compressed cleartext traffic from consumer internet of things devices. *arXiv preprint arXiv:1805.02722*, 2018.

[18] Alan Heckert, James Dray, and San Vo. Andrew rukhin, juan soto, james nechvatal, miles smid, elaine barker, stefan leigh, mark levenson, mark vangel, david banks. *NIST Special Publication*, 800:22, 2001.

[19] Amir R. Khakpour and Alex X. Liu. An information-theoretical approach to high-speed flow nature identification. *IEEE/ACM Trans. Netw.*, 21(4):1076–1089, August 2013.

[20] Jawad Khalife, Amjad Hajjar, and Jesus Diaz-Verdejo. A multilevel taxonomy and requirements for an optimal traffic-classification model. *Netw.*, 24(2):101–120, March 2014.

[21] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

[22] Y. Kumano, S. Ata, N. Nakamura, Y. Nakahira, and I. Oka. Towards real-time processing for application identification of encrypted traffic. In *2014 International Conference on Computing, Networking and Communications (ICNC)*, pages 136–140, Feb 2014.

[23] Pierre L'Ecuyer and Richard Simard. Testu01: Ac library for empirical testing of random number generators. *ACM Transactions on Mathematical Software (TOMS)*, 33(4):22, 2007.

[24] Suchul Lee, Hyunchul Kim, Dhiman Barman, Sungryoul Lee, Chong-kwon Kim, Ted Kwon, and Yanghee Choi. Netramark: a network traffic classification benchmark. *ACM SIGCOMM Computer Communication Review*, 41(1):22–30, 2011.

[25] Binglong Li, Qingxian Wang, and Junyong Luo. Forensic analysis of document fragment based on svm. In *null*, pages 236–239. IEEE, 2006.

[26] Marc Liberatore and Brian Neil Levine. Inferring the source of encrypted http connections. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 255–263. ACM, 2006.

[27] J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, Jan 1991.

[28] Andrew W Lo and A Craig MacKinlay. The size and power of the variance ratio test in finite samples: A monte carlo investigation. *Journal of econometrics*, 40(2):203–238, 1989.

[29] Raul HC Lopes. Kolmogorov-smirnov test. In *International encyclopedia of statistical science*, pages 718–720. Springer, 2011.

[30] Robert Lyda and James Hamrock. Using entropy analysis to find encrypted and packed malware. *IEEE Security & Privacy*, 5(2), 2007.

[31] Paras Malhotra. *Detection of encrypted streams for egress monitoring*. Iowa State University, 2007.

[32] George Marsaglia and Arif Zaman. Monkey tests for random number generators. *Computers & mathematics with applications*, 26(9):1–10, 1993.

[33] A. W. Moore and D. Zuev. Discriminators for use in flow-based classification. *Intel Research*, 2005.

[34] Tadahiko Murata and Hisao Ishibuchi. Moga: multi-objective genetic algorithms. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 1, page 289. IEEE, 1995.

[35] Thuy TT Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76, 2008.

[36] Y. Okada, S. Ata, N. Nakamura, Y. Nakahira, and I. Oka. Application identification from encrypted traffic based on characteristic changes by encryption. In *2011 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR)*, pages 1–6, May 2011.

[37] Julien Olivain and Jean Goubault-Larrecq. Detecting subverted cryptographic protocols by entropy checking. *Laboratoire Spécification et Vérification, ENS Cachan, France, Research Report LSV-06-13*, 2006.

[38] Liam Paninski. Estimation of entropy and mutual information. *Neural computation*, 15(6):1191–1253, 2003.

[39] Liam Paninski. Estimating entropy on m bins given fewer than m samples. *IEEE Transactions on Information Theory*, 50(9):2200–2203, 2004.

[40] Nornadiah Mohd Razali, Yap Bee Wah, et al. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of statistical modeling and analytics*, 2(1):21–33, 2011.

[41] Vassil Roussev and Simson L Garfinkel. File fragment classification-the case for specialized approaches. In *Systematic Approaches to Digital Forensic Engineering, 2009. SADFE'09. Fourth International IEEE Workshop on*, pages 3–14. IEEE, 2009.

[42] Salvatore Ruggieri. Efficient c4. 5 [classification algorithm]. *IEEE transactions on knowledge and data engineering*, 14(2):438–444, 2002.

[43] Adi Shamir and Nicko van Someren. Playing 'hide and seek' with stored keys. In Matthew Franklin, editor, *Financial Cryptography*, pages 118–124, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[44] Claude E Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.

[45] Salvatore J Stolfo, Ke Wang, and Wei-Jen Li. Fileprint analysis for malware detection. *ACM CCS WORM*, 2005.

[46] Qixiang Sun, Daniel R Simon, Yi-Min Wang, Wilf Russell, Venkata N Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *null*, page 19. IEEE, 2002.

[47] S Momina Tabish, M Zubair Shafiq, and Muddassar Farooq. Malware detection using statistical analysis of byte-level file content. In *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*, pages 23–31. ACM, 2009.

[48] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security*, 13(1):63–78, Jan 2018.

[49] Petr Velan, Milan Čermák, Pavel Čeleda, and Martin Drašar. A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management*, 25(5):355–374, 2015.

[50] Ke Wang and Salvatore J Stolfo. Anomalous payload-based network intrusion detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 203–222. Springer, 2004.

[51] Ruoyu Wang, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. Steal this movie: Automatically bypassing DRM protection in streaming media services. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 687–702, Washington, D.C., 2013. USENIX.

[52] Y. Wang, Z. Zhang, L. Guo, and S. Li. Using entropy to classify traffic more deeply. In *2011 IEEE Sixth International Conference on Networking, Architecture, and Storage*, pages 45–52, July 2011.

[53] Michael Weber, Matthew Schmid, Michael Schatz, and David Geyer. A toolkit for detecting and analyzing malicious software. In *Proceedings of the 18th Annual Computer Security Applications Conference*, ACSAC '02, pages 423–, Washington, DC, USA, 2002. IEEE Computer Society.

[54] Andrew M White, Srinivas Krishnan, Michael Bailey, Fabian Monrose, and Phillip A Porras. Clear and present data: Opaque traffic and its security implications for the future. In *NDSS*, 2013.

[55] Daniel Wood, Noah Apthorpe, and Nick Feamster. Cleartext data transmissions in consumer iot medical devices. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*, pages 7–12, New York, NY, USA, 2017. ACM.

[56] Charles V Wright, Fabian Monrose, and Gerald M Masson. On inferring application protocol behaviors in encrypted network traffic. *Journal of Machine Learning Research*, 7(Dec):2745–2769, 2006.

[57] Feipeng Yan, Ming Xu, Tong Qiao, Ting Wu, Xue Yang, Ning Zheng, and Kim-Kwang Raymond Choo. Identifying wechat red packets and fund transfers via analyzing encrypted network traffic. In *IEEE International Conference On Trust, Security And Privacy In Computing And Communications*, pages 1426–1432. IEEE, 2018.

[58] Ibrar Yaqoob, Ibrahim Abaker Targio Hashem, Arif Ahmed, SM Ahsan Kazmi, and Choong Seon Hong. Internet of things forensics: Recent advances, taxonomy, requirements, and open challenges. *Future Generation Computer Systems*, 92:265–275, 2019.

[59] Like Zhang and Gregory B White. An approach to detect executable content for anomaly based network intrusion detection. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8. IEEE, 2007.

[60] Qinghua Zhang, Douglas S Reeves, Peng Ning, and S Purushothaman Iyer. Analyzing network traffic to detect self-decrypting exploit code. In *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 4–12. ACM, 2007.

[61] Yin Zhang and Vern Paxson. Detecting backdoors. In *USENIX Security Symposium*, 2000.