Neural Computing and Applications (2022) 34:20379–20393
https://doi.org/10.1007/s00521-022-07586-7

ORIGINAL ARTICLE

Reliable detection of compressed and encrypted data

Fabio De Gaspari[1] · Dorjan Hitaj[1] · Giulio Pagnotta[1] · Lorenzo De Carli[2] · Luigi V. Mancini[1]

Received: 8 February 2022 / Accepted: 28 June 2022 / Published online: 24 July 2022

Abstract
Several cybersecurity domains, such as ransomware detection, forensics and data analysis, require methods to reliably identify encrypted data fragments. Typically, current approaches employ statistics derived from byte-level distribution, such as entropy estimation, to identify encrypted fragments. However, modern content types use compression techniques which alter data distribution pushing it closer to the uniform distribution. The result is that current approaches exhibit unreliable encryption detection performance when compressed data appear in the dataset. Furthermore, proposed approaches are typically evaluated over few data types and fragment sizes, making it hard to assess their practical applicability. This paper compares existing statistical tests on a large, standardized dataset and shows that current approaches consistently fail to distinguish encrypted and compressed data on both small and large fragment sizes. We address these shortcomings and design EnCoD, a learning-based classifier which can reliably distinguish compressed and encrypted data. We evaluate EnCoD on a dataset of 16 different file types and fragment sizes ranging from 512B to 8KB. Our results highlight that EnCoD outperforms current approaches by a wide margin, with accuracy ranging from $\sim 82\%$ for 512B fragments up to $\sim 92\%$ for 8KB data fragments. Moreover, EnCoD can pinpoint the exact format of a given data fragment, rather than performing only binary classification like previous approaches.

Keywords Machine learning · Cybersecurity · Forensics · Encryption detection

1 Introduction

Reliable detection of encrypted data fragments is an important primitive with many applications to security and digital forensics. For instance, ransomware detection algorithms use estimates of write-operations' data randomness to quickly identify evidence of malicious encryption processes [1–4]. When performing digital forensic analysis of hard drives and phones, it is oftentimes important to identify encrypted archives [5]. Finally, encryption detection is widely used in network protocol analysis [6, 7].

A popular approach to address this problem is to estimate the Shannon entropy of the sequence of interest using the Maximum Likelihood Estimator (MLE): $\hat{H}_{MLE}$. This approach leverages the observation that the distribution of byte values in an encrypted stream closely follows a uniform distribution; therefore, high entropy is used as a proxy for randomness. This estimator has the advantage of being simple and computationally efficient. As non-encrypted digital data are assumed to have low byte-level entropy, the estimator is expected to easily differentiate non-encrypted and encrypted content.

While this approach remains widely used (e.g., [1–4]), a number of works have highlighted its limitations. Modern applications tend to compress data prior to both storage and transmission. Popular examples include the zip compressed file format, and HTTP compression [8] (both using the

✉ Dorjan Hitaj
hitaj.d@di.uniroma1.it

Fabio De Gaspari
degaspari@di.uniroma1.it

Giulio Pagnotta
pagnotta@di.uniroma1.it

Lorenzo De Carli
ldecarli@wpi.edu

Luigi V. Mancini
mancini@di.uniroma1.it

[1] Dipartimento di Informatica, Sapienza Università di Roma, Viale Regina Elena 295, 00161 Rome, Italy, Italy

[2] Department of Computer Science, Worcester Polytechnic Institute, Street, Worcester, Massachusetts 01609, USA

DEFLATE algorithm). As compression removes recurring patterns in data, compressed streams tend to exhibit high Shannon entropy. As a result, compressed data exhibit values of $\hat{H}_{MLE}$ that are close and oftentimes overlapping with those obtained by encryption. In principle, compressed content can be identified by using appropriate parsers. However, many security-related applications, such as ransomware detection, traffic analysis and digital forensics, generally do not have access to whole-file information, but rather work at the level of *fragments* of data. In these settings, the metadata that is required by parsers is not present or is incomplete [9]. Given this issue, a number of works have been looking at alternative tests to distinguish between encrypted and compressed content [10–16]. While these works have the potential to be useful, there has been limited evaluation of their performance on a standardized dataset. Consequently, there is no clear understanding of how these approaches: (i) fare on a variety of compressed file formats and sizes, and (ii) compare to each other. The potential negative implications are significant: the use of ineffective techniques for identifying encrypted content can hinder the effectiveness of ransomware detectors [17, 18], and significantly limit the capability of forensic tools.

Our work compares state-of-the-art approaches on a large dataset of different data types and fragment sizes. We find that, while more useful than entropy estimates, current approaches fail to achieve consistently high accuracy. Tests based on byte-value distribution, such as χ^2, can distinguish some encrypted and compressed content, but have accuracy issues (ref. Sect. 5). Such tests, in a sense, "collapse" the entire distribution to a single scalar value, losing information concerning the shape of the distribution. It is therefore natural to ask if Deep Neural Networks (DNNs) can improve such results due to the fact that DNNs can consider the entire discrete distribution (modeled as a feature vector), and can learn to recognize complex distributions [19, 20]. To address this, we propose EnCoD (**En**cryption/**Co**mpression **D**istinguisher), a novel neural network-based approach. Our evaluation shows that EnCoD outperforms existing approaches for most considered file types, over all considered fragment sizes.

EnCoD can distinguish between compressed and encrypted data fragments as small as 512B with 86%accuracy. The accuracy increases to up to 94% when distinguishing between encrypted and purely compressed data (i.e.,zip, gzip), and up to 100% in the case of compressed application data fragments (e.g., pdf, jpeg, mp3) when the fragment size is 8KB. Furthermore, we investigate the applicability of robust feature extraction techniques such as autoencoders to our architecture, in an effort to understand whether feature vector pre-processing can lead to increased performance compared to a plain neural network (NN) architecture in this domain.

This paper revises and extends our previous conference paper [21], by considering a larger and more diverse dataset of file fragments and evaluating the effectiveness of data pre-processing on accuracy. Overall, we make the following contributions:

- We review and categorize existing literature on the topic of distinguishing compressed and encrypted data fragments.
- We build and make available to the community a large, standardized dataset of data fragments of different sizes from 16 different data formats.[1]
- We systematically evaluate and compare state-of-the-art approaches on our dataset for different fragment formats and sizes.
- We propose a new neural-network-based approach and show that it outperforms current state-of-the-art tests in distinguishing encrypted from compressed content for most considered formats, over all considered fragment sizes.
- We propose a new multi-class classifier that can label a fragment with high accuracy as encrypted data, general-purpose compressed data (zip/gzip/rar/bz2), or one of multiple application-specific compressed data (png, jpeg, pdf, mp3, office, video).
- We investigate the effectiveness of data pre-processing techniques such as autoencoders for our architecture, and show that plain neural network models outperform these approaches in the considered domain.
- We thoroughly discuss the implications of our findings and effectiveness of the evaluated approaches in distinguishing compressed and encrypted data.

The rest of this paper is structured as follows: Sect. 2 provides background on entropy estimation and its applications. Section 3 reviews existing approaches to the problem. Section 4 presents and evaluates a novel approach to the problem, based on deep learning. Section 5 evaluates the performance of the considered approaches, discussing their strengths and limitations. Section 6 discusses the implications of our findings. Section 7 discusses related work and Sect. 8 concludes the paper.

[1] The full dataset is available at https://drive.google.com/file/d/1IDNv3U1hRILXblwT9fI3G-D8hJquiequ.

2 Background

Determining the format of a particular data object (e.g., a file in permanent storage, or an HTTP object) is an extremely common operation. Under normal circumstances, it can be accomplished by looking at content metadata or by parsing the object. Things get more complicated, however, when no metadata is available and the data object is corrupted or partly missing. In this paper, we focus on detection of *encrypted content* and, in particular, on distinguishing between encrypted and compressed data fragments. We begin by examining relevant applications of encryption detection primitives.

2.1 Ransomware detection

Ransomware encrypts user files with the aim of making them unusable for the user. It then presents a prompt asking the user to pay a ransom in order to receive the decryption key. Ransomware attacks can cause significant financial damage to organizations [22–24].

Mitigating a ransomware infection requires rapid detection and termination of all ransomware processes. A number of approaches based on *behavioral process analysis* have been proposed for this purpose [1–4]. These approaches typically rely on a classifier trained on various process-related features to distinguish benign and ransomware processes. Virtually all proposed behavioral detectors use entropy of file write operations as one of the key features, based on the insight that frequently writing encrypted content is a characteristic behavioral fingerprint of ransomware. Entropy is typically estimated using $\hat{H}_{MLE}$. In several approaches, entropy is estimated on the content of individual file writes [1, 3, 4], therefore the estimation procedure has only access to partial file fragments.

2.2 Forensics

Digital forensics oftentimes involves analysis of phone [25] or PC [9] storage that has been corrupted, or uses an unknown format. Therefore, forensic techniques attempt to recover data of interest (contacts, pictures, etc.) by searching for blocks with recognizable structure. These techniques typically only have access to data fragments, rather than whole files.

Encrypted and compressed data represent a corner case, as they exhibit a complete lack of structure. Still, detecting such content may be important in data recovery operations (e.g., if sensitive data is known to have been encrypted). Distinguishing between compressed and encrypted blocks is notoriously difficult, and some forensic approaches label data as *"compressed or encrypted,"* without attempting to pinpoint which one of the two it is [5].

2.3 Network traffic analysis

Network traffic analysis examines flows in/out of a network to identify security issues. Regulations (e.g., HIPAA in the U.S.) and best practices expect sensitive data to be encrypted in transit; therefore, entropy-based analyzers have been proposed to ensure that all traffic leaving a monitored network is encrypted [7]. Another application is reverse-engineering of network protocols used by malware. It has been observed [6] that malware protocols may mix encrypted and non-encrypted content within the same message. Encryption detection primitives can be applied to break messages into encrypted and non-encrypted fields.

In both cases above, encryption detectors have partial visibility on the data stream and can only access fragments of data (e.g., an encrypted stream broken into individual packets), rather than whole data objects.

2.4 Challenges

In the three domains above, the use of Shannon entropy has been proposed in order to identify encrypted content. Entropy is used to measure the *information content* of a byte sequence; highly structured data exhibit low entropy, while unstructured data—such as a randomly distributed sequence—have high entropy. Therefore, an entropy estimate can be used as a proxy for how close a sequence of bytes is to being randomly distributed. Most encryption algorithms output ciphertexts whose byte-value distributions tend to follow a uniform distribution. As a result, an encrypted bytestream will almost invariably exhibit high entropy.

One of the most common approaches to entropy estimation is the maximum likelihood estimator $\hat{H}_{MLE} = -\sum_{i=0}^{255} f_i log_2(f_i)$, where f_i is the frequency of byte value i in the sequence. The entropy range is $[0-8]$. The frequency f_i of byte value i, which is measurable, is used in place of the probability $P(i)$ of that value occurring, which is unknown. This approach is commonly used in some of the applications above (e.g., [1, 7]), due to its simplicity and efficiency.

This reasoning assumes that, while encrypted data has high entropy, non-encrypted data does not. This appears reasonable, as most relevant data types (e.g., text, images, audio) are information-rich and highly structured. However, this assumption does not hold true in modern computing. Modern CPUs can efficiently decompress data for processing, and compress it back for storage or transmission; this is oftentimes performed in real time and

transparent to the user. As a result, most formats tend to apply compression [26]. Informally, a good compression algorithm works by identifying and removing recognizable structures from the data stream; as a result, compressed data tend to exhibit high entropy. In practice, this fact compromises the ability of entropy-based detectors to distinguish encrypted and non-encrypted, compressed content.

2.4.1 Entropy estimates for common data formats

In order to substantiate the claim above, we computed entropy estimates using a dataset consisting of 10,000 file fragments. The dataset covers various popular file formats and AES-256-encrypted data. We considered multiple fragment sizes, from 512B to 8KB (details in Sect. 4). Figure 1 summarizes the distribution of estimated entropy values for eight different formats with block size 2048 (some ranges truncated for clarity). Results for other block sizes were qualitatively similar. As illustrated in Fig. 1, both general-purpose (e.g., zip, rar) and domain-specific (e.g., jpeg, mp3) compression algorithms result in data which exhibits entropy whose ranges are overlapping with that of encrypted content (enc). The only format that can be unambiguously distinguished is png. Even so, png still overlaps with various other formats. Interestingly, utilities that create and modify data in zip, gzip, office and png format internally all use the DEFLATE algorithm for compression: the differences in entropy are likely due to differences in file structure and algorithm implementation.

Due to the limits of entropy estimation, the attention of the community has been increasingly focusing on alternative measures that can more precisely estimate whether data follow a random distribution. However, no comprehensive review of such approaches exists. In the next section, we review state-of-art approaches, while we evaluate and compare them in Sect. 5.

3 Review of existing techniques

This section reviews three state-of-the-art approaches to distinguish encrypted and compressed content: the NIST suite, χ^2 and HEDGE [15]. Strictly speaking, these approaches test the *randomness* of a string of bytes, and make no attempt to determine its type. However, due to their high precision they can be used to distinguish true pseudorandom (encrypted) sequences and compressed ones which, while approximating a randomly generated stream, maintain structure.

The NIST suite and χ^2 are standard statistical tests for identifying randomly-distributed data. HEDGE is a recently proposed statistical approach which shows promising results. HEDGE is a combination of a subset of the NIST tests and two forms of χ^2 tests. Note that, despite the inclusion of HEDGE, we decided to also report separate results for NIST and χ^2 due to the fact that those are designed to be, and oftentimes are, used as standalone tests.

3.1 NIST SP800-22

The NIST SP800-22 specification [27] describes a suite of tests whose intended use is to evaluate the quality of random number generators. The suite consists of 15 distinct tests, which analyze various structural aspects of a byte sequence. These tests are commonly employed as a benchmark for distinguishing compressed and encrypted content (e.g., [15, 16]). Each test analyzes a particular property of the sequence, and subsequently applies a test-specific decision rule to determine whether the result of the analysis suggests randomness or not. When using the NIST suite for discriminating random and non-random sequences, an important question concerns aggregation of the results of individual tests. Analysis of the tests [27] suggests that they are largely independent. Given this observation, and the intrinsic complexity of *a priori* defining a ranking between the tests, we use a *majority voting* approach. In other words, we consider a fragment to be random (and therefore encrypted) when the majority of tests considers it so. Since some of the tests require a block

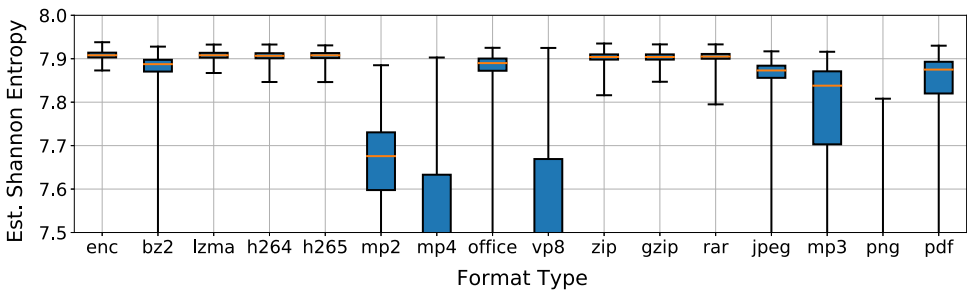


Fig. 1 Entropy ranges for common formats (2048B blocks)

length much bigger than the ones we use for our smaller fragment sizes, we did not consider in the voting the tests that cannot be executed.

3.2 χ^2 Test

The χ^2 test is a simple statistical test to measure goodness of fit. It has been widely applied to distinguish compressed and encrypted content [10, 13, 15]. Given a set of samples, it measures how well the distribution of such samples follows a given distribution. Mathematically, the test is defined as:

$$\chi^2 = \sum_{i=0}^{255} \frac{(N_i - E_i)^2}{E_i}$$

where N_i is the actual number of samples assuming value i, and E_i is the expected number of samples assuming value i according to the known distribution of interest. Since the distribution being evaluated for goodness of fit is the discrete uniform distribution, $\forall i E_i = L/256$, where L is the particular fragment length being considered. The results of the test can be interpreted using either a fixed threshold, or a confidence interval [15].

3.3 HEDGE

HEDGE [15] simultaneously incorporates three methods to distinguish between compressed and encrypted fragments: χ^2 test with absolute value, χ^2 with confidence interval and a subset of NIST SP800-22 test suite. Out of the NIST SP800-22 test suite HEDGE incorporates 3 tests: *frequency within block test*, *cumulative sums test*, and *approximate entropy test*. These tests were selected due to (i) their ability to operate on short byte sequences, and (ii) their reliable performance on a large and representative dataset. In the HEDGE detector, the threshold of the number of the above-mentioned NIST SP800-22 tests failed is set to 0. For the χ^2 with absolute value test, the thresholds are precomputed for each of the considered packet sizes, by considering the average and its standard deviation. For χ^2 with confidence interval, the $\chi\%$ interval is $(\chi\% > 99\% || \chi\% < 1\%)$. For classifying the content of a packet, HEDGE applies the three randomness tests to the input data. Data are considered random only if it passes all tests.

4 EnCoD: a learning-based approach

Past work and our own evaluation suggest that tests based on byte-value distribution, such as χ^2, can distinguish some encrypted and compressed content, but have accuracy issues (ref. Sect. 5). Such tests, in a sense, "collapse" the entire distribution to a single scalar value, losing information concerning the shape of the distribution. It is therefore natural to ask if Deep Neural Networks (DNNs) can improve such results. DNNs can consider the entire discrete distribution (modeled as a feature vector) and can learn to recognize complex distributions [19].

In order to evaluate the potential of DNNs, we designed EnCoD, a set of two distinct neural network-based approaches for distinguishing encryption and compression.

4.1 Model architecture #1: binary classifiers

Our first model is a binary classifier trained to distinguish a single specific compressed format from encrypted content. It may be used in cases where only one compressed format is known to exist in the dataset (e.g., detecting writes of encrypted data performed by a potential ransomware on image files vs legitimate writes of JPEG-compressed data). We explored several alternative architectures for this application, and we found that the structure depicted in Fig. 2a provides the best performance. The binary-classifier architecture consists of 4 fully-connected layers with dimensions as shown in the figure. We initialize the model weights using Glorot uniform [28]. The activation function is ReLU for the first three layers, followed by a softmax on the output layer. We used a batch size of 64 for training our model. Each hyperparameter has been chosen using grid search. We used the same procedure also for the model described in Sect. 4.2.

4.2 Model architecture #2: content-type detector

In many applications, a classifier may encounter more than one type of compressed data. Furthermore, it may be important to determine the specific type being encountered. To support these use cases, we design a *content-type detector*: a multi-class classifier that can determine whether a given fragment is encrypted, or belongs to one of multiple known compressed formats. We explored several designs for the neural network, converging to the model depicted in Fig. 2b. Its architecture consists of 5 fully-connected layers with dimensions as shown in the figure. We initialize model weights using LeCun normal [29]. Differently from the binary models, this multi-class classifier seemed prone to the dying neuron problem associated with the ReLU activation function [30]. We therefore opted for the SelU activation function [31] for the first 4 layers, followed by a softmax on the output layer. We used a batch size of 64 instances for training.

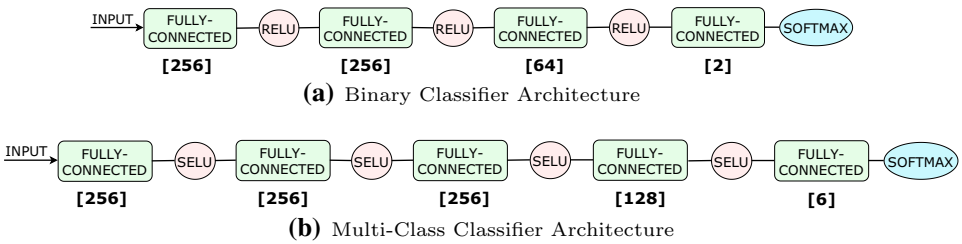


Fig. 2 Neural network architectures

4.3 Model architecture extension: autoencoders

In addition to the architectures described in Sects. 4.1 and 4.2, we design a third model architecture that makes use of autoencoders. Autoencoders (AE) are NN models that are trained to compress an N-dimensional feature vector into an M-dimensional latent representation, with $M < N$, and then reconstruct the original input from the latent representation. AE are composed of two parts: the encoder, which compresses the original feature vector into the latent representation, and the decoder, which takes the output of the encoder and reconstructs the input. AE are generally used to extract robust features from a feature vector and aid classification [32, 33].

In our third architecture, we use the encoder portion of a trained autoencoder to pre-process samples into a compressed latent representation, which is then used as input by a fully-connected NN. We design two different set of NN with two different encoders:

- AE architecure #1: The encoder is composed of 3 fully-connected layers of size $256 - 200 - 128$, respectively, followed by a 3-layer fully-connected NN of size $128 - 64 - 2$.
- AE architecture #2: The encoder is composed of 4 fully-connected layers of size $256 - 156 - 128 - 64$, respectively, followed by a 3-layer fully-connected NN of size $64 - 64 - 2$.

Both AE are trained for 25 epochs using the *Adam* optimizer and *Mean Squared error* as loss function. We initialize the weights using Glorot uniform for the NN model and uniform distribution for the AE. The activation function used is ReLU for all layers except for the output ones, which use softmax in the NN and sigmoid in the AE. We used a batch size of 64 for training the NN and 128 for the AE. The hyperparameters were chosen using grid search.

4.4 Fragment dataset

We built a dataset of 400 million encrypted and compressed fragments from 16 different data formats. For the compressed data, we selected a set of formats covering common, popular content types. To generate the encrypted data fragments, we used the AES cipher in CBC mode implemented by the PyCryptodome library.[2] We chose AES because it is the most widely used and well-known symmetric cipher, representative of modern ciphers which result in byte streams consistently close to random.

In constructing the dataset, we focused on ensuring a diversity of compressed formats, rather than compression algorithms. While algorithms such as DEFLATE are used in multiple compression formats, they are generally used with different parameters and/or embed compressed data in different ways within the compressed archive. Consequently, compressed archives created with different formats tend to differ considerably from each other even when using the same underlying compression algorithm. This observation is empirically confirmed by our evaluation in Sect. 5. Finally, our dataset does not include data which is both compressed and encrypted, and we ensured such data are not present in the dataset. The dataset is comprised of the following data types:

1. **AES encrypted data (enc)**. We used the AES implementation provided by the Cryptodome Python library. AES was configured to use CBC mode with 256-bit keys, with a random IV generated before encrypting each file.
2. **zip, gzip, rar, bz2, xz:** DEFLATE, rar, Burrows–Wheeler and Lempel–Ziv–Markov compressed data. These algorithms are among the most used for generic file compression, with DEFLATE and rar being akin to de-facto standards. DEFLATE is also widely used for documents (such as in the Microsoft office file formats), and network applications (e.g., HTTP header compression).
3. **png and jpeg images:** png is used for lossless image compression; it internally uses DEFLATE, but png files present a structure that is different from that of zip files. jpeg uses DCT-based lossy compression.
4. **mp3 audio files:** MP3 compressors use a psychoacoustic model to remove inaudible frequencies from audio data, and compress the resulting data using a lossy algorithm based on the modified-DCT transform.
5. **pdf documents:** PDF is an office format used for document exchange and form filling. Internally, PDF files consist of a tree of objects that can be compressed

[2] https://pycryptodome.readthedocs.io.

using a variety of techniques. In practice, most PDF documents contain a large amount of compressed content, such as embedded images.

6. **Microsoft office files:** Ms office is one of the most used tool suites for office productivity. Internally, office files use the deflate algorithm for compression.
7. **h264, h265, mpeg2, mpeg4 and vp8 video formats:** h264 is one of the most widely used video codecs today, being the recommended codec for Youtube videos. H265 is the successor of h264 and substantially improves compression rate while maintaining the same video quality. Mpeg4 (Xvid codec) is a format that was widely used before h264. Vp8 and mpeg-2 are fairly dated video codecs that are not often used anymore, but there still exists old contents using them.

4.4.1 Fragment generation process

We generate fragments from a dataset of files:

- **zip/gzip/rar/bz2/xz/enc:** we used various textual documents obtained from a 2020 English Wikipedia dump [3]. We created four copies of each file, each of which was either compressed using one of zip, gzip, rar, bz2, xz utilities (with default parameters), or encrypted using AES-256.
- **png:** we crawled $\sim 116,000$ png images from the web and various repositories [34].
- **jpeg:** we downloaded $\sim 68,000$ images from the Open Images Dataset v5[4] and various online sources.
- **mp3:** we used the FMA medium dataset[5], which contains 25,000 mp3 files.
- **pdf:** we crawled $\sim 3,000$ randomly-selected papers from arXiv.[6]
- **office:** we sampled 4500 Word, 1700 PowerPoint and 1800 Excel files from a private hard drive. For privacy reasons, these files are not included in the provided dataset.
- **video files:** we downloaded a large, h264-encoded video from Youtube and re-encoded it to the remaining formats.

We split each file into fragments of 512B, 1KB, 2KB, 4KB, and 8KB. In an effort to ensure that the dataset remains balanced, we randomly sampled 1M fragments for each fragment size/data-type combination.

[3] English Wikipedia Dump - https://dumps.wikimedia.org/enwiki/.

[4] Open Images Dataset v5 - https://www.figure-eight.com/dataset/.

[5] FMA: A Dataset For Music Analysis - https://github.com/mdeff/fma.

[6] Arxiv Online Repository - http://arxiv.org/.

4.5 Dataset analysis methodology

Statistical tests (NIST, χ^2, HEDGE)

For each fragment size, we randomly selected 10,000 compressed fragments (evenly distributed across the different compressed data types) and 10,000 encrypted fragments. We then executed the tests directly on these fragments.

EnCoD/*Binary Classifiers*

We separately trained and evaluated classifiers for each fragment size. The features that are fed to our models for training/classification are derived from the histograms of the byte values for the observed fragment size. Each feature is the value of the probability density function at a given bin, normalized such that the integral over the range is 1.

We trained the binary classifiers by randomly selecting 3M vectors from the encrypted class and 3M vectors from the data type that we aim to distinguish. We partitioned this dataset into 85% training, 5% development and 10% test. Before fitting the data to the model for training, we applied a MinMax scaler to scale the dataset from the range [0, 1] to the range [0, 2] (range selected via grid search). Scaling helps the ML model to more easily capture minute differences in the inputs, allowing to better distinguish among the classes and converge faster.

EnCoD/*Content-Type Detector*

To train the content-type detectors, for each fragment size we randomly sampled 6M feature vectors consisting of a mix of the considered file types. This dataset was partitioned into training, development and test sets in the same ratios used for the binary classifiers. We also scaled the dataset using the MinMax scaler with the same parameters used above.

EnCoD/*Binary Classifiers with Encoder*

We trained one autoencoder per fragment size for all file types. The autoencoder was trained by randomly sampling 8M feature vectors in total (500,000 per each of the 16 file types). The feature vectors are created following the procedure used to train the plain binary classifiers. The dataset was partitioned into training, development and test sets following the same ratios used for the binary classifiers. The binary models are trained as previously described. The only variation is that the latent representation generated by the trained encoder is used as input by the NN, rather then the original feature vector derived from a given sample.

5 Evaluation

This section comprehensively evaluates existing approaches (see Sect. 3) and our own neural network-based approach, EnCoD. We frame the evaluation in terms of the following comparisons:

1. **Binary classification: all formats.** In Sect. 5.2, we consider the ability of different detectors to discriminate encrypted and compressed data, regardless of the specific compressed format. Results show that our classifier outperforms NIST, χ^2-test and HEDGE for all fragment sizes, with NIST performance approaching that of EnCoD only for large fragment size.
2. **Binary classification by format.** In Sect. 5.3, we break down the performance of χ^2, NIST and HEDGE by compressed format. We also report the performance of our per-format binary classifiers (see Sect. 4.1). The latter perform comparably or better than other tests on all formats but one.
3. **Format fingerprinting.** In Sect. 5.4, we evaluate the accuracy of our multi-class classifier in labeling unknown fragments as the correct compressed format (or as encrypted). Results show that our classifier is able to distinguish the file type with an overall accuracy of 83% for the 2048 byte fragment size. It also achieves high precision, especially on png, jpeg, mp3.
4. **Autoencoder approach.** In Sect. 5.5, we analyze the performance of the encoder-based feature extraction approach in the binary classification task. Our evaluation shows that the plain NN approach outperforms the encoder-based consistently for all considered formats, with the exception of the pdf file type.

5.1 Implementation

We implemented the classifier described in Sect. 4 using the Keras[7] Library for machine learning. For the NIST tests, we used the official implementation.[8] In order to aggregate the NIST tests results, we use the majority voting approach described in Sect. 3.1. In order to label fragments as compressed or encrypted based on χ^2 results, we used the thresholds suggested in the HEDGE paper [15], as the analysis in HEDGE is specifically aimed at producing a dataset-independent threshold for general use. We implemented HEDGE according to the published description [15]. Finally, all experiments were conducted using the dataset described in Sect. 4.

[7] Keras Library for Machine Learning–https://keras.io.

[8] NIST suite–https://csrc.nist.gov.

5.2 Binary classification: all formats

The first part of our evaluation considers the binary classification problem of determining whether a given high-entropy data fragment is compressed or encrypted. Given a fragment, the χ^2 test, HEDGE, and the NIST test suite return whether the fragment's content appears random or not. Therefore, a binary classifier can be constructed from the above-mentioned three tests by simply labeling the random content as encrypted. Our binary classifier used for this evaluation is based on our multi-class classifier. The multi-class classifier labels each fragment either as encrypted, or as one of the fourteen supported compressed formats. Since in this experiment we are only interested in distinguishing encryption and compression, regardless of the type, we combine all compressed type labels into one. Effectively, we consider classification in two labels: (1) a macro-label "compressed," which is comprised of the labels {*zip, rar, gzip, bz2, png, jpeg, mp3, pdf, h264, h265, mpeg2, mpeg4, vp8, office* } and (2) the label "encrypted." We analyze file type fingerprinting accuracy separately in Sect. 5.4. It is worth noting that in experiment shown in Fig. 4 we do not consider the xz file type. This is due to the fact that none of the considered approaches is able to reliably distinguish xz files from encryption for small fragment sizes, as we will see in Sect. 5.3.

The results of this evaluation are depicted in Fig. 4. As we can see, the performance of all classifiers tends to improve as fragment size increases. This behavior is expected, as it is hard to approximate distribution information from short fragment sizes. However, as the fragment size increases, differences in distribution become more apparent and the models can exploit additional information for classification. We further discuss this phenomenon in Sect. 6. In the binary classification task, EnCoD outperforms all the other approaches on all block sizes, with the NIST approach reaching similar performance for $8K$ fragment size only. The χ^2 accuracy remains consistently low across the range of block sizes, while performance for HEDGE increases for larger sizes but remains approximately ~ 10 percentage points lower than EnCoD's. These results suggest that the χ^2 test has an intrinsic difficulty in discriminating non-random content which closely approaches a uniform random distribution.

5.3 Binary classification by format

Our second experiment considers the question of whether some compressed formats are harder than others to distinguish from encrypted content. Such phenomenon may arise due to (i) differences in effectiveness between compression algorithms in removing redundancy (and thus

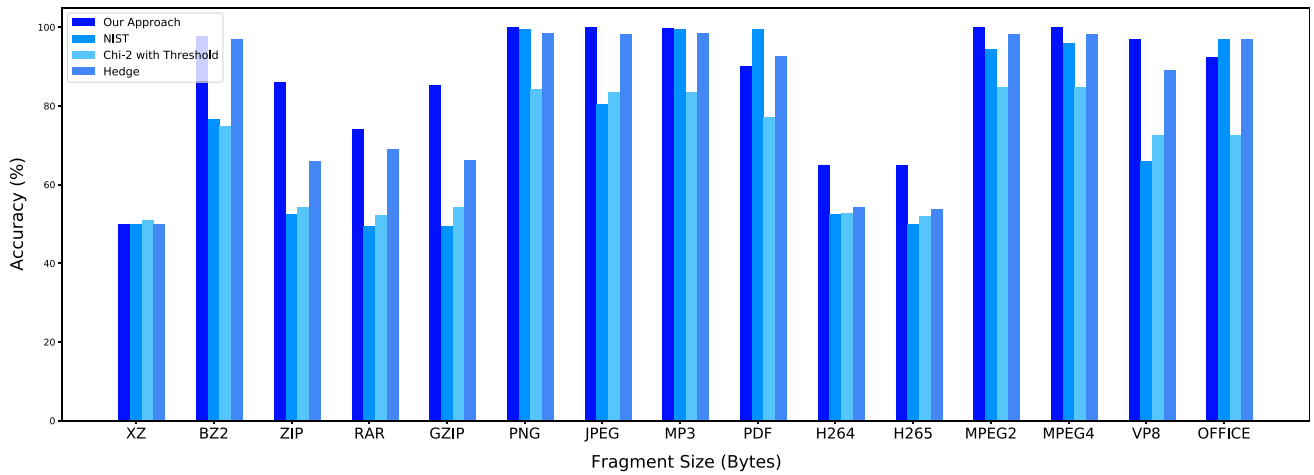


Fig. 3 Performance comparison between our binary-classifier approach, NIST, χ^2 with Threshold and HEDGE. All results are for 2048 Bytes data fragments

structure) from the uncompressed data; and (ii) presence (or absence) of metadata, or other structured information interleaved with compressed data.

In order to answer this question, we break down results for the χ^2-test, NIST suite, HEDGE test and EnCoD by format. In this experiment, for EnCoD we evaluate multiple binary NN classifiers, one per file type (see Sect. 4.1). Each type-specific classifier is trained to distinguish content of a given type from encrypted content (e.g., zip vs encrypted). Note that, while each of these classifiers is trained specifically on one format, the other tests (χ^2, NIST and HEDGE) work the same regardless of the format. Despite this limitation, we believe this experiment to provide an informative analysis, as there are scenarios in which file type is known a priori, and we are interested only in differentiating between that type and encryption. For instance, if we consider a storage dedicated only to pictures backups, we can use a binary classifier (e.g., png/enc) to detect potential ransomware activity encrypting the pictures.

Figure 3 shows the comparison between the four approaches on 2048-byte blocks. Overall, neural network-based classifiers tend to fare better than the other tests, particularly on challenging formats such as zip/gzip, rar and bz2. PDF is the only format on which the NIST and HEDGE tests outperform the neural network classifier, while for the office format EnCoD performs slightly worse but comparably. Interestingly, the χ^2 fares slightly better than NIST on most formats, but its accuracy is significantly worse on formats that are typically easy to distinguish, such as PNG and MP3. We believe this to be due to the fact that the NIST tests look at a richer set of properties beyond byte value distribution, such as a presence of runs and repeated sequences. HEDGE test outperform χ^2 on all file types, while outperforming NIST on most formats, beside PDF, and have similar performance on PNG, MP3, office and all

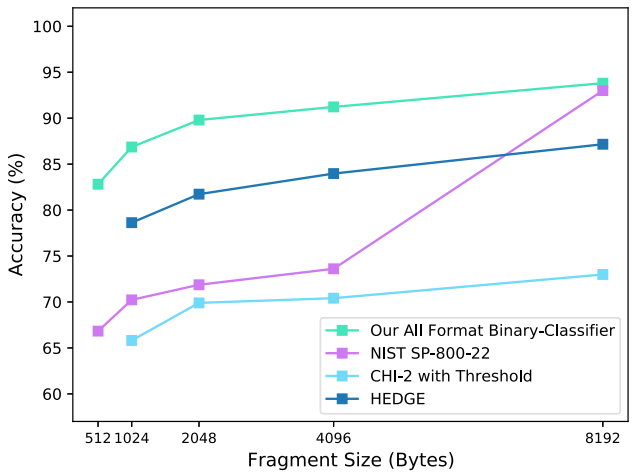
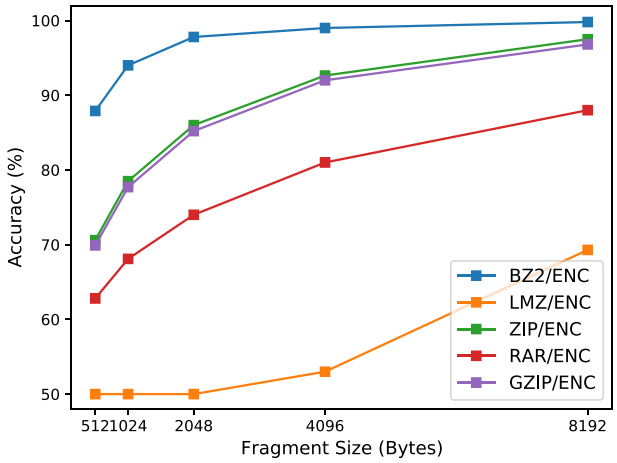


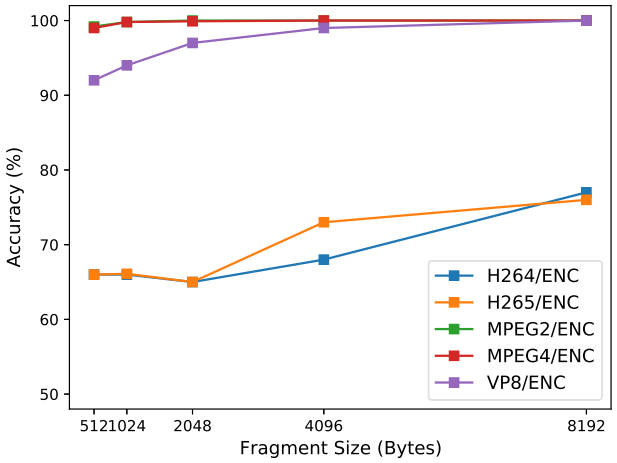
Fig. 4 Performance comparison (binary classification: all formats)

video formats but vp8. Intersingly, all approaches fail to consistently distinguish the xz compressed type (Lempel-Ziv-Markov chain algorithm), with accuracy at $\sim 50\%$. This behavior is indicative of the great compression performance of this algorithm, which pushes compressed data extremely close to a uniform distribution. Due to the inability of any of the considered approaches to consistently detect xz files, this type is not included in any of the aggregated results presented in this evaluation.

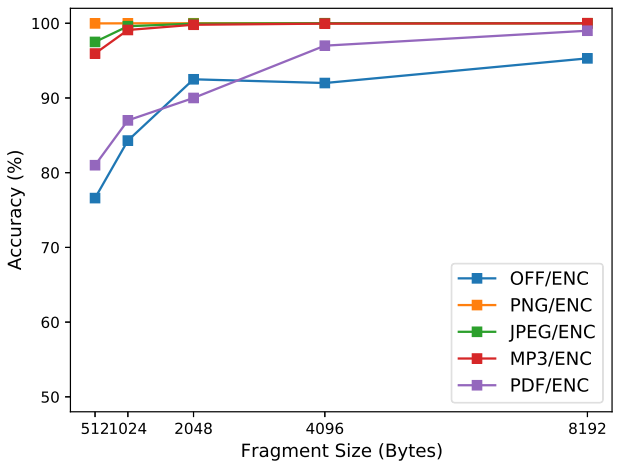
Finally, Fig. 5 presents the performance of all our binary classifiers across all considered fragment sizes. The results are split in three images for clarity, all images have the same scale. These results highlight once again how accuracy increases significantly as block size increases. Indeed, for larger fragment sizes EnCoD can successfully classify xz compressed files from encryption reliably, with approximately 70% accuracy on 8k fragment size, compared to 50% for fragment sizes in the range $[512 - 2k]$.



(a) Performance of our binary classifiers for compressed formats



(b) Performance of our binary classifiers for video formats



(c) Performance of our binary classifiers for miscellaneous formats

Fig. 5 Performance of our binary classification models on different formats and block size ranging from 512 bytes to 8192 bytes

5.4 Format fingerprinting

Our multiclass classifier has the ability to (1) distinguish encrypted and compressed data, and (2) pinpoint the specific format compressed data belong to. This is a significant improvement over the functionality of existing tests, that can only distinguish encryption and compression, but cannot tell the specific format or even generic type of the compressed data. In this section, we analyze the effectiveness of our multi-class classifier in fingerprinting the correct type of compressed content. Since for some type families, we have many subtypes (e.g., for video we have 5 types, for cmp we have 4), while for others only one or two, in this experiment we group some of the classes in macro-labels. Specifically, the office macro-label includes all office file types, the compressed (cmp) macro-labels includes zip, gzip, rar, bz2 and the video macro-label includes h264, h265, mpeg2, mpeg4 and vp8.

Figure 6 shows the confusion matrix for the multi-class classifier. Results indicate that our classifier is able to pinpoint the type of a given sample with consistently high precision for most formats, especially png, mp3, jpeg and encryption. It performs fairly well on the other considered compressed formats such as cmp (which contains a mixture of zip, rar, gzip and bz2 feature vectors) but with a slightly higher rate of misclassified instances between enc and cmp. This can be explained by the fact that their distributions are very close, and intrinsically hard to distinguish. The worse-performing class is the video macro-label, for which a sizable portion of the samples is classified as encrypted content. However, given the results of the binary classification presented in Fig. 3, this behavior is expected. The

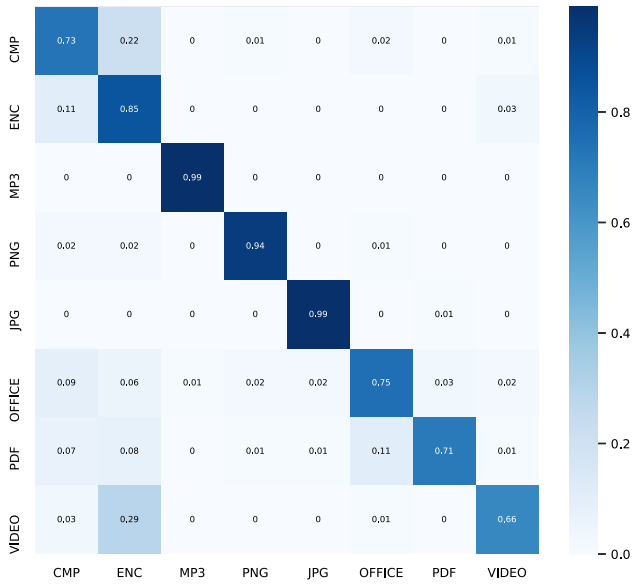


Fig. 6 Confusion matrix for the content-type classifier

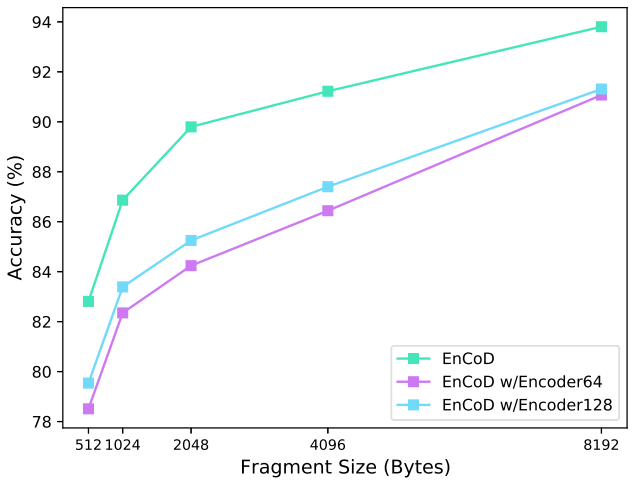


Fig. 8 Performance comparison for all formats binary classification between plain EnCoD, EnCoD with Encoder64 and EnCoD with Encoder128. Encoder64 uses 64 dimensions for the latent space representation and Encoder128 uses 128 dimensions

binary classifiers were already struggling in distinguishing the h264 and h265 types from encryption. Therefore, it is expected that the multiclass classifier would struggle as well with these formats.

5.5 Autoencoder approach

We analyze the effectiveness of the autoencoder approach presented in Sect. 4.3 compared to the base EnCoD binary approach. As previously discussed, we have two separate encoder architectures, encoder64 and encoder128, compressing the feature vector in a 64-dimensional and 128-dimensional latent representation, respectively. This compressed representation is then fed to a binary NN classifier to distinguish between encrypted data and compressed data of a specific type.

Figure 7 compares the performance of base EnCoD binary classifiers to the binary classifiers extended with encoder64 and with encoder128. As we can see, the base EnCoD models outperform the encoder models in all binary classification tasks, with the only exception being the pdf type where the encoders slightly outperform the base model. The difference in accuracy is especially apparent for the four general purpose compression formats bz2, zip, rar and gzip, where the best autoencoder (encoder128) results in accuracies 5 to 10 percentage points lower than the base model. Similar results can be observed in Fig. 8, which compares the average accuracy of all binary models

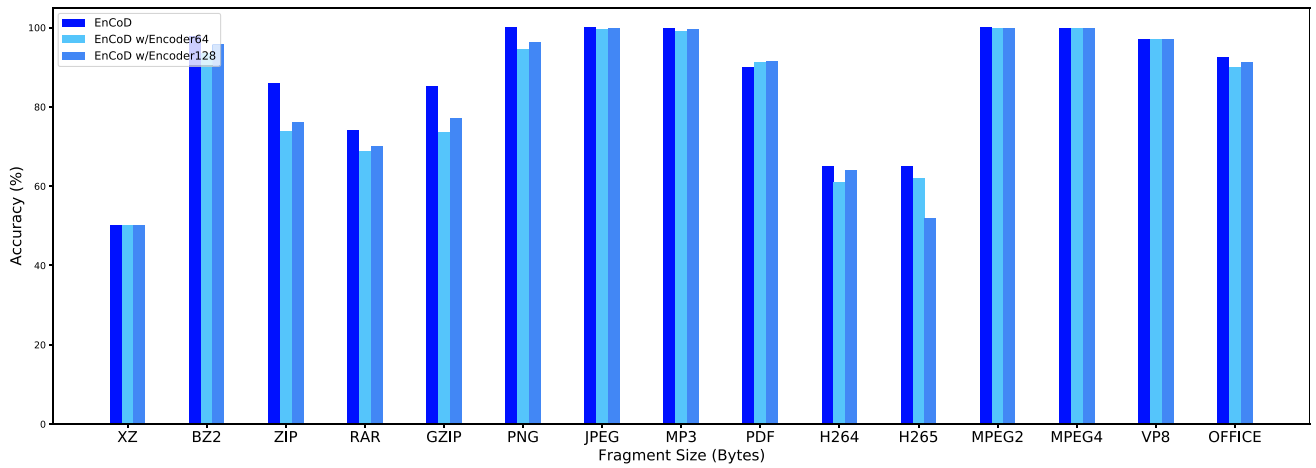


Fig. 7 Performance comparison between plain EnCoD binary classifiers, EnCoD binary classifier with Encoder64 and EnCoD binary classifier with Encoder128. Encoder64 uses 64 dimensions for the latent space representation and Encoder128 uses 128 dimensions. All results are for 2048 Bytes data fragments

Table 1 Time required by each approach to classify one sample, in seconds

Approach	Mean	Median	Std.dev
NIST	0.1	0.1	0.004
HEDGE	0.44	0.43	0.008
Binary classifier	0.00046	0.00044	0.00012

for different data fragment sizes. The performance of the encoder-based approaches trails that of the base EnCoD models for all fragment sizes, with accuracy differences in the range of 4 to 10 percentage points.

These results indicate that, while the encoder is able capture meaningful features for classification in the latent representation, it also loses critical information that the plain NN can use to achieve better performance.

5.6 Overhead

In the final part of our evaluation, we analyze the practical applicability of the three approaches, comparing their runtime in order to understand if they can be deployed in time-critical applications. For this test, we used a small dataset comprised of 1000 randomly-selected compressed or encrypted samples. We ran three approaches (NIST, HEDGE and our binary ML model) on each sample, taking individual runtime and repeating the experiment 1000 times. We did not include the autoencoder approach in this evaluation, given its poor performance when compared to the plain NN models. Table 1 presents the results of our evaluation. As we can see, while both mean and median runtime for NIST tests are faster then HEDGE, our proposed binary classifier is considerably faster than both. Both mean and median runtime for the ML model are three orders of magnitude faster than both NIST and HEDGE, making it easily applicable to scenarios that require fast classification results such as ransomware detection. It is worth noting that the evaluation of our ML model was carried out by measuring the time required to predict a single sample, rather than a batch of samples. However, our model can easily classify multiple samples in parallel by exploiting the heavy parallelism of GPUs, further decreasing the runtime required per individual sample.

6 Discussion of findings

Results shown in Sect. 5 highlight the difficulty of discriminating compressed and encrypted fragments. State-of-the-art statistical tests tend to fare better than entropy measures (ref. Sect. 2), but their performance varies significantly depending on the specifics of the compressed format and fragment size. Moreover, such approaches can only determine whether a given fragment is encrypted with a certain confidence, but cannot distinguish between different compressed formats. EnCoD, the learning-based approach introduced in Sect. 4, tackles both these limitations. Both per-format and multi-class classifiers outperform existing tests on all considered file types/block sizes. Moreover, our multi-class classifier can be used to determine the format of a given unknown fragment, even in the complete absence of any context or information on its type.

Results show that accuracy improves consistently with increasing fragment size. This is in a sense to be expected; all approaches considered in this paper leverage differences between the byte value distribution of random data (which is uniform) and that of compressed data. Perfectly estimating the byte value distribution of a short data stream is generally not possible. As sequences get shorter, the probability that the estimated distribution may not reflect the typical distribution for their content type increases. However, as the size of the sample increases, the estimated empirical distribution approaches the underlying data distribution, allowing us to capture any deviation from the uniform distribution. For modern compression algorithms, these deviations are quite minor, and a 512-byte block gives even accurate tests very little data to work with. However, when enough data are available, it is possible to identify the class of data with high accuracy; our learning-based classifier exceeds 90% accuracy already for 2048-byte blocks. In general, we recommend against using any one approach as the sole guidance for automated security decisions (e.g., dropping/allowing flows, terminating processes, etc.). However, when integrated as part of a more complete set of features in a larger system, our proposed classifiers can provide an additional robust feature to use in the decision-making process.

Given the discussion above, we suspect an intrinsic bound on the accuracy reachable by any classifier which looks purely at byte value distributions. However, approaches attempting to parse fragments or identify recognizable structures are likely to incur an impractical computational cost. Moreover, it is not apparent that any such structure is preserved for very short fragment sizes.

7 Related work

7.1 Entropy-based encryption detection

Use of entropy estimation to detect encrypted content is common in ransomware detection. Proposals such as RWGuard [2], UNVEIL [4], Redemption [1] and ShieldFS [3] use entropy of written content either directly as a feature, or as part of feature calculation. It should be noted that none of these detectors use entropy as the sole

feature for detection. However, evidence from Sect. 2 suggests that they may be better ignoring entropy altogether. In the realm of digital forensics, entropy estimation has been used to determine the type of unknown disk data fragments. One of the most complete approaches is that of Conti et al. [5]. However, the same authors found that such estimates have limited discerning power in distinguishing encrypted and compressed content, and aggregated the two types under a single label.

Entropy estimation has also been applied to the real-time analysis of network traffic. Dorfinger's Master thesis [7] proposes a system for discriminating encrypted and non-encrypted traffic, to ensure that all communications from a target network are encrypted. Similar approaches were also proposed by Mamun et al. [35] and Malhotra [10]. Zhang et al. proposed an entropy-based classifier for the identification of botnet traffic [36]. All these approaches also suffer from the limitations of using high entropy as a fingerprint of encryption. Wang et al. [11] report positive results in using an SVM classifier to discriminate between various data types using entropy estimates. Their application scenario is different from ours, as they consider both low-entropy (non-compressed) and high-entropy (compressed or encrypted) formats. We only consider high-entropy formats, which are difficult to distinguish using entropy alone. MovieStealer [37] aims at identifying encrypted and decrypted-but-compressed media buffers in order to break DRM. It uses an entropy test to single out encrypted and compressed buffers from other data, and the χ^2-test to distinguish them. It requires 800KB of data to reliably identify random data, which is far beyond the fragment size in the scenarios that we consider.

7.2 Non-entropy-based approaches

HEDGE, by Casino et al. [15], evaluates a combination of χ^2-test and a subset of NIST SP800-22 [27] to discriminate encrypted and compressed traffic. They use a dataset which is significantly smaller than ours, and do not discuss learning-based approaches. A limitation of this class of approaches is the fairly low accuracy, especially for small block sizes (ref. Sect. 5). Also, this and other similar approaches based on statistical randomness tests (e.g., [13, 16]) cannot distinguish between different types of compressed archives. Mbol et al. [12] investigate the use of the Kullback–Leibler divergence (relative entropy) to differentiate encrypted files from JPEG images. Their analysis does not investigate other formats and assumes the availability of blocks of significant size (128 to 512KB) from the beginning of each file. Especially in forensic and networking applications, uninterrupted blocks of such size are difficult to obtain.

While the application of neural networks to the problem at hand is fairly new, there exist some preliminary work. Ameeno et al. [38] show promising preliminary results; however, the analysis is limited in scope: it only attempts to distinguish zip archives from rc4-encrypted data and considers whole files (not fragments).

8 Conclusions

Discriminating encrypted from non-encrypted content is important for a variety of security applications, and oftentimes tackled via entropy estimation. We comprehensively highlighted the limits of this technique and reviewed the effectiveness of the leading alternative approaches: χ^2-test, NIST SP800-22 test suite, and HEDGE. In addition, we proposed EnCoD, a novel neural network classifier of our own design. In order to ensure generality of results, we created a dataset of 400M fragments covering five different sizes and 16 data formats.

Results show that previous state-of-the-art methods have blind spots which result in low accuracy for certain fragment sizes/data types. However, our neural network-based approach appears promising. While statistical tests only discriminate between compressed and encrypted data fragments, our multiclass classifier is able to classify between specific compressed formats with $\sim 83\%$ accuracy already on 2KB fragments, and our binary classifier reaches $\sim 90\%$ accuracy on the same fragment size. This suggests that systems incorporating encrypted content detection (e.g., ransomware detectors) would be better served by learning-based, rather than hand-crafted statistical approaches. This finding also suggests that learning may have useful applications to other problems in content-type inference. Overall, we believe this work is an important step forward toward reliable encryption detection.

Acknowledgements We would like to thank Daniele Venturi and Guinevere Gilman for their useful insights and comments. This work was partially supported by Gen4olive, a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 101000427, and by the Italian MISE Ministry, Project Proposal No. 1660, named "IDC - INTELLIGENCE DRIVEN CERT" - Agreement for Innovation under the DM August 2, 2019 - Sustainable Growth Fund - "Intelligent Factory" application area.

Funding Open access funding provided by Università degli Studi di Roma La Sapienza within the CRUI-CARE Agreement.

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

References

1. Kharraz A, Kirda E (2017) Redemption: real-time protection against ransomware at end-hosts. Research in Attacks, Intrusions, and Defenses
2. Mehnaz S, Mudgerikar A, Bertino E (2018) Rwguard: a real-time detection system against cryptographic ransomware. Research in Attacks, Intrusions, and Defenses
3. Continella A et al (2016) Shieldfs: a self-healing, ransomware-aware filesystem. In: annual computer security applications conference (ACSAC)
4. Kirda E (2017) Unveil: a large-scale, automated approach to detecting ransomware (keynote). In: IEEE international conference on software analysis, evolution and reengineering
5. Conti G et al (2010) Automated mapping of large binary objects using primitive fragment type classification. Digit Investig 7:S3–S12
6. De Carli L, Torres R, Modelo-Howard G, Tongaonkar A, Jha S (2017) Botnet protocol inference in the presence of encrypted traffic. In: IEEE international conference on computer communications
7. Dorfinger P, Panholzer G, John W (2011) Entropy estimation for real-time encrypted traffic identification
8. Fielding R et al. (1999) Rfc 2616, hypertext transfer protocol – http/1.1 .http://www.rfc.net/rfc2616.html
9. Park B et al (2008) Data extraction from damage compressed file for computer forensic purposes. Int J Hybrid Inf Technol 1(4):89–102
10. Malhotra P (2007) Detection of encrypted streams for egress monitoring. Master of Science, Iowa State University, Ames. https://lib.dr.iastate.edu/rtd/14632/
11. Wang Y, Zhang Z, Guo L, Li S (2011) Using entropy to classify traffic more deeply. In: 2011 IEEE 6th international conference on networking, architecture, and storage pp 45–52
12. Mbol F, Robert J-M, Sadighian A (2016) An efficient approach to detect torrentlocker ransomware in computer systems. Cryptol Netw Secur 532–541
13. Palisse A, Durand A, Le Bouder H, LeGuernic C, Lanet J-L (2017) Data aware defense (DaD): towards a generic and practical ransomware countermeasure. Secure IT Syst 10674
14. Hahn D, Apthorpe N, Feamster N (2018) Detecting compressed cleartext traffic from consumer internet of things devices 1805:02722
15. Casino F, Choo K-KR, Patsakis C (2019) HEDGE: efficient traffic classification of encrypted and compressed packets. IEEE Trans Inf Forensics Secur 14(11):2916–2926
16. Choudhury P, Kumar KRP, Nandi S, Athithan G (2020) An empirical approach towards characterization of encrypted and unencrypted VoIP traffic. Multimed Tools Appl 79(1–2):603–631
17. De Gaspari F, Hitaj D, Pagnotta G, De Carli L, Mancini L V (2020) The naked sun: malicious cooperation between benign-looking processes. In: 18th international conference on applied cryptography and network security
18. De Gaspari F, Hitaj D, Pagnotta G, De Carli L, Mancini L V (2022) Evading behavioral classifiers: a comprehensive analysis on evading ransomware detection techniques. Neural Comput Appl
19. Lee H, Ge R, Ma T, Risteski A, Arora S (2017) On the ability of neural nets to express distributions. In: Proceedings of the 30th conference on learning theory, COLT
20. Pagnotta G, Hitaj D, De Gaspari F, Mancini L V (2022) Passflow: guessing passwords with generative flows. In: Proceedings of the 52nd Annual IEEE/IFIP international conference on dependable systems and networks (DSN'22)
21. De Gaspari F, Hitaj D, Pagnotta G, De Carli L, Mancini L V (2020) Encod: distinguishing compressed and encrypted file fragments. In: international conference on network and system security pp 42–62
22. Atlanta spent \$2.6m to recover from a \$52,000 ransomware scare. https://www.wired.com/story/atlanta-spent-26m-recover-from-ransomware-scare/ (2018)
23. Wannacry cyber attack cost the nhs £92m as 19,000 appointments cancelled. https://www.telegraph.co.uk/technology/2018/10/11/wannacry-cyber-attack-cost-nhs-92m-19000-appointments-cancelled/ (2018)
24. Ransomware attacks grow, crippling cities and businesses. https://www.nytimes.com/2020/02/09/technology/ransomware-attacks.html (2020)
25. Walls RJ, Learned-Miller E, Levine BN (2011) Forensic triage for mobile phones with DEC0DE. In: USENIX Security Symposium
26. Wallace G K (1992) The jpeg still picture compression standard. IEEE Trans Consum Electron
27. Rukhin A et al. (2010) A statistical test suite for random and pseudorandom number generators for cryptographic applications. Special Publication 800-22r1a, NIST
28. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. Soc Artif Intell Stat (AISTATS)
29. LeCun Y, Bottou L, Orr G B, Müller K-R (1998) Efficient backprop. Neural Networks: Tricks of the Trade
30. Trottier L, Giguere P, Chaib-draa B (2017) Parametric exponential linear unit for deep convolutional neural networks. In: 2017 16th IEEE international conference on machine learning and applications (ICMLA)
31. Klambauer G, Unterthiner T, Mayr A, Hochreiter S (2017) Self-normalizing neural networks. CoRR **abs/1706.02515**
32. Piskozub M, De Gaspari F, Barr-Smith F, Mancini L, Martinovic I (2021) Malphase: fine-grained malware detection using network flow data. In: Proceedings of the 2021 ACM on asia conference on computer and communications security
33. Qi Y, Wang Y, Zheng X, Wu Z (2014) Robust feature learning by stacked autoencoder with maximum correntropy criterion. In: 2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)
34. Nguyen A, Yosinski J, Clune J (2015) Deep neural networks are easily fooled: high confidence predictions for unrecognizable images. In: 2015 IEEE conference on computer vision and pattern recognition (CVPR) pp 427–436
35. Mamun MS I, Ghorbani A A, Stakhanova N (2016) An entropy based encrypted traffic classifier. Inf Commun Secur 282–294
36. Zhang H, Papadopoulos C, Massey D (2013) Detecting encrypted botnet traffic. In: 2013 Proceedings IEEE INFOCOM pp 3453–1358

37. Wang R, Shoshitaishvili Y, Kruegel C, Vigna G (2013) Steal this movie - automatically bypassing drm protection in streaming media services. In: 22nd USENIX Security Symposium
38. Ameeno N, Sherry K, Gagneja K (2019) Using machine learning to detect the file compression or encryption. Amity J Comput Sci 3(1):6