Fingerprinting Cryptographic Protocols with Key Exchange using an Entropy Measure

Shoufu Luo*, Jeremy D. Seideman†, Sven Dietrich‡
*†‡ The Graduate Center, City University of New York, NY
‡John Jay College, City University of New York, NY
Email: *sluo2@gradcenter.cuny.edu,†jseideman@gradcenter.cuny.edu,‡spock@ieee.org

***Abstract*—Encryption has become increasingly prevalent in many applications and for various purposes, but its use also brings new challenges to network security. In this paper, we take the first steps towards addressing some of these challenges by introducing a novel system to identify key exchange protocols. These protocols are usually required if encryption keys are not shared in advance. We observed that key exchange protocols yield certain patterns of high-entropy data blocks, such as those found in key material. We propose a multi-resolution approach to accurately detect high-entropy data blocks and a method of generating fingerprints for cryptographic protocols. We provide experimental evidence that our approach has the potential to identify cryptographic protocols by their unique key exchanges, leading to the ability to detect malware traffic that includes customized key exchange protocols.**

I. Introduction

When designing and securing networks, the malicious use of encryption brings new challenges to network defense and traffic examination. For example, encryption has prevented botnet traffic from being inspected and detected by defense systems based on deep-packet inspection (DPI), which had previously proven to be very effective. Inspection and detection are used at different phases of network analysis, such as as a preventative measure and as a "post-mortem" analysis. At both phases, it is important to be able to determine what the network traffic stream is, whether it is benign or malicious; being able to identify key segments of communication, even when encrypted, is useful for this task. When parties employ symmetric encryption and decryption, a secret key k is shared among them, either pre-shared or negotiated on the fly using cryptographic key-exchange protocols. Most common cryptographic protocols [1]–[3] using symmetric encryption to secure the channel use a key exchange protocol, such as the Diffie-Hellman key exchange [4].

There are different ways to distribute key material along the traffic stream, and the method selected depends on the protocol design. As key material has high entropy compared to normal traffic, the traffic for the key exchange exhibits detectable characteristics, namely the uniqueness of the distribution of key material. This allows for the identification of discriminating characteristics, as shown in Figure 1. Using an entropy metric, we can test the hypothesis of whether a byte string is "random," if that byte string is sufficiently long. However, this problem becomes more difficult if the given string is relatively short (undersampled), or if the goal is to identify which part of the string contains random bytes. This is especially true when determining the boundaries of those groups of random bytes (also known as *blocks of interest*). It is therefore challenging to characterize a stream by the distribution of sequences of embedded random bytes, or so-called "high-entropy blocks."

Fig. 1: Visualization of Entropy Distribution: dark portions are high-entropy blocks.

To avoid generating traffic that would be detected and treated as an anomaly, malware might try to use standard cryptographic protocols for secure communication, effectively preventing DPI. However, standard protocols such as SSL/TLS can potentially be subject to a man-in-the-middle attack. Malware, in general, tends to avoid using standard protocols and instead employs customized variants in order to avoid detection or attack. According to a recent study, only 10% of malware utilizes SSL/TLS for traffic encryption [5]. To ensure fresh key material, a new key exchange is desirable for every new command-and-control (C&C) session of the malware [6], [7].

Our work offers a systematic way to characterize network traffic through key exchange behaviors and generate scalable fingerprints based on detected high-entropy blocks. The system mainly consists of two parts: detection of high-entropy blocks and the fingerprint generation. First, we identify those blocks from a traffic stream using sample entropy measured over a sliding window. Second, with all high-entropy blocks identified, we generate fingerprints for network flows using the distribution of those blocks. The use of the fingerprints makes it easier to identify the network flows, enabling us to classify them as malicious or benign. Our contribution also includes:

- A new method of identifying cryptographic protocols, which also raises the bar for malicious actors that abuse customizing cryptographic protocols to evade inspection.
- A voting mechanism that efficiently boosts the accuracy of entropy-based classification when undersampled entropy is calculated, using a multi-resolution analysis.
- A statistical approach to estimate the range of high-entropy data blocks and build scalable entropy-based fingerprints for key exchange protocols in the form of

regular expressions.

To the best of our knowledge, our work is the first attempt to fingerprint key exchange protocols by the suspected distribution of key material and apply this technique to malware detection by traffic analysis. By design, our approach can be implemented and deployed as a standalone system. However, it is not the intention to replace any existing detection techniques, but rather to augment them either in real-time or in a forensic setting for threat resolution and mitigation. This system can be built into existing systems as a plug-in component, such as those relying on a certain degree of payload analysis, e.g. [8]. Moreover, a component of our system can be a useful tool for the security community, for identifying high-entropy portions of a given data block, for use such as the detection of packed malware binaries.

A. Related Work

Olivain et al. [9] proposed the use of cumulative entropy of network flows for the detection of specific attacking behaviors targeted at known cryptographic protocols such as SSL. Instead of an aggregation though, our work aims to fingerprint the entropy distribution along the examined traffic. Our approach is still applicable for their purpose in a more precise way. For this reason, we adopt the technique they propose, *N-truncated entropy*, for entropy estimation, which is also used by Dorfinger et al. [10] for classifying encrypted and unencrypted traffic. There is prior work that shows how entropy tests can be used to detect encrypted or compressed packets from network streams [11]. Their work examined the use of entropy to identify data that is "opaque," which could be anything from multimedia files to encrypted traffic sent over normally unencrypted channels. Their work indicated a need for larger numbers of sample packets for their calculation. Our work provides a more reliable mechanism to detect high-entropy areas, this being one of our essential contributions.

There is also precedent for the use of entropy analysis of traffic compression for IDS. The authors of [12] introduce a method by which they establish the entropy of traffic generated by various communication protocols. As the authors felt that compressed data makes IDS tasks difficult, they sought to use a wide variety of entropy measurements (but not different estimators) to create a filtering method. Their method, while identifying different communication protocols, does not identify methods of cryptography. Our method could be augmented with theirs, however, if we wanted to identify communication *and* cryptographic methods.

Our work is also motivated by the field of protocol identification. Much of the work in that field is learning-based for the most part, relying on observable features [13], [14]. For example, Wright et al. [14] proposed to identify the cryptographic protocol of individual encrypted TCP connections using post-encryption observable features, including timing, size, and direction. To some extent, our approach can also be also leveraged for this purpose. However, there are known obfuscation techniques which could be used to evade this technique, such as obfsproxy [15] and FTE [16]. As discussed in [17], obfuscation *can* be detected with entropy-based tests over the packet payloads. Our approach does this by extracting entropy-based fingerprints.

Zhang et al. [18] proposed to detect encrypted traffic by looking for *N* sequential high-entropy packets of the first *M* packets of one network flow, incorporating the cumulative entropy technique. They then built upon this work [19] by detecting high-entropy flows as an additional measure to score a host to identify it as a bot for BotHunter [20]. While applicable to the same problem, our approach is different from theirs by fingerprinting malware with customized cryptographic protocols, such as Nugache. Our work is different from theirs as our work does not rely on another system for detection.

Richer [21] proposed a similar approach to Zhang et al. by using a classifier based on the regularity in the data over time and size, employing an entropy-based estimator to detect botnet activity. They were able to properly train their classifier, but their work was designed to detect botnet "beacons" based on traffic patterns. Our work instead tries to identify the traffic itself and determine how it is being encrypted.

The remainder of this paper is organized as follows. We begin with background on entropy and its estimators in Section II. In Section III, we discuss our methodology in detail, including how to identify high-entropy blocks, the voting mechanism, and a filtering method for false positive reduction. Following that, Section IV presents evaluation and analysis of our approach with three different datasets. Finally, we conclude by discussing the limitations of this work and directions of future work.

II. Background

A. Entropy

Introduced by Shannon [22], entropy is used as a measurement of the amount of information that is produced by a process. It can also be used, therefore, to determine the information that is missing before reception if that information is transmitted. In the context of cryptography, though, it is used as a measure of randomness (or uncertainty), equating higher entropy with higher randomness. Let X be a discrete random variable under an arbitrary distribution $\mathcal{P}$ on a countable alphabet $\Sigma = \{x_1, ..., x_m\}$. The definition of Shannon entropy can be generally expressed by Equation 1:

$$H(X) = -\sum_{i=1}^{m} p(x_i) \log_2 p(x_i) \tag{1}$$

The entropy $H(X)$ yields a maximum value when all $p(x_i)$ are equal to $\frac{1}{m}$, i.e. uniformly distributed. In cryptography, as a fundamental requirement of security, key material should have high entropy in order to be hard to predict.

B. Entropy Estimator

Entropy can be easily obtained by the Equation 1 if given a random variable whose probability distribution is known. However, in practice, $\mathcal{P}$ is unknown for most scenarios.

Frequently, $p(x_i)$ could be still estimated by the relative frequencies of the outcome x_i from a large number of trials. The probability of x_i is thereby $\hat{p}(x_i) = \frac{n_i}{N}$, where n_i is the number of times x_i occurs and N is the total number of trials or samples. Using this probability measurement, the *sample entropy*, or Maximum Likelihood Estimator (MLE) [23], can be estimated by Equation 2:

$$\hat{H}_N^{MLE}(X) \equiv -\sum_{i=1}^{m} \hat{p}(x_i) \log_2 \hat{p}(x_i) \tag{2}$$

MLE is an unbiased estimator of $H(X)$ when N tends to infinity and where $\hat{p}(x_i)$ approximates $p(x_i)$, $\hat{H}_N^{MLE}(X)$ approximates actual $H(X)$. When N is not sufficiently large, though (*undersampled*), $\hat{H}_N^{MLE}(X)$ is highly biased, in particular when $N < m$ or $N \sim m$. There is no universal rate at which the error of MLE compared to $H(X)$ would be close to zero [23].

There are attempts which aim to reduce the bias directly, such as the Miller-Madow corrector [24], the Jackknife corrector [25] and the Paninski corrector [26]. However, the bias is still significantly high when $N < m$ or $N \sim m$. Moreover, it has been difficult to find an unbiased estimator [26], [27]. It is worth noting the Paninski corrector is unbiased if and only if $\mathcal{P}$ has a uniform distribution, which can not be guaranteed.

MLE is by no means the only entropy estimation that could be employed. The authors of [28] examined several other methods. These methods include various statistical methods (including frequencies and substrings), Markov-based estimates, and the use of compression. The use of other estimators and how their results compare to those of MLE would make an interesting exercise, which can be explored in the future.

According to [9], $\hat{H}_N^{MLE}(X) \sim H(X)$ is valid if and only if $N \gg m$, which typically means N is of the order of roughly at least *10* times as large as m. In other words, if $\Sigma_0 = \{0\text{x}00, ..., 0\text{xff}\}$ (i.e. m=$|\Sigma_0|$=256), it would require around 2,000 samples to possibly obtain a reasonable estimated entropy. That makes it impractical for the purpose of profiling network traffic as key material is usually, at most, hundreds of bytes. For example, in a typical TLS handshake, a client random number only contains 28 bytes.

C. N-truncated entropy $H_N(X)$

Similar to Olivain et al. [9], an accurate entropy value is not our main focus, but rather the probability of a string being generated from a uniform distribution. The *N-truncated entropy* $H_N(X)$ they proposed meets our needs, which is the average of the sample entropy $\hat{H}_N^{MLE}(X)$ over all strings of length of N drawn at random from the distribution $\mathcal{P}$, as defined in Equation 3.

$$\sum_{\Sigma_i n_i = N} \left[\binom{N}{n_0, ..., n_{m-1}} \prod_{i=0}^{m-1} p_i^{n_i} \left(-\sum_{i=0}^{m-1} \frac{n_i}{N} \log_2 \frac{n_i}{N} \right) \right] \tag{3}$$

By construction, $\hat{H}_N^{MLE}(X)$ is an unbiased estimator of $H_N(X)$ for an arbitrary distribution $\mathcal{P}$. More importantly, $\hat{H}_N^{MLE}(X)$ gives a statistical indication of how close the distribution $\mathcal{P}$ is to being uniform by comparing to $\hat{H}_N^{MLE}(W)$, assuming that W is a random variable under a uniform distribution $\mathcal{U}$. In Section III-B, we describe how to obtain both values. Alternatively, if a string s of length N with each sample is drawn from $\mathcal{P}$, we use $\hat{H}_N^{MLE}(s)$ instead of $\hat{H}_N^{MLE}(X)$. To differentiate this, w is instead used for a uniform distribution, $\mathcal{U}$. $H_N(X)$ has an upper bound of $log_2 \min\{m, N\}$ as it reaches its maximum value if all $\hat{p}_{x_i}$ are equal, either $\hat{p}_{x_i} = \frac{1}{N}$ if $N < m$ or $\hat{p}_{x_i} = \frac{1}{m}$ otherwise. In either case, uncertainty reaches its maximum.

III. Methodology

In this section, we discuss the techniques we used and developed, and present experimental evidence demonstrating their effectiveness.

A. Sliding Window

To obtain the information entropy of different portions within the traffic stream, we utilize a sliding window that moves over the traffic, with a step of one byte at a time, while sample entropy is calculated for each segment of bytes in that window.

In order to local all high-entropy blocks in a string, we move a *N-character sliding window* over the string, with a 1-character step. We evaluate $\hat{H}_N^{MLE}(s)$, an unbiased estimator of entropy, where s is the string that lies completely within the window. We refer to the bytes in each window as a **block**. It follows that $\|s\| = N$, which is the size of the window, and subsequently the block size.

The window size determines the sample size, which directly impacts the accuracy of sample entropy. If the sample size is too small, the sample entropy might not be accurate enough to be meaningful. Equation (4) roughly estimates $Pr[X = e]$, the probability of a N-byte string appearing to be "random", i.e. each character in the alphabet only occurring once in the string. Fix Σ to be Σ_0 and then let m=256. With N=16 defining a 16-byte sliding window, $Pr[X = e] = 0.6197$. In other words, there is a 40% probability that an arbitrary string will appear to be random and will be a false positive. However, under the same assumptions but with $N = 32$, $Pr[X = e] = 0.082$. This confirms the discussion in Paninski, et al. [26] that one should never use less than 16 bytes for entropy estimation when Σ_0 is used.

$$Pr[X = e] = 1 \cdot \frac{m-1}{m} \cdot ... \cdot \frac{m-N+1}{m} = \prod_{i=0}^{N-1} \frac{m-i}{m} \tag{4}$$

Similarly, if the sliding window size is too large, blocks are likely to mix high-entropy areas with low-entropy areas, confusing the difference between them. As shown in Figure 2, when the window size is small, e.g. 16-byte, the curve is fuzzy and has too many valleys (low-entropy) and peaks (high-entropy), while as the window size goes larger, e.g. 1024 or 2048-byte, the curve becomes flatter; valleys or peaks are no longer distinctive.

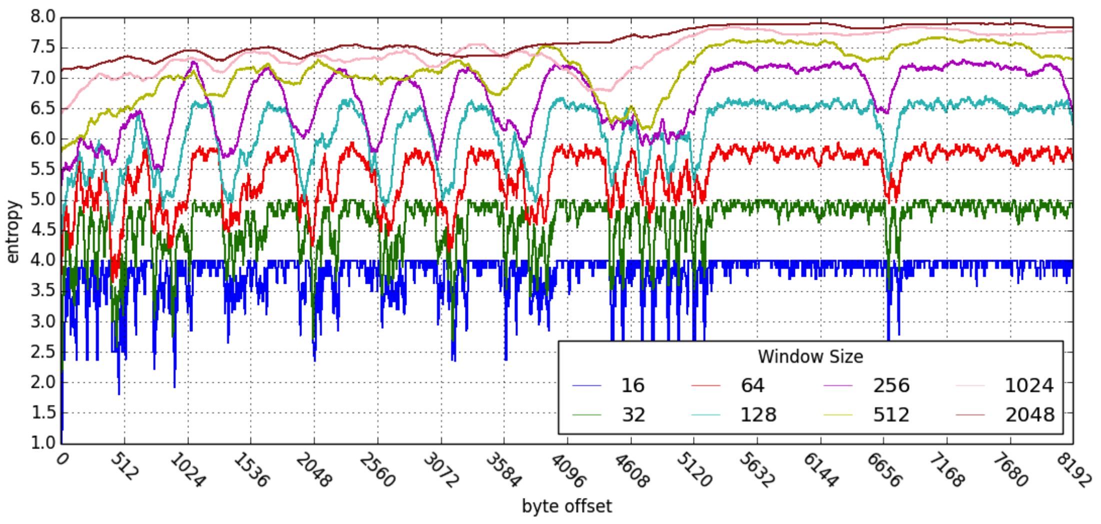

Fig. 2: Entropy plot of a TLS sample traffic using different sliding window sizes, from bottom to top (-byte): 16, 32, 64, 128, 256, 512, 1024 to 2048.

A smaller window is more likely to mistakenly identify a non-random data area to be "random" (false positive), while a larger window possibly fails to identify real high-entropy area (false negative). The choice of window size will heavily depend on the minimum length of key materials of interest. In the case of TLS, we choose a 32-byte sliding window as it is good for the minimum length of interests, a 28-byte client random number. In summary, as the window slides over the data with a one-byte step, each block is labeled as either high-entropy or low-entropy. A list of consecutive high-entropy blocks or low-entropy blocks then forms a **unit**, more precisely a high-entropy unit or a low-entropy unit respectively.

B. Baseline $H_N(\mathcal{U})$

To identify a high-entropy block, we use the method employed by [9], the Monte-Carlo method, as it provides a level of confidence of a string being drawn from a random distribution. We first repeatedly generate strings of length of N with each byte sampled from a random source, such as */dev/urandom* on Unix. Then, we calculate the mean μ and standard deviation σ of sample entropy using all samples. Here, μ and σ summarize the distribution of the sample entropy of random strings of length N. By a specific number t of standard deviations, we can obtain the proportion of sample strings falling within the range of $\mu \pm t \times \sigma$. This proportion provides us with an indication of confidence that a string is random if it falls within the given range. As exceeding the upper bound does not affect the randomness of the string, we ignore the upper bound and use the lower bound as a cutoff for a string being random, denoted by θ:

$$\theta = \mu\big(\hat{H}_N^{MLE}(w)\big) - t \times \sigma\big(\hat{H}_N^{MLE}(w)\big) \tag{5}$$

With a confidence factor expressed by the proportion ρ:

$$\rho = \frac{number\ of\ samples\ above\ \theta}{number\ of\ samples} \tag{6}$$

Consequently, any strings falling below the threshold are considered not to be random, i.e. low-entropy blocks. Similarly, any strings falling above the threshold will be considered random, i.e. high-entropy. Table I shows thresholds (θ) for w using different window sizes (N) above a minimum level of confidence 99.0%.

N	μ	σ	t	θ	ρ
16	3.94199	0.08290	2.8	3.7098	99.2%
32	4.88171	0.08134	2.7	4.6620	99.3%
64	5.76562	0.07664	2.6	5.5663	99.2%
128	6.55003	0.06733	2.5	6.3817	99.2%
256	7.17518	0.05240	2.5	7.0441	99.2%
512	7.59073	0.03364	2.4	7.5099	99.0%
1024	7.80894	0.01726	2.5	7.7658	99.1%
2048	7.90804	0.00814	2.5	7.8877	99.2%

TABLE I: $\hat{H}_N^{MLE}(w)$ under Various Configurations

The confidence measures the confidence of a string not being random when falling *out* of the range, rather than a confidence of a string being random when falling *within* the range. For example, let N be 64 and Σ=Σ_0, then μ=5.7656 and σ=0.0766. With 99.4% of samples above θ=μ-3σ (i.e. $t = 3$), we would have at *least* 99.4% confidence that a string s, with $\hat{H}_N^{MLE}(s)$=5.5120, is not close to random, and therefore not a high-entropy block. In this calculation, t is our control variable. We can choose a smaller t to tighten the range and give us a higher confidence, or a larger t to loosen the range, at the expense of a lower confidence. In our study, we chose a smaller t in order to obtain a relatively high confidence, at least 99.0%.

Using the threshold value, we could then transform the sample entropy score to either 0 or 1. The plot then becomes a square wave where *one* indicates high-entropy and *zero* for low-entropy as shown in Figure 3. The shadow in the upper plot shows the cutoff (threshold) value.

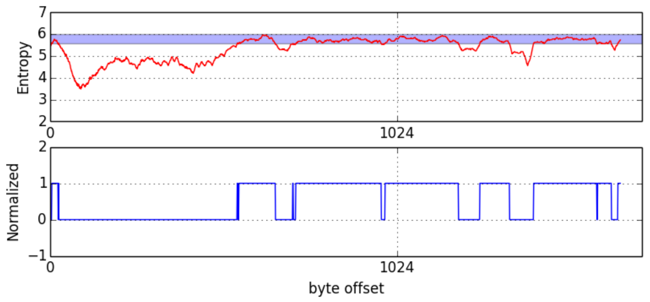

Fig. 3: Normalized plot of high-entropy blocks

C. The Choice of Σ

Due to statistical limitations, some data blocks may mistakenly be labeled as high-entropy blocks, i.e. a false positive, which will lead to an error in the fingerprint; therefore we must be avoided or minimize them. To achieve this, we devised a voting mechanism using multi-resolution analysis, utilizing the choice of alphabet Σ. This mechanism drastically reduces the rate of false positives.

So far, we have based our discussion on the choice of Σ to be Σ_0 (m=256) with each character being a byte. In cryptography, however, the randomness of key material is defined at a more restrictive level, i.e. at a *bit* level, so Σ=$\{0, 1\}$ (m=2). Let's consider one experiment of tossing one coin that has two outcomes, and another experiment of tossing eight independent coins with two outcomes for each. According to basic probability theory, if each coin is uniformly drawn from Σ=$\{0, 1\}$, the outcome of eight coins (Σ_0) will still follow a uniform distribution. In our estimation of $\hat{H}_N^{MLE}(w)$, we *do* generate each random byte by randomly sampling eight times over $\{0, 1\}$ for all of our sample strings. That being said, given that each bit is independently sampled uniformly from $\{0, 1\}$, we could define τ, a number of bits (i.e. coins) used to generate our random variables, and from those, choose a random variable; such a random variable will be guaranteed to have a uniform distribution.

As an extension to the previously defined $\hat{H}_N^{MLE}(w)$, we outline the thresholds and their confidence levels for different values of τ while fixing $N = 32$. We use the term **τ-bit measure** (e.g. 2-bit measure). Previously, N could be interpreted as either the window size and the sample size. In the case of τ-bit measure, the sample size changes, i.e. $\frac{8}{\tau}$ N ($\tau \leq 8$). For convenience, we abuse the notation N, using it as the window size for the remainder of this paper. It is worth noting the use of τ-bit measure does not change the fundamentals of *N-truncated entropy* as it simply uses a larger sample size and a different alphabet.

Statistical methods such as sample entropy generally ignore potential structures or patterns occurring in the data. Therefore, a string with a high sample entropy score is not always guaranteed to be random, such as if the string is made up of repeating sequences of bytes. For example, given a hexadecimal string s be "55 55 bb bb" (0101 0101 0101 0101 1010 1010 1010 1010 in binary), we see that $\hat{p}_0 = \hat{p}_1 = \frac{1}{2}$ if a 1-bit measure (τ=1) used, i.e. Σ=$\{0, 1\}$; subsequently $\hat{H}_N^{MLE}(s)$ = 1. Consequently, s will be labeled as a high-entropy block despite the fact that it is not, resulting in misclassification. Taking another example from real world, this hexadecimal string from a TLS session: 16 03 01 0c 13 0b 00 0c 0f 00 0d 0e 10 04 7a 30 82, which is a block of control information[1] from the TLS handshake traffic. The two bytes *03 01* indicate the TLS version, i.e. TLS 1.0, *0c 13* for the length,*0b* for the protocol type, and another 3 bytes of length *00 0c 0f*. This block also may not appear "random" if an 8-bit measure is used. Such cases are prone to false positives and impact the fingerprint process.

τ	m	μ	σ	t	θ	ρ
1	2	0.9971	0.00399	4.18	0.9804	99.28%
2	4	1.9829	0.01387	3.59	1.9331	99.20%
4	16	3.8196	0.06715	3.02	3.6168	99.31%
8	256	4.8817	0.08135	3.0	4.6356	99.35%

TABLE II: τ-bit measure $\hat{H}_{32}^{MLE}(w)$

However, the idea is that if a string is random, no matter which τ-bit measure is being used, its sample entropy $\hat{H}_N^{MLE}(s)$ should be always close to $\hat{H}_N(\mathcal{U})$. Thus, we propose to use a voting mechanism instead of only using a τ-bit measure. The voting rule we employed is that if *any* of the chosen τ-bit measures reject the randomness of that block, the block will be labeled as non-random. It is executed as a simple AND operation among the outcome of all measures. Figure 4 shows the effectiveness of combining three τ-measures, where the resulting signature by voting precisely outlines all high-entropy blocks in the TLS session. The bottom plot line, X-signature, is based on the voting over the three 1-bit, 4-bit and 8-bit measures.

D. Filtering Threshold

Our voting mechanism effectively reduces false positives. However there are some scenarios where this approach may still not be sufficient to eliminate all false positives. There is still a chance that while voting, all selected τ-bit measures may falsely identify an ordinary block to be high-entropy because some randomness within the data makes it appear to be a true high-entropy block. If there are supposedly no high-entropy data blocks then the length of a data block with random data should be less than the minimum length of interest; the size of detected high-entropy units would appear to be relatively small compared to what is expected if there actually exists a high-entropy data block of interest. We define a filtering threshold, ξ, chosen to eliminate those small high-entropy units. Our empirical study suggests $\xi = 9$ as a good choice when a 32-byte sliding window size is chosen for detecting a minimum 20-byte high-entropy key material blocks. This means if there are only 9 consecutive high-entropy blocks detected between two low-entropy units, then we identify the result as a false positive and filter it out. Here, "filter out" means labeling these

[1]Control information is commonly known to have low entropy.

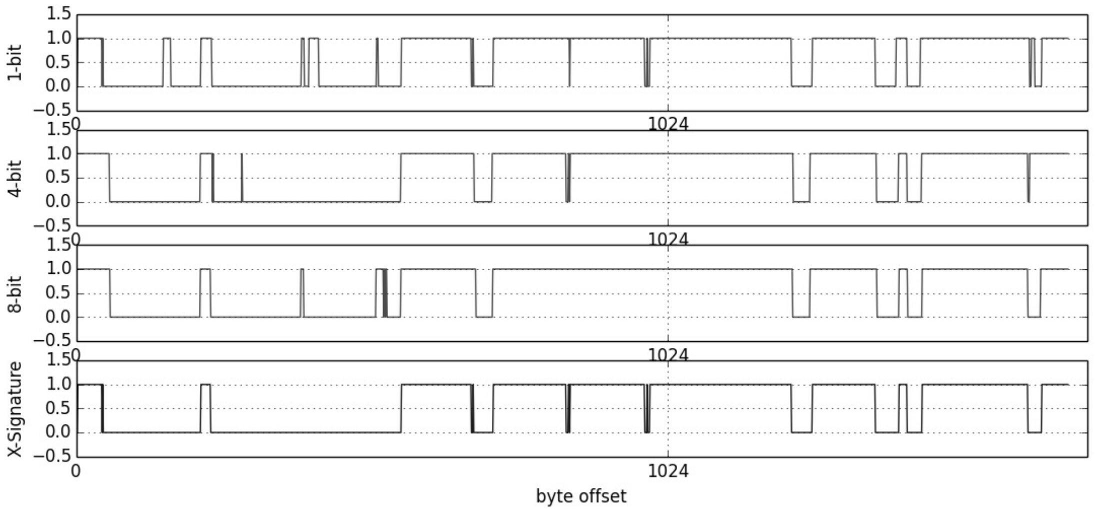

Fig. 4: A traffic sample from a TLS 1.2 session with a 1024-bit RSA public key.

blocks to be low-entropy instead of high-entropy, effectively removing them from further examination.

E. Calibration

In order to fingerprint traffic by the shape of the square wave as showin in Figure 4, we must go beyond the identification of high-entropy blocks. We must also descirbe the length of each unit within the traffic flow. Due to statistical inheritance and our calculation method, the length of each unit (i.e. the number of detected consecutive high-entropy or low-entropy blocks) may vary because when the sliding window partially includes the target random bytes, it may still continue to yield high sample entropy blocks until the window moves sufficiently away from the target. For example, a TLS traffic stream contains a client random number as a sequence of 28 bytes. It is not illogical to anticipate that there will not be exactly one high-entropy block detected in this case. Additionally, the total number of high-entropy blocks detected around that chunk of data will not be fixed from case to case. However, our intention is not to determine an absolute value for each unit among all cases, but rather a certain reasonable range. For this reason, we resort to *Monte-Carlo* methods to empirically estimate the range. For example, to estimate the length of high-entropy unit around client random bytes, we sampled 100,000 *client hello* messages from TLS sessions.

The result shown in Figure 5 indicate that most of the length of the 28-byte client random string followed by the list of cipher suites fall within a range of six to twenty-four high-entropy blocks. If a 32-byte TLS session ID (also random bytes) is present along with the client random bytes, adding up to 60 bytes, we obtain a range of $[38, 52]$ as shown in Figure 5. A more conservative range would be $[20, 52]$.

F. Fingerprinting

Fingerprinting is a process to profile a key exchange protocol by its distribution of high-entropy blocks along the traffic streams the protocol generates during use. A entropy-based fingerprint is a series of interleaving high-entropy units and low-entropy units with the length of each unit specified as a range. The reason that high-entropy blocks have to interleave with low-entropy ones is that otherwise two adjacent high-entropy or low-entropy blocks would be merged into one. Let (s, l, r) represent one unit where $s \in \{1, 0\}, l, r \in \mathbf{Z}^+$, where s be the sign indicating a high-entropy unit or low-entropy, l be the minimum length and r be the maximum length. An entropy-based fingerprint then is the concatenation of an ordered list of n instances of (s, l, r) with s alternating among one and zero. Alternatively, it can be concisely expressed as below, where $s_i \in \{1, 0\}, l_i, r_i \in \mathbf{Z}^+$. The benefit of such a representation is that this form can be represented as a standard regular expression and therefore the matching process can be done efficiently. The regular expression form will provide a flexible way of expressing the fingerprint, for instance, optional units, as will be shown in the experiment section.

$$\Big|\Big|_{i=1}^{n} s_i\{l_i, r_i\}, s_i \neq s_{i+1}$$

The fingerprinting is straightforward in three steps:

1) Identify high-entropy and low-entropy areas (units) of the anticipated traffic from a cryptographic protocol;
2) Follow the technique described in Section III-E and estimate the range for each area;
3) Formalize the units in a regular expression.

Taking TLS using a cipher-suite of DHE-RSA-* as an example, the fingerprint obtained through our method is as below:

$$1\{8, 54\}0\{20, 1024\}1\{8, 54\}0\{30, 800\}1\{80, 260\}....$$

We use the following steps to detect and classify traffic flows: (1) scan the traffic stream by sliding a window over it and estimating sample entropy for each window using different τ-bit measures; (2) normalize each block by its entropy score

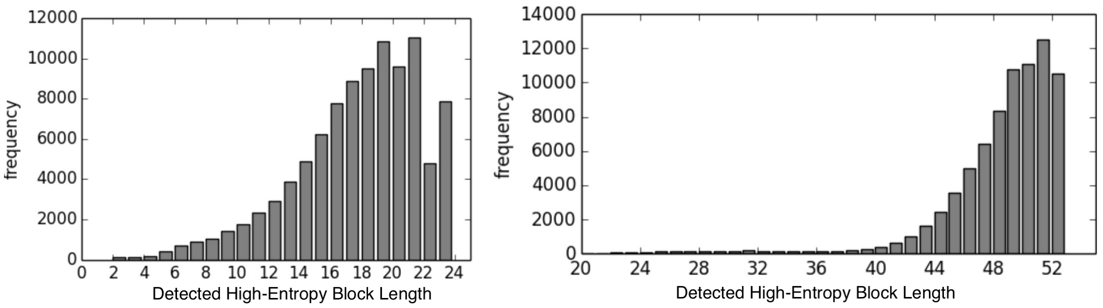

Fig. 5: Distribution of length of detected high-entropy blocks (1) Left: over the TLS 28-byte client random string (2) Right: over the TLS 28-byte client random string and 32-byte session ID.

to either one or zero using the pre-calculated threshold θ; (3) perform the voting of outcomes from each measures; (4) filter out noise using ξ; (5) use a regular expression to match the predefined fingerprint against the output (i.e. a string consisting of zeros and ones).

In our demonstration, we emphasize the DHE-RSA-* cipher-suite for the TLS protocol, as our approach aims to profile a particular key exchange protocol, and TLS is capable of using different key exchange protocols. SSL has evolved over time into the standard TLS protocol, which supports a long list of cipher suites with different key exchange protocols. To demonstrate, we choose to profile one set of key exchange protocol cipher suites, i.e. DHE-RSA-*, (see Section III). It is worth noting that most botnet C&C protocols are much simpler as they are generally desinged to perform a limited number of tasks.

IV. Evaluation

SSL/TLS is a well-known cryptographic protocol with fair complexity. The successful characterization of the TLS protocol provides the full ability to characterize other, simpler botnet C&C protocols (i.e. characterization of a more complex protocol indicates that a simpler protocol can also be characterized by our method). For our evaluation, we first use TLS as our primary target; we then extend it to the Nugache botnet. All streams are bidirectional and the packets of a stream are correctly ordered with all TCP/IP headers removed. *Tshark* [29] was used as a primary tool to process network traces in `pcap` [30].

A. Datasets

We obtained a data set of TLS network traffic from the ZMap project [31]. Initially, we extracted 16,240 TCP streams on standard port 443 from 800MB of raw traffic data, which we further reduced to 5,794 completed and validated TLS streams[2]. Then, we extracted from those 5,794 streams the 1,378 streams that used one of the DHE-RSA-$*$ cipher suites listed in Table III. We split 1,378 instances into two sets: the *d00200* set of 218 instances for parameter selection and signature refinement and the test set *d00300* of 1,160 instances for the testing of the final signature, denoted as the *d00015* set. We also extracted 1,204 TLS instances using other cipher suites. We extracted 337 Nugache traffic streams from a set of raw Nugache traffic and divided instances into two groups: 162 instances of training set and 175 instances of testing set. Similar to TLS, we use the training set to tune the fingerprint and the testing set for validation.

We examined the possibility of using Ethernet frames instead of the TCP packet flow. However, we ultimately decided against that because of several issues, most notably fragmentation. If a high-entropy block defined in the TCP flow is fragmented in an Ethernet frame, then instead of a single block there might be smaller high-entropy blocks. That would change the fingerprint of the cryptographic protocol employed. As we are generally looking at activity at the application level [6], we decided to focus on the analysis of what is assumed to be the application payloads.

Cipher ID	**Name**
0x00015	`TLS_DHE_RSA_WITH_DES_CBC_SHA`
0x00016	`TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA`
0x00033	`TLS_DHE_RSA_WITH_AES_128_CBC_SHA`
0x00039	`TLS_DHE_RSA_WITH_AES_256_CBC_SHA`
0x00045	`TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA`
0x00067	`TLS_DHE_RSA_WITH_AES_128_CBC_SHA256`
0x0006B	`TLS_DHE_RSA_WITH_AES_256_CBC_SHA256`
0x00088	`TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA`
0x0009A	`TLS_DHE_RSA_WITH_SEED_CBC_SHA`
0x0009E	`TLS_DHE_RSA_WITH_AES_128_GCM_SHA256`
0x0009F	`TLS_DHE_RSA_WITH_AES_256_GCM_SHA384`

TABLE III: TLS Cipher suites of Choice: DHE-RSA-*

In addition, we used 3,412 non-TLS TCP streams from a data

Port	80	25	22	143	21	111	179	110
#	582	189	168	125	96	44	18	5

TABLE IV: The number of steams from different traffic types identified by port

set generated by UNSW-NB15 [32]. This data set contains a variety of traffic types, but without any TLS traffic, meaning we could use it as another dimension of negative cases for

[2]A large portion of hosts scanned by the ZMap client did not respond or reject connections for various reasons during TLS negotiation

testing the fingerprints. Table IV shows the traffic type of the majority by service ports, only including standard ports under 1024. The table does not show the whole spectrum of traffic types in this dataset, but rather provides a quick look.

B. TLS

We tested the signature generated as previously described over the training set *d00200* with thresholds of the confidence ρ above 99.2% for different τ-bit measures. The results shown in Table V do not seem promising at all, as all the best results from 1-4-8 and 1-2-4-8 only reach a recall rate of 62.84% and 64.22% respectively, with the confidence of 99.85%, but it does confirm that the strategy of using multiple τ-bit measures significantly improves the recall rate. Also, it is interesting to note that the recall rate of multiple τ-bit measures drops significantly below 10% with a confidence of 99.99%. This is expected because the threshold is too relaxed (with a higher proportion of high entropy blocks) to be accurate.

τ \ ρ	99.20%	99.85%	99.97%	99.99%
1-bit	8.72%	36.70%	26.15%	26.14%
2-bit	15.13%	10.55%	23.39%	23.39%
4-bit	47.25%	25.68%	8.26%	11.93%
8-bit	42.40%	28.44%	3.21%	10.09%
1-2-8	31.19%	17.43%	7.80%	4.58%
1-4-8	45.41%	**62.84%**	38.99%	5.50%
1-2-4-8	39.44%	**64.22%**	39.44%	5.05%

TABLE V: Recall rate of the original signature for TLS

By manually reviewing those failures, we found three major issues in the generation of our original signature. One is the range of the server random bytes. It was a tighter range than it anticipated, which was previously set to (+, 8, 54) since we used the range estimated from client random bytes. We found that to be inadequate as the bytes *after* the server random bytes appear more random than those after client random bytes, and are therefore more likely produce a longer high-entropy block. As we did for client random bytes, we increased the maximum length to 64 to rectify this. The second major issue is that we failed to consider optional random bytes such as key identifier fields for both issuer and subject of the certificate. The third issue relates on the fact that two high-entropy areas might be adjacent to each other without a sufficient gap and may accidentally get merged into a larger high-entropy area, such as the signature of the certificate and the server key exchange parameters. For the latter two cases, we introduce optional blocks into the signature, making the signature scalable. In the regular expression, we can include optional strings. For example, our TLS signature has been extended to include optional strings as below. This adjustment boosts the recall for most cases, as shown in Table VI. For both cases of 1-4-8 and 1-2-4-8, the recall increases by around 20%.

$$...1\{80,260\}(0\{20,1024\}|\{8,160\}(1\{8,70\}|$$
$$\{8,70\}0\{0,300\}1\{8,70\})0\{0,500\})...$$

The filter threshold ξ is used to remove false positives and make the fingerprint more reliable. As the threshold increases, the detection accuracy of high-entropy blocks will increase as we are eliminating those accidental "high-entropy" blocks. At a certain point, this elimination may have an adverse effect as true high-entropy blocks may be eliminated by a excessively large value of ξ. We experimented with different values of ξ using a 4-bit measure, as shown in Figure 6. Given its initial purpose, this parameter should be kept as small as possible for effective filtering. Thus, $\xi = 9$ is chosen based on the empirical results. As suggested by our test results, it appears to be a proper choice for other measures, e.g. 1-4-8 measure.

τ \ ρ	99.20%	99.85%	99.97%	99.99%
1-bit	12.39%	6.88%	40.37%	40.37%
2-bit	21.10%	19.27%	38.53%	40.83%
4-bit	84.40%	73.39%	33.03%	13.30%
8-bit	67.43%	51.83%	11.01%	16.51%
1-2-8	41.28%	25.69%	18.34%	11.93%
1-4-8	55.05%	**82.57%**	67.43%	12.39%
1-2-4-8	49.54%	**87.61%**	67.43%	12.39%

TABLE VI: Recall using refined fingerprint

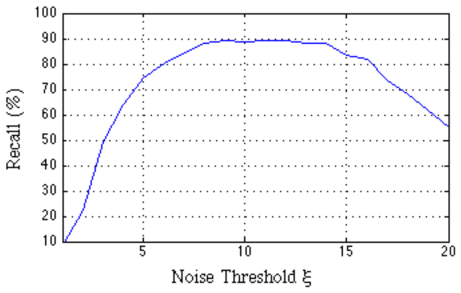

Fig. 6: Noise Threshold Selection over TLS traffic using a 4-bit measure

1) Test Results: After two improvement procedures – signature refinement and parameter selection – the ultimate test over the testing sets, *d00300*, is shown in the table below. The multiple τ-measure 1-4-8 now produces a good recall rate.

ξ=9	TP	FN	Recall
4-bit measure (ρ=99.20%)	1056	104	91.03%
1-4-8 measure (ρ=99.85%)	1079	81	93.02%

Dataset	Total	Positives	Negatives
d00200	1,160	1,079	81
d00300	1,204	61	1,143
d00015	3,412	0	3,412

TABLE VII: TLS signature over different datasets, i.e. *d00200*: TLS w/ selected Ciphers, *d00300*: TLS w/ ohter Ciphers and *d00015*: non-TLS.

Finally, we fixed our filter threshold $\xi = 9$ and used the 1-4-8 measure. We summarize our results over three datasets as follows. Overall, the TLS signature has a precision of nearly 94.6% and its accuracy is around 94%, as we only included the negative cases from *d00300* so as to have a equivalent size of positive cases. On the other hand, negative cases from non-TLS traffic in *d00015*, turned out to be a relatively trivial amount, even though some of instances *do* contain high-entropy traffic, such as SSH traffic on port 22.

C. Application on Botnet Detection: Nugache

The Nugache botnet was one of the first peer-to-peer botnets to use strong cryptography to protect its C&C channel; the inter-peer communication was encrypted using individually negotiated session keys derived using a hybrid RSA/Rijndael scheme [6], [7], [33]. Specifically, Nugache uses a two-way RSA-like key exchange protocol for every session with a minimum length of 512 bits for the modulus. One peer sends the length of the key to announce a peer key exchange, followed by an actual public key [6]; the other peer in turn replies with a message of the same length encrypted with that public key. Compared to TLS, signature extraction for Nugache is much easier because of the simplicity of its key exchange. Since there is little control information in key exchange messages, if we only consider the payload, the signature can be simply defined as 1*, meaning high-entropy blocks throughout, which is also a strong detectable characteristic distinct from other cryptographic protocols. Following the same consideration, we choose $\xi = 9$, which yields a fair recall rate.

τ \ ρ	99.20%	99.85%	99.97%	99.99%
2-bit	**92.21%**	67.90%	39.50%	17.28%
1-2-8	88.27%	**90.12%**	73.46%	56.17%
1-4-8	89.51%	**92.21%**	75.93%	56.17%
1-2-4-8	**90.12%**	**95.06%**	77.16%	56.17%

TABLE VIII: Recall on Nugache (N=32)

The initial fingerprint we generated for Nugache includes two high-entropy areas, corresponding to the two-way key exchange. First, we test all τ-bit measures with a fixed filtering threshold value of ξ=9. It shows the 2-bit measure produces good results (92.21% with $\rho = 99.20\%$) while our voting mechanism clearly outperforms a single τ-bit measure given the same level of confidence. We conservatively choose the 1-4-8 measure as our metric in a general.

Dataset	Desc	Total	Ps	Ns
N	*Nugache*	175	162	13
d00200, d00300	TLS	2,364	0	2,364
d00015	non-TLS	3,412	0	3,412

TABLE IX: Nugache fingerprint over different datasets

In Table IX, we summarize our testing results of the Nugache signature over three datasets. It is encouraging that the Nugache signature generates no false positive and so has a precision of 100%. For obfuscation techniques, there still a portion of the traffic, although small, that will appear to have low entropy.

V. Limitations & Future Work

One may argue that high entropy does not necessarily imply encryption, compressed data, or multimedia data. While this is true, our focus is the distribution of high-entropy data blocks, not solely the presence of high-entropy data. A study [18] provides evidence against such "common sense," where it was shown that multimedia files could yield low entropy instead, although the authors also pointed out that in some cases compressed files do have high entropy. Such cases require a much closer look, which we left for future work. Furthermore, character encodings such as base64 [34] can significantly reduce the entropy of a string. For this case, we assume that a base64 detector as well as a decoder could be deployed to canonicalize the traffic data; future experiments can evaluate that effectiveness. It is also possible that one could easily inject arbitrary bytes to disturb the original distribution of high entropy and low entropy. In this case, we consider it to be a new protocol for which the traffic could be possibly fingerprinted, e.g. using optional units as we did for TLS. It does indicate the open problem of the detection of mutated protocols deployed by botmasters in order to avoid detection and identification. If the signature generation process is automated, then this approach would still be efficient. However, if more advanced obfuscation techniques such as those outlined in [15], [16] are applied, then our approach will fail at identifying the obfuscated protocol. Nevertheless, our proposed techniques may be still used to detect the obfuscation techniques themselves.

To avoid being fingerprinted, malware could adopt plain TLS instead of customizing the protocol, running the risk of SSL inspection. This warrants explanation, as it may explain why there only 10% of malware samples utilize TLS. Nevertheless, other research [5] also found that malware or botnets that *do* utilize TLS tend to do so in a very customized way, such as by advertising significantly fewer cipher suites than enterprise TLS clients. A shorter list of cipher suites will reduce the control information (i.e. low-entropy blocks) and therefore may result in different fingerprints than enterprise-grade TLS clients. Investigating how effective our approach would be in such a scenario is left for future work. Under certain circumstances, it is possible that our approach may not be sufficient to rule out all possible false positives and we could extend this work by examining how our our approach works in synergy with other tools in order to reduce false positives and generate more accurate fingerprints.

As we previously stated, there are other entropy estimators aside from MLE. Given what was explored in [28], such as alternative estimation methods and entropy predictors, a future study could extend our method to other entropy calculations. Those alternate calculations could be evaluated in order to help tune our method to provide the best results.

Finally, we are interested in looking at more diverse data, such as compressed data, SSH, and other malware traffic. Examining both benign and malicious data in multiple settings will allow us to fine-tune our approach and make it resilient against unknown cipher suites or obfuscation.

VI. Conclusion

In this paper, we proposed a novel voting-based method for accurately detecting high-entropy blocks of data in a network traffic stream, and a method based on regular expressions for generating a scalable fingerprint of that traffic. Our approach can effectively put malware authors on the defense, as a longer key used for a more securely encrypted connection would make it more easily characterized and therefore more detectable. If a shorter key is used for making the connection less vulnerable to detection, then the malware author would only achieve a less secure connection. Our technique can be used to augment other detection and classification techniques to create more comprehensive tools.

References

[1] A. Freier, P. Karlton, and P. Kocher, "RFC 6101 (historic): The Secure Sockets Layer (SSL) Protocol Version 3.0," http://tools.ietf.org/html/rfc6101, August 2011.

[2] T. Dierks and E. Rescorla, "RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2," http://tools.ietf.org/html/rfc5246, August 2008.

[3] A. Freier, P. Karlton, and P. Kocher, "RFC 4251: The Secure Shell (SSH) Protocol Architecture," http://http://tools.ietf.org/html/rfc4251, Jan 2006.

[4] E. Rescorla, "RFC2631: Diffie-Hellman Key Agreement Method," https://tools.ietf.org/html/rfc2631, 1999.

[5] B. Anderson, S. Paul, and D. McGrew, "Deciphering malware's use of TLS (without decryption)," *arXiv preprint arXiv:1607.01639*, 2016.

[6] D. Dittrich and S. Dietrich, "P2P as botnet command and control: a deeper insight," in *Proceedings of the 3rd International Conference on Malicious and Unwanted Software (Malware)*, 2008.

[7] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich, "Analysis of the Storm and Nugache Trojans: P2P is here," in *USENIX ;login: vol. 32, no. 6*, December 2007.

[8] T.-F. Yen and M. K. Reiter, "Traffic aggregation for malware detection," in *Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment conference (DIMVA)*. Springer, 2008, pp. 207–227.

[9] J. Olivain and J. Goubault-Larrecq, "Detecting subverted cryptographic protocols by entropy checking," Laboratoire Spécification et Vérification, ENS Cachan, France, Research Report LSV-06-13, Jun. 2006, 19 pages.

[10] P. Dorfinger, G. Panholzer, and W. John, "Entropy estimation for real-time encrypted traffic identification," in *Proceedings of the Third International Conference on Traffic Monitoring and Analysis*. Springer-Verlag, 2011, pp. 164–171.

[11] A. M. White, S. Krishnan, M. Bailey, F. Monrose, and P. A. Porras, "Clear and present data: Opaque traffic and its security implications for the future." in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2013.

[12] S. Smith, S. Neyens, and R. Hammell II, "The use of entropy in lossy network traffic compression for network intrusion detection applications," in *International Conference on Cyber Warfare and Security*. Academic Conferences International Limited, 2017, pp. 352–360.

[13] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel traffic classification in the dark," in *Proceedings of SIGCOMM '05*. New York, NY, USA: ACM, 2005, pp. 229–240.

[14] C. V. Wright, F. Monrose, and G. M. Masson, "On inferring application protocol behaviors in encrypted network traffic," *J. Mach. Learn. Res.*, vol. 7, pp. 2745–2769, Dec. 2006.

[15] R. Dingledine, "Obfsproxy: The next step in the censorship arms race." *https://blog.torproject.org/blog/ obfsproxy-next-step-censorship-arms-race*, 2012.

[16] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Protocol misidentification made easy with format-transforming encryption," in *Proceedings of the 20th ACM SIGSAC conference on Computer and Communications Security (CCS)*. ACM, 2013, pp. 61–72.

[17] L. Wang, K. P. Dyer, A. Akella, T. Ristenpart, and T. Shrimpton, "Seeing through network-protocol obfuscation," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2015, pp. 57–69.

[18] H. Zhang, C. Papadopoulos, and D. Massey, "Detecting encrypted botnet traffic," in *Proceedings of the IEEE INFOCOM*, 2013, 6 pages.

[19] H. Zhang and C. Papadopoulos, "Early detection of high entropy traffic," in *IEEE Conference on Communications and Network Security (CNS)*, Sept 2015, pp. 104–112.

[20] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "BotHunter: Detecting Malware Infection Through IDS-driven Dialog Correlation," in *Proceedings of the 16th USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–16.

[21] T. J. Richer, "Entropy-based detection of botnet command and control," in *Proceedings of the Australasian Computer Science Week Multiconference*, ser. ACSW '17. New York, NY, USA: ACM, 2017, pp. 75:1–75:4. [Online]. Available: http://doi.acm.org/10.1145/3014812.3014889

[22] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, July, October 1948.

[23] A. Antos and I. Kontoyiannis, "Convergence properties of functional estimates for discrete distributions," *Random Structures & Algorithms*, vol. 19, no. 3-4, pp. 163–193, Oct. 2001.

[24] G. A. Miller, "Note on the bias of information estimates," *Information Theory in Psychology: Problems and Methods*, pp. 95–100, 1955.

[25] B. Efron and C. Stein, "The jackknife estimate of variance," *The Annals of Statistics*, vol. 9, pp. 586–596, May 1981.

[26] L. Paninski, "Estimation of entropy and mutual information," *Neural Computation*, vol. 15, no. 6, pp. 1191–1253, Jun. 2003.

[27] T. Schürmann, "Bias analysis in entropy estimation," *J. Phys. A Math. Gen*, pp. 295–301, 2004.

[28] M. S. Turan, E. Barker, J. Kelsey, K. McKay, M. L. Baish, and M. Boyle, "Recommendation for the entropy sources used for random bit generation," *NIST Special Publication 800-90B*, 2018, available at https://doi.org/10.6028/NIST.SP.800-90B.

[29] G. Combs, "Wireshark: The network protocol analyzer," http://www.wireshark.org/.

[30] V. Jacobson, C. Leres, and S. McCanne, "tcpdump, a powerful command-line network analyzer," http://www.tcpdump.org.

[31] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide scanning and its security applications," in *Proceedings of the 22nd USENIX Security Symposium*, August 2013.

[32] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Military Communications and Information Systems Conference (MilCIS)*. IEEE, 2015, pp. 1–6.

[33] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. Dietrich, and H. Bos, "SoK: P2PWNED - modeling and evaluating the resilience of peer-to-peer botnets," in *2013 IEEE Symposium on Security and Privacy*, May 2013, pp. 97–111.

[34] S. Josefsson, "RFC4648: The Base16, Base32, and Base64 Data Encodings," http://tools.ietf.org/html/rfc4648, 2006.