Received September 14, 2020, accepted October 4, 2020, date of publication October 13, 2020, date of current version October 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3030559

Rethinking the Weakness of Stream Ciphers and Its Application to Encrypted Malware Detection

WILLIAM STONE[1], DAEYOUNG KIM[1], VICTOR YOUDOM KEMMOE[1], MINGON KANG[2], AND JUNGGAB SON[1], (Member, IEEE)
[1]Department of Computer Science, Kennesaw State University, Marietta, GA 30060, USA
[2]Department of Computer Science, University of Nevada Las Vegas, Las Vegas, NV 89154, USA

Corresponding author: Junggab Son (json@kennesaw.edu)

This work was supported by the Ministry of Science and ICT (MSIT), South Korea, through the High-Potential Individuals Global Training Program supervised by the Institute for Information and Communications Technology Planning and Evaluation (IITP) under Grant 2019-0-01601. The preliminary version of this article was presented in the Proceedings of 2019 International Conference on Computing, Networking and Communications (ICNC'19) [1].

ABSTRACT One critical vulnerability of stream ciphers is the reuse of an encryption key. Since most stream ciphers consist of only a key scheduling algorithm and an Exclusive OR (XOR) operation, an adversary may break the cipher by XORing two captured ciphertexts generated under the same key. Various cryptanalysis techniques based on this property have been introduced in order to recover plaintexts or encryption keys; in contrast, this research reinterprets the vulnerability as a method of detecting stream ciphers from the ciphertexts it generates. Patterns found in the values (characters) expressed across the bytes of a ciphertext make the ciphertext distinguishable from random and are unique to each combination of ciphers and encryption keys. We propose a scheme that uses these patterns as a fingerprint, which is capable of detecting all ciphertexts of a given length generated by an encryption pair. The scheme can be utilized to detect a specific type of malware that exploits a stream cipher with a stored key such as the DarkComet Remote Access Trojan (RAT). We show that our scheme achieves 100% accuracy for messages longer than 13 bytes in about 17 μsec, providing a fast and highly accurate tool to aid in encrypted malware detection.

INDEX TERMS Encryption, Intrusion Detection, malware, network security, stream ciphers.

I. INTRODUCTION

Key generation and use are the most significant factors in the security of a stream cipher; consequently, they are also the source of many vulnerabilities for the cryptosystem [2]. It is a well-known fact that encrypting multiple messages using a stream cipher with the same key could lead to an unintended leakage of information, e.g., the many time pad (MTP) attack [3]. There are many examples of attacks which make use of the MTP or related keys to break the cryptosystem, i.e., recovering plaintexts and/or encryption keys [4], [5]. However, we reinterpret this weakness from the new perspective of ciphertext classification. In a stream cipher, the encryption process represents a deterministic mapping of the transition of one specific value to another on a byte-by-byte basis. Encrypting a certain value repeatedly using the same combination of a cipher and key results in the output of the same value each time, and also, encrypting multiple values using the same combination reveals patterns in the distributions of values represented at each byte of the generated ciphertexts.

Utilizing these patterns and the principles of *perfect secrecy* [6], this paper proposes a ciphertext discrimination function: a machine learning algorithm with the capability to classify a ciphertext as either likely to have been generated by a stream cipher and key pair or not. The discrimination function is developed using a Bayesian approach where the likelihood that a collected message was generated by a given cipher and encryption key can be determined using a model trained from ciphertexts generated by the same combination of encryption elements. Then, the discrimination function compares the calculated likelihood of the collected message against a message length-based threshold in order to classify the message into one of the two groups. Since the only assumption made is that the encryption scheme and key are held constant, the detection scheme can be extended to conditions where the key is not explicitly known.

One promising application of the discrimination function is the detection of *encrypted malware*. Encrypted malware is malicious software that utilizes encryption techniques to

The associate editor coordinating the review of this manuscript and approving it for publication was Tai-Hoon Kim.

hide its malicious intentions from a method of detection such as traditional network monitoring. Since it cannot decrypt ciphertexts for inspection, an encrypted malware packet will appear as normal, benign communication to an unintelligent monitor [7]. Developing a method to detect malicious activity without necessitating the decryption of each message is crucial. Decrypting each message individually is expensive, further reduces message integrity, and is impossible without the monitoring service knowing the scheme and key used. The ciphertext discrimination function proposed in this paper presents such a method. Training the monitor to detect encrypted malware packets without decryption also adds the notable benefits of fast and accurate inspection while protecting the monitoring station from potential threats that may occur during the process of inspection. Although there is no generic solution that can detect all types of malware, our proposed scheme can effectively detect a specific type of malware such that it transfers encrypted packets using a stream cipher with a fixed encryption key. Figure 1 illustrates an encrypted malware detection scenario using the discrimination function.

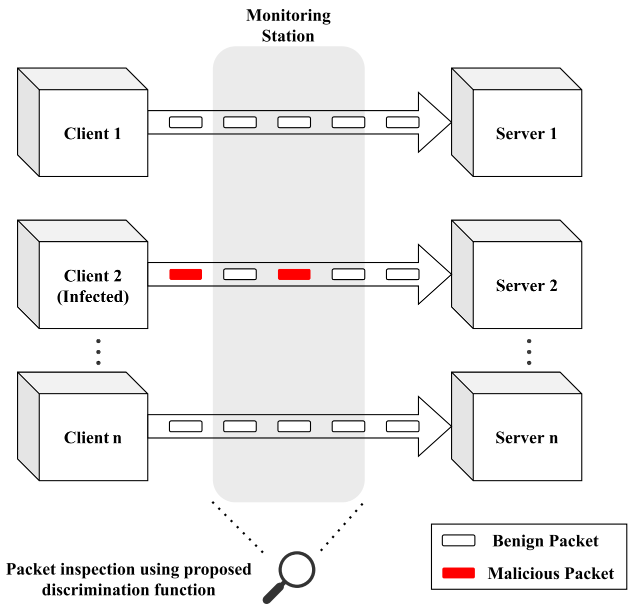


FIGURE 1. Conceptual overview of a trained network monitor detecting an infected machine from packets sent over the network.

Our simulation results show that the discrimination function is capable of classifying ciphertexts from the following stream ciphers: RC4, ChaCha20-Poly1305, and Salsa20. Furthermore, it works well with a subset of block cipher modes of operation that operate similarly to stream ciphers, such as Counter (CTR), Galois/Counter (GCM), and output feedback (OFB) modes. To demonstrate that the discrimination function is effective in detection of encrypted malware packets, we perform a comprehensive experiment with the Remote Access Trojan (RAT) DarkComet. DarkComet is known to use RC4 to encrypt its communication between attacking and victim machines using a static key [8], [9]. The results show that the proposed scheme is highly fast and accurate, as it achieves an accuracy of 100% for messages longer than 13 bytes, completing detection in approximately 17 microseconds (μsec) on an Intel Core (TM) i7-8700 processor.

The remainder of this paper is organized as follows: a literature review presented in Section II, followed by Section III which describes the statistical weakness found in stream ciphers that allows us to determine the discrimination function. Section IV outlines the scheme we have implemented to perform the detection and a discussion of the testing and evaluation of the scheme is provided in Section V. An example of deploying the scheme to detect encrypted malware packets from a real-world simulation is given in Section VI. Finally, in Section VII, we give some final statements and possible direction for future works.

II. RELATED WORKS

To provide insight into the current state of malware and detection, a comprehensive literature survey has been conducted. In this section, we introduce works related to the topics of known weaknesses of stream ciphers, encryption techniques used in malware, and more specifically, the utilization of stream ciphers in encrypted malware.

A. WEAKNESSES OF AND ATTACKS ON STREAM CIPHERS

The vulnerabilities of stream ciphers often originate during the generation of the key stream [10]. The cipher must create a key stream equal in length to the plaintext that is to be encrypted and the key stream must appear random. Since this is completed through various algorithms and not generated truly randomly in practical applications, certain insecurities may propagate through the encryption scheme. Here we will present works that study some of these vulnerabilities and attacks which exploit them.

In [11], F. Armknecht presented the results of algebraic attacks on stream ciphers, where he outlines a theoretical attack on a Bluetooth encryption system. ChaCha20-Poly1305 is commonly used for Secure Shell (SSH) and Transport Layer Security (TLS) secure communication tools. In [12], the authors exposed a vulnerability of OpenSSH and OpenSSL that allows for the discovery of cryptographic artifacts existing in memory, providing an interested party with the ability to crack secure-tunneled communications by targeted memory extraction. B. Jungk and S. Bhasin proposed potential power and electromagnetic side-channel attacks on ChaCha20-Poly1305 in [13].

From the time it was leaked to the public in 1994, RC4 has been under research scrutiny in attempts to uncover potential vulnerabilities. Jindal and Singh surveyed RC4, detailing the implementation of the cipher, its application, and some of the well-known weaknesses in [14]. They classified the different vulnerabilities in the following manner: weak keys (set of keys in the cipher which leave a trace in the keystream or output bytes), key collisions (the generation of similar output states from two different keys), key recovery from the keystream, state recovery (recovering the internal state of the cipher), and biased bytes (an event produces a nonuniform probability which does not follow the expected randomness of byte production). A few notable weaknesses which have been identified and well-documented include: bias of the second byte towards 0 (biased bytes) [5], the initial byte generated by the Key Scheduling Algorithm is highly related to a few bytes from the key as discovered by Roos in 1995 (weak keys) [15], and the discovery that only a few keys may determine the output state and many output bits with significant probability by Fluhrer, Mantin, and Shamir (weak keys) [16]. A practical key recovery attack is described in [17]. Exploiting the potential for key collision, the secret key can be discovered in non-negligible time when the key is sufficiently large in a *related-key model*. The authors of [18] investigated event outcomes of the RC4 stream cipher, reporting on non-randomness and biases that further contribute to the cipher's insecurity. During their research, they were able to define a bias created by the length of the secret key used to create the cipher's keystream, which was then used for proofs of attacks on Wired Equivalent Privacy (WEP) and Wireless Protected Access (WPA). The bias discovered shows a correlation between the length of the secret key (ℓ) and the ℓ-th byte of the keystream. Vanhoef and Piessens introduced attacks on WPA-TKIP and TLS, which employ RC4, where they proposed a *fixed-plaintext algorithm* that returns a set of probable plaintexts [19]. The authors break WPA-TKIP by using biases that they detect empirically through statistical analysis, allowing them to uncover the TKIP MIC key.

B. USE OF ENCRYPTION IN MALWARE DEVELOPMENT

Various studies have been conducted on the subject of encryption employed by malware. In 2007, Martignoni *et al.* proposed a malware detection technique referred to as *Omni-Unpack* which attempted to solve the problem of packed malicious software or malware whose payload was either encrypted or compressed in an attempt to hide its presence [20]. In polymorphic malware applications, the programmer may alter the encryption or compression, making traditional signature detection difficult to impossible; the authors investigated a method to unpack the malware and expose the original source code so that a software scanner could identify the malware based on the original signature. In [21], R. Zhao *et al.* proposed a new approach to detecting the encryption functions within network applications. Through dynamic taint and data pattern analysis, the authors were able to detect various encryption routines, including RC4, which can be used in signature detection of the malware.

While malware may employ encryption techniques for the purpose of polymorphism, it may also hide its communications using secure traffic protocols. A research series performed by a group at Cisco studied the use of encryption and TLS in malware [22]. Through this work, a random forest classifier was ultimately created which distinguishes TLS flow generated by malware [23]. In [24], Prasse *et al.* proposed a Long Short-term Memory neural network which uses only observable features of HTTPS traffic (client and host IP addresses and ports, timestamps, data flow volume, and the unencrypted host domain name) for malware classification, claiming it outperforms random forest models in similar applications with a 64% detection rate and 70% precision. The authors of [25] proposed a distance-based, supervised learning solution which suffers from the pitfall of relying on collecting a large amount of traffic data and extracting the features before any analysis can be completed. An approach that analyzes persistent communications, instead of the presence of anomalies within the persistent communication itself, is offered in [26].

Encryption is also a critical aspect of ransomware applications. When a machine is infected with this type of malware, the data stored on the machine is encrypted and the secret key which unlocks the data is held at ransom. Surveys on the history and growth of ransomware are presented in [27] and [28], which include details on the different families of ransomware, notable attacks, and prevention techniques. An early-warning scheme which allows for the detection of potential ransomware infection was developed in [29] using the following indicators: file type changes, similarity measurement between original data and its encrypted version, and Shannon entropy. The authors determined these indicators from the actions that each class of ransomware performs upon deployment.

C. MALWARE EMPLOYING STREAM CIPHERS

Stream ciphers are often chosen as the means of encryption in malware applications due to their speed and ease of implementation. RC4 is among the most popular choices, having a few notable examples such as DarkComet, Zeus, Citadel, and Dridex. DarkComet has been used in an array of examples, ranging from amateur attacks on personal data to state surveillance [30], [31]. The Zeus malware, which emerged in the late 2000s, is a trojan horse malware that served as a significant threat in the banking industry. Binsalleeh *et al.* provided an in-depth analysis of the Zeus botnet in their paper [32]. Through reverse-engineering of the toolkit, the authors were able to uncover the encryption key and a method to thwart the malware's HTTP communications through the injection of false information. It was found that Zeus packets contain information about the length of the packet and that upon XORing the ciphertext and plaintext, the "stream key" can be found and used as a method for detecting encrypted Zeus packets [33]. A framework for the detection of the DarkComet RAT was developed in [34], with the authors claiming 95.23% detection accuracy.

While RC4 appears to be the dominant stream cipher used in communication obfuscation, likely due to its legacy role in securing network traffic, other contemporary stream ciphers used in malware development may be found in crypto-ransomware applications. A survey of trends in ransomware development is given in [35]; a few malware packages provided as examples are also listed with their respective encryption schemes (RC4, BlowFish, and Salsa20) and how the key is derived in each of these examples. A novel approach to discover the artifacts of malware communications using crypto-libraries found in Microsoft Windows is described in [36]. Their method does not require any prior knowledge and is extensible to other implementations in the cipher suite of TLS, such as AES and ChaCha20; AES-GCM is used for all malware samples sourced in their approach. The ransomware *LockerGoga* was used to attack Norsk Hydro in 2019; the authors of [37] found the scheme used to encrypt the affected data was AES with CTR mode.

III. STATISTICAL WEAKNESSES OF STREAM CIPHERS

We use the term statistical weakness to describe a scenario in which the difference between the probability of an event occurring in theory and the probability of the event that is actually observed is significant. In the domain of encryption, this is most likely attributed to the creation and introduction of randomness. The primary goal of a cipher is to obfuscate a message to ensure privacy: only the author of the message and privileged personnel (those who have the appropriate decryption key) are able to read its contents. This is achieved by making the message appear completely random to any potential adversary or entity who does not possess the appropriate decryption key.

In our preliminary research, we discovered that the RC4 stream cipher generates unique patterns of ciphertexts under the following conditions: encryptions are computed under a fixed key and that the plaintext message and key both be derived from the same set of literary characters [1]. This section will provide more details regarding the preliminary results, the extension of the discovery to other popular stream ciphers, and explain how these patterns occur.

A. PRELIMINARY RESULTS

This paper is a continuation of the findings presented in [1]. As a result of their work, the authors present a machine learning algorithm which can be used to identify RC4 ciphertexts based on the probability of characters observed in the message. If ASCII-encoded messages are repeatedly encrypted under a fixed ASCII-encoded key, each byte of the resulting ciphertexts will produce a unique pattern. Each pattern represents the values that could possibly be expressed at that byte of the message. As more messages are analyzed, the patterns can then be used as probability distributions which represent the probability that a value may be found at any given byte. Because the patterns are unique and non-random, it is possible to train a binary classifier to identify ciphertexts using the patterns to determine the likelihood that a ciphertext was generated via RC4.

B. PATTERNS GENERATED BY STREAM CIPHERS

Before continuing on to describe the source of these patterns and the results of the other ciphers that were studied, it is important to first introduce the notation we will use for the remainder of this document. We define a message generically as x having length L. Each character of x is symbolized as x_i, where x_i is the character at the i-th byte (i.e. position) of the message such that $0 \leq i < L$ and $x_0 \leq x_i < x_L$. More specifically, we will refer to a plaintext or ciphertext message as p or c, respectively. A stream cipher, E, encrypts x under its secret key k written as: $E(k, x)$ or we may also denote the cipher mechanism and its key as E_k.

Human languages are defined over a set of characters. In the English language, both cases of all the alphabetical characters, Arabic numerals, and common symbols create a set of length 95, known as *printable characters*. A computer interprets these symbols through their binary representation, which is translated through an encoder. The ASCII encoding scheme defines characters using one byte which provides a range of 256 possibilities. However, as we have described, only 95 values are printable or readable by humans. In ASCII, these values span the range of $32\text{-}126_{10}$. This discrepancy can be used to our advantage because we know that a stream cipher encrypts messages character-by-character via a deterministic mapping of a plaintext character to a ciphertext character. This means that because the plaintext values originate from this truncated range, the ciphertext characters must also exist in a subset of equally reduced size. So for any given message encoded by ASCII, there are only 95 possible values for each character, and consequently, only 95 values for each character in the ciphertext produced by a stream cipher. We can use this property to generate patterns for byte values expressed in the ciphertext which are unique and representative of a stream cipher and encryption key pair.

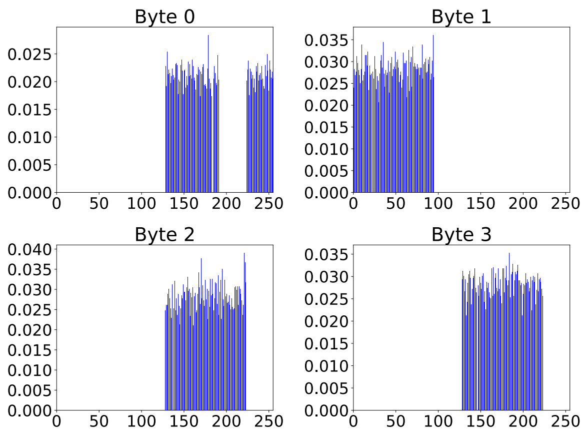


FIGURE 2. The distributions of values expressed over the first four bytes of ChaCha20-Poly1305 ciphertexts (x-axis: byte value in the range of 0 to 255, y-axis: probability).

Beyond RC4, we have found that other stream ciphers or block ciphers whose mode of operation causes it to perform like a stream cipher also generate similar statistical patterns. We extended the study into the following schemes: ChaCha20-Poly1305, Salsa20, and the Counter, Output-Feedback, and Galois/Counter modes of operation. Figures 2 through 6 show the patterns generated under our testing conditions. The figures were generated through fixed-key encryption of 10,000 random messages containing the full

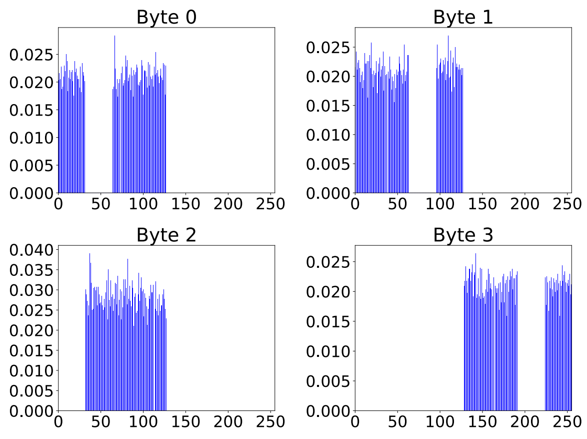


FIGURE 3. The distributions of values expressed over the first four bytes of Salsa20 ciphertexts (x-axis: byte value in the range of 0 to 255, y-axis: probability).

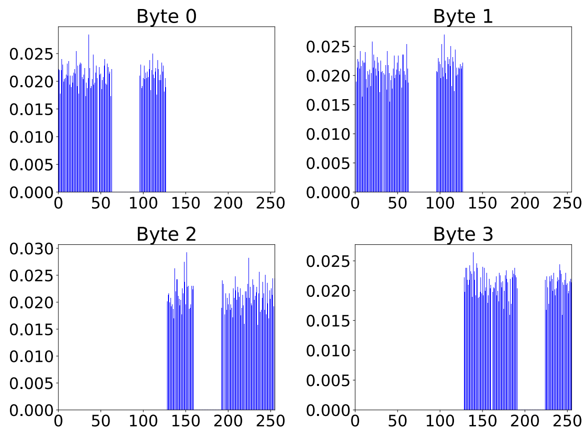


FIGURE 4. The distributions of values expressed over the first four bytes of AES-CTR ciphertexts (x-axis: byte value in the range of 0 to 255, y-axis: probability).

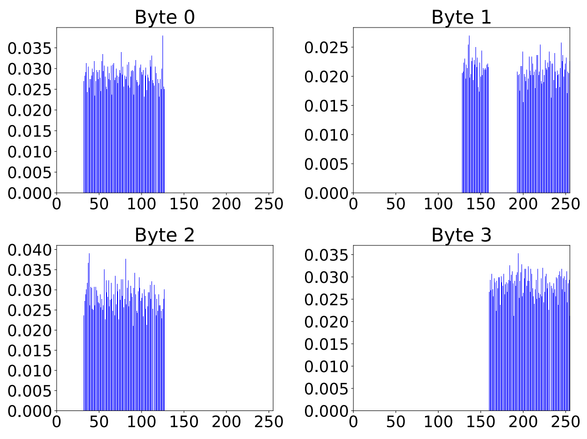


FIGURE 5. The distributions of values expressed over the first four bytes of AES-OFB ciphertexts (x-axis: byte value in the range of 0 to 255, y-axis: probability).

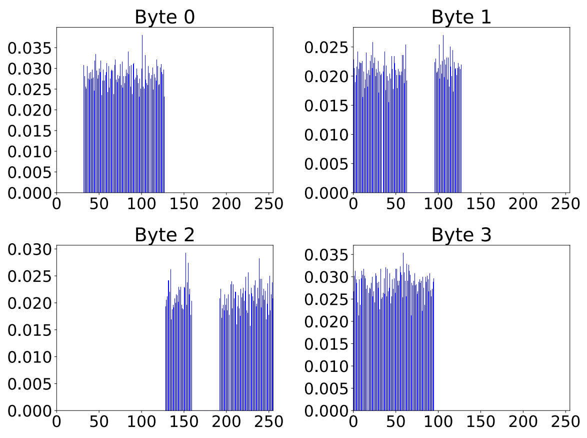


FIGURE 6. The distributions of values expressed over the first four bytes of AES-GCM ciphertexts (x-axis: byte value in the range of 0 to 255, y-axis: probability).

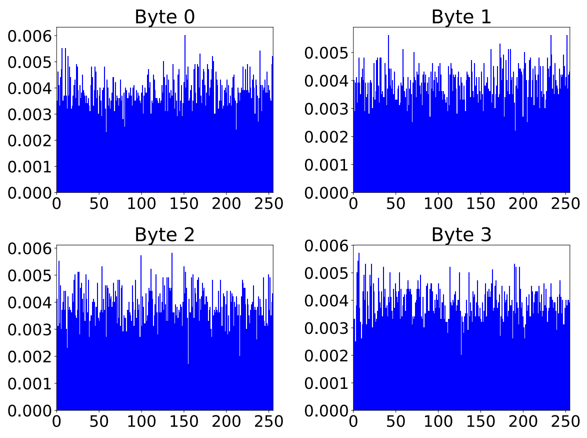


FIGURE 7. The distributions of values expressed over the first four bytes of AES CBC mode ciphertexts (x-axis: byte value in the range of 0 to 255, y-axis: probability).

array of printable ASCII values. Compare the distributions of these figures with those of Figure 7. The results of the latter are generated by AES operating under the standard Cipherblock-Chaining (CBC) mode which performs like a traditional block cipher. Here, there is no distinguishable pattern and observed values are well-distributed over the range of the 256 possible values.

C. USING PATTERNS TO DETECT CIPHERTEXTS

Because these distinct patterns are unique to a given encryption scheme and key, we can use them as a fingerprint for the detection of the encrypting pair. The patterns can then be used to construct a model capable of predicting the likelihood that a message was encrypted by the given scheme under the specified key. Using the results of the analyzed ciphertexts, we can record the values at the i-th positions of each message x with matrix $M = [x_{ij}]$, where j is the decimal representation of the character found at the i-th position of x such that $0 \leq j \leq 255$. The indices of M are used as frequency bins, so upon analysis of x_i, we determine the decimal representation of the character, j, and update M by incrementing the value at position (i, j). This way, M may be used to determine whether or not a character will appear at a specified byte in the ciphertexts generated by an encryption pair or the probability of that character existing at that byte.

IV. STREAM CIPHERTEXT ANALYSIS SCHEME

In this paper, we propose machine learning-based schemes that can detect ciphertexts generated by stream ciphers. Specifically, we present two training scenarios: (a) when an encryption key is easily obtainable and (b) when an encryption key is not obtainable. Then, we present two detection scenarios: (i) when a complete set of ciphertext is available, and (ii) when only a partial ciphertext is available.

A. NOTATION

For any byte c_i in the ciphertext c, let the probability of the value occurring at that byte be defined as $P(c_i)$. The conditional probability that c is generated by encryption scheme E under encryption key k is denoted as $P(c|E_k)$ and that the value found at the i-th byte of the ciphertext given E_k is denoted as $P(c_i|E_k)$.

B. TRAINING PHASE

We consider two scenarios for modeling the ciphertexts generated by a stream cipher and key pair: when the encryption key is known and when it is not. Because the mechanism of the stream cipher creates these patterns under a fixed key, the key is not necessary to perform the analysis. Training the models in either scenario is the same; the only step that changes is how the data used to train the model is obtained. In the first scenario, knowledge of the key allows us to create the optimal dataset because we are able to deduce precisely which bytes the cipher will be directly mapped to. We can use the key to generate all possible encryptions deterministically, allowing the model to be trained from theoretical calculations. In the second scenario, when the key is unknown, the patterns must be determined empirically.

1) BUILDING A MODEL

In order to exploit the statistical weakness of stream ciphers as a means for detection as described in Section III-C, we construct our model around the matrix M. How we construct M is dependent on our access to the encryption key; however, in either case, the cells of M are used to collect the values observed at each byte of each message in the observed data.

Let us first consider the case in which the key is known. In this case, we are able to execute a type of chosen plaintext analysis of the cipher, encrypting every possible printable value. That is, a character x_j in the printable ASCII range generates the character c_j when encrypted under E_k, or $c_j = E(k, x_j)$, where $32_{10} \leq x_j < 127_{10}$. Each message is the repeated character x_j, and we perform this for each character (i.e. 95 times). This allows us to find the encryption of each possible character at each position over the length L. We take each c_{ij} generated during this analysis and use it to populate the values of M, where c_{ij} is the value j found at the i-th byte of c. The bins of M are initialized to 0 and are updated by incrementing at the position (i, j) accordingly.

Algorithm 1 Proposed Training Model

```
    Generate training data:
1:  if a key is known then
    Generate $D \supset E(k, m_i)$, $A_{32} \leq m_i \leq A_{127}$, where $A$ is the
    set of all ASCII characters represented in decimal form
    and $m_i$ is the repeated character found at $A_i$
2:  else
    Generate $D$, through network monitoring. Captured sus-
    picious messages are formatted such that each character
    in the message is represented as its respective decimal
    value
3:  end if
    Construct training model:
4:  $M = \sum_{i=0}^{L} \sum_{j=0}^{255} a_{ij} = 0$
5:  for message ∈ D do
6:      $l = 0$
7:      for character ∈ message do
8:          $M[l][character] + +$
9:          $l + +$
10:     end for
11: end for
    Calculate probability distributions:
12: for position ∈ M do
13:     $sum = \sum_{i=0}^{L} position[i]$
14:     for val ∈ position do
15:         $val = val \div sum$
16:     end for
17: end for
```

Without access to the encryption key, we are unable to execute the same test in indistinguishable time. Instead, let us assume that we are able to capture a sufficiently large set of ciphertexts generated under our specified conditions of using a fixed cipher and key. Using these samples, we are able to iterate over each character in each message of the set, populating M through the same analysis of incrementing the c_{ij}-th position of M. Algorithm 1 details the steps taken to construct the model in either training scenario.

2) FINDING PROBABILITY IN PATTERNS

After generating M, we now have a collection of encryptions for all printable ASCII characters. Due to the nature of stream ciphers, the encrypted value can only exist as one of 95 values and because encrypted messages are generated using a fixed key, we can observe a one-to-one relationship between x and c. So using the patterns created by M, we can determine the likelihood that c was generated by E_k.

First, the elements of M are converted from sums of observations to their respective frequencies by dividing each element by the total messages which were analyzed. Now, each element in M is equivalent to $P(c_{ij}|E_k)$, or the probability that we would expect the observed value at each byte given the cipher and encrypting key. In order to determine the probability that an arbitrary message was generated by E_k,

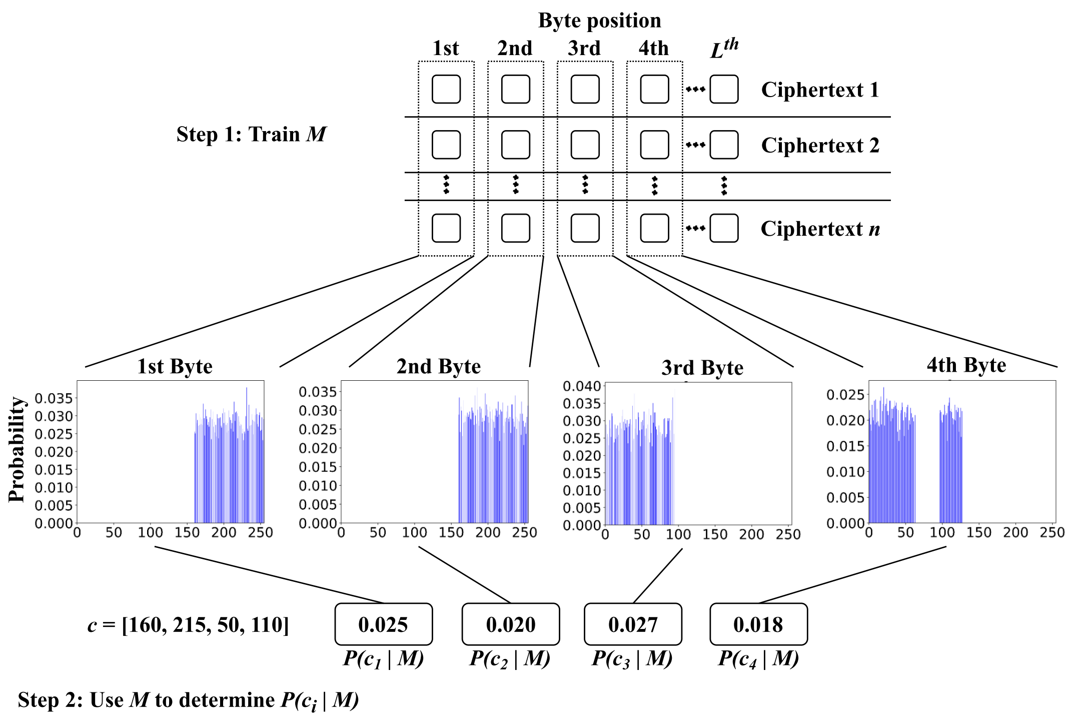


FIGURE 8. Conceptual overview of the proposed scheme. Frequencies of values expressed at each byte across *n* ciphertexts are collected in matrix *M*. $P(c_i|M)$ is determined by looking up the value at $M[i, c_i]$. (x-axis: byte value in the range of 0 to 255).

we calculate the a priori likelihood using Bayes' Theorem (1). Once $P(E_k|c)$ is calculated, we use the discrimination function (2) to determine the binary classification of the result, where θ is the classification rule also referred to as the *threshold*.

$$P(E_k|c) = \frac{P(c|E_k) \cdot P(E_k)}{P(c)}. \tag{1}$$

$$P(E_k|c) \underset{\neg E_k}{\overset{E_k}{\gtrless}} \theta \tag{2}$$

The independent probabilities of $P(E_k)$ and $P(c)$ are constant. The probability of c being encrypted by E_k and not being encrypted by E_k are equal, i.e. $P(E_k) = P(\neg E_k)$. Assuming a uniform distribution of the possible byte-values of c (i.e. if c was generated truly randomly), any ciphertext c_1 is just as likely to exist as another ciphertext c_2. Thus, we can simplify Equation (1) to the proportional relationship:

$$P(E_k|c) \propto P(c|E_k). \tag{3}$$

That is, the probability of E_k given c is proportionally related to the probability of c given E_k. We can use this relationship and Bayes' assumption of event independence to define the probability of a ciphertext c given an encryption scheme E with key k as the product of probabilities of the value observed at each byte of c over its length L:

$$P(c|E_k) = \prod_{i=1}^{L} P(c_i|E_k). \tag{4}$$

Our model performs a simple binary classification: a message is generated by E_k or it is not. We show that after a minimum number of bytes, messages are distinguishable from random in Section V which allows us to determine θ as a point on a line which linearly separates the two classes.

C. DETERMINING THRESHOLDS

Due to the variable length of the captured information, it is crucial that we determine θ as a function of L. We determine an optimal decision threshold from the theoretical probability of a message with length L if it was produced by a truly random, uniform distribution. This uniform distribution is what Claude Shannon concludes in his definition of perfect secrecy. To appear random, the probability of all possible events or encryptions must be equal [6]. In this case, each event can be said to have probability of 1 divided by the number of possible events.

Let us consider a ciphertext c with length L. Because a stream cipher encrypts byte-by-byte, we can consider each character of c, c_i, as an individual event. In Section III-B, we described how the ASCII encoding scheme uses a one-byte binary number to represent all possible characters. In theory, this should allow for 256 possible values to be represented by a single character such that each c_i, can be considered to have probability $P(c_i) = 1/256$. For a number to be truly randomly-generated, each event must be independent of another. Considering each byte independently, we can calculate the probability of any c as the product of the independent probabilities of each c_i:

$$P(c) = \prod_{i=0}^{L} P(c_i). \tag{5}$$

Because we are calculating small probabilities, we consider the log-likelihood as:

$$P(c) = \prod_{i=0}^{L} \ln(P(c_i)). \tag{6}$$

In the case that c was generated randomly and encoded in ASCII we can determine the probability of c as:

$$P(c) = \prod_{i=0}^{L} \frac{1}{256} \; or \; \frac{1}{256} \times L. \tag{7}$$

However, we have described a weakness of the stream cipher that, when paired with an encoding scheme such as ASCII, only produces values over a truncated range due to only a fraction of the possible values being printable. The number of printable values in ASCII is only 95, meaning that when c is generated by a stream cipher each character effectively has the probability of $P(c_i) = \frac{1}{95}$ and the probability of c is $P(c) = \frac{1}{95} \times L$. As a result of the difference between $P(c_i), 0 \leq c_i < 256$ and $P(c_i), 0 \leq c_i < 95$, we are able to calculate an optimal threshold that perfectly separates both classifications.

The optimal threshold may be found by calculating (7) for the length of the message currently evaluated. This threshold is optimal because the calculated probability returned by the equation is the $P(c)$ when each c_i is uniformly distributed. Any deviation from this value indicates that the message being evaluated is not truly random.

We can also determine a solution to calculate θ from the training data. In some monitoring scenarios, certain messages may be repeated periodically (i.e., a connection status or keepalive message). By calculating θ from the collected data, we can incorporate these repeated messages in determining the likelihood that a message belongs to the encryption pair. First, $P(c|E_k)$ is calculated for each message in the training dataset. Because we expect all messages in the set to be generated under the same conditions, we can order them with respect to how probable they are to test positive, finding the least probable positive case (most likely to read false negative) and denote it as P^+. Then, we may calculate the probability of the message given a uniform distribution of bytes at each element of the message using (7) and denote it as P^-. Finally, we calculate the threshold as the midpoint of the distance between the two: $\theta = \frac{P^+ + P^-}{2}$ which yields the threshold that maximizes the separation between classes at the current length. Finally, we need to determine θ as a function of message length. As the length of the message increases, we gain more information in calculating the likelihood that it was produced by a given cipher. With more information, the likelihoods begin to diverge and the margin between the two classes increases. Assuming the margins continue to diverge in this manner, we calculate the simple linear equation: $\theta = mL + \theta_0$ to represent θ as a function of L, where m is the slope of the line and θ_0 is the intercept.

D. DETECTION PHASE

Once the models have been trained, they can then be used to make predictions about our suspected ciphertexts. The following sections describe the detection algorithms for when the complete ciphertext is available and when we only have access to a partial ciphertext.

1) DETECTION OF STREAM CIPHERTEXTS WHEN A COMPLETE CIPHERTEXT IS AVAILABLE

First, we develop a solution to the scenario in which we have as much information as possible: the complete ciphertext c generated by E_k. We denote the likelihood that E was used to generate c as $P(E_k|c)$ which we defined to be proportional to $P(c|E_k)$ in (3). Thus, by calculating the independent probabilities of each byte of c given the stream cipher E_k, we can determine $P(E_k|c)$ as:

$$P(E_k|c) \propto \prod_{i=1}^{L} P(c_i|E_k). \tag{8}$$

The presence of a stream cipher may be detected using the discriminant function found in (2). Here, the likelihood that the given stream cipher was used to generate the ciphertext is evaluated against the threshold (θ) to determine whether we can classify c as a ciphertext generated by E_k or not. The steps taken to calculate (8) and appropriately classify c are provided in Algorithm 2.

Algorithm 2 Detection of a Complete Ciphertext

```
    Training model:
 1: Use the training model in Algorithm 1
    Calculate probability:
 2: $p = 0$
 3: $i = 0$
 4: for $x_i \in x$ do
 5:     $p_i = M[i][x_i]$
 6:     $p = p + \ln(p_i)$
 7:     $i++$
 8: end for
    Determine threshold
 9: $l = length(x)$
10: $\theta = \ln(1/256) \times l$
    Detection
11: if $p > \theta$ then
12:     return $\mathbf{E}_k$
13: else
14:     return $\neg\mathbf{E}_k$
15: end if
```

2) DETECTION OF STREAM CIPHERTEXTS WHEN A PARTIAL CIPHERTEXT IS AVAILABLE

We also extend our detection scheme to the scenario in which a ciphertext is only partially available. In the previous section, we described how our proposed scheme detects ciphertexts encrypted by a stream cipher when the positions of values in

Algorithm 3 Detection of a Partial Ciphertext

```
    Training model:
 1: Use the training model in Algorithm 1
    Detection:
 2: for $i = 1$ to $(L - M + 1)$ do
 3:     $p_i$=0
 4:     for $j = 1$ to $M$ do
 5:         $p_i = p_i + \ln P(x_{i+j-1} = s_j|\mathbf{RC4}_k)$
 6:     end for
 7:     $p$=argmax $p_i$
 8:     if $(p > \zeta)$ then
 9:         return $\mathbf{E}_k$
10:     end if
11: end for
12: return $\neg\mathbf{E}_k$
```

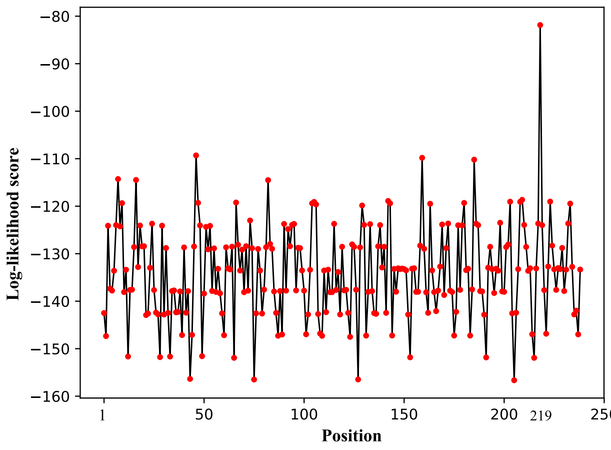


FIGURE 9. Log-likelihood of a partial packet across a network packet.

the ciphertext are known. However, in a real-world scenario, it is often difficult to know the exact position of a ciphertext in traffic and some network protocols may experience some amount of data loss. Since the patterns observed in the collected ciphertexts are position-dependent, our proposed scheme will not function properly when presented with only a partial message. Instead, we must define a method to find the most probable starting point of the partial message and calculate an accurate likelihood from that position.

Suppose that we analyze only a slice of a network packet, $s = \{s_i|1 \leq i \leq L_s\}$ and $s \subset c$, where the size of the partial packet is much shorter than that of the ciphertexts we have gathered previously, i.e., $L_s \ll L$. We must determine $P(E_k|s)$, which we can use to infer if $c = E(k, x)$. We begin by determining the most probable position of s in c. We define the likelihood function, $P(i|s, E_k)$, which is how likely the partial packet s is a subset of the ciphertext starting at i, where $1 \leq i \leq L$. The log-likelihood, $\ln P(i|s, E_k)$, can be computed by:

$$\ln P(i|s, E_k) = \sum_{j=1}^{M} \ln P(x_{i+j-1} = s_j|E_k), \tag{9}$$

where $P(x_{i+j-1} = s_j|E_k)$ represents the probability that the byte value s_j is observed in the $(i+j-1)$-th byte of the message x. Thus, $P(x_{i+j-1} = s_j|E_k)$ can be estimated by:

$$P(x_i|E_k) = \frac{c_i(x_i) + \alpha}{\sum_{m=0}^{255} c_i(m) + \alpha} \tag{10}$$

and the maximum likelihood function returns the most probable position where the statistical patterns of the partial packet match. The most probable position of the partial packet in c can be obtained by calculating:

$$i^* = \underset{i}{\text{argmax}} \ln P(i|s, E_k), \tag{11}$$

where $1 \leq i \leq L - L_s + 1$. An example of the log-likelihood of a partial packet is illustrated in Figure 9. Using (9), we describe how we might find the position of a partial packet in an original ciphertext. A partial packet of 18 bytes was extracted from an RC4 ciphertext, the likelihood scores were calculated in order to find the position of the partial packet in the original RC4 ciphertext.

As seen in Figure 9, the distribution of the log-likelihood shows that the highest score is approximately -82 at position 219, indicating the partial packet is most likely a subset of the original ciphertext located between bytes 219 and 236.

In order to determine whether or not the partial packet s is a subset of c, the discriminant function, $\mathcal{L}(s, k)$, is defined by a log-posterior probability, $\ln P(E_k|s)$, which can be estimated by the log-likelihood:

$$\begin{aligned} \mathcal{L}(s, k) &= \ln P(E_k|s) \\ &\propto \ln P(i^*|s, E_k) \\ &= \sum_{j=1}^{M} \ln P(x_{i^*+j-1} = s_j|E_k). \end{aligned} \tag{12}$$

Finally, ciphertexts can be detected by comparing with the threshold (ζ):

$$\mathcal{L}(s, k) = \sum_{j=1}^{M} \ln P(x_{i^*+j-1} = s_j|E_k) \underset{\neg E_k}{\overset{E_k}{\gtrless}} \zeta. \tag{13}$$

We empirically estimated the optimal threshold (ζ^*) and the minimum size (L_s) of the partial packet for ciphertext detection, comparing the distributions of log-likelihood scores of the two groups with synthetic data. The first group, denoted as G_C, includes only the log-likelihood scores computed in the correct position, and the second, denoted as G_I contains the scores in the other positions. A cut-off value that discriminates the two distributions is considered the optimal threshold. In this study, the optimal threshold is simply determined by the midpoint of the range, $[\max(G_I), \min(G_C)]$. Specifically, we generated 10,000 RC4 ciphertexts and selected a partial packet of length L_s in a random position of each ciphertext.

We considered various lengths of the partial packets: $L_s \in \{16, 18, 20, 26, 32, 34, 36, 40\}$. The empirically

TABLE 1. **The distributions of log-likelihood in G_C and G_I with various lengths.**

		G_C	G_I	Threshold value
16 bytes	Min	-74.17	-147.36	-73.39
	Max	-71.48	-72.61	
	Avg	-72.78	-119.65	
18 bytes	Min	-83.48	-165.78	-84.72
	Max	-80.41	-85.97	
	Avg	-81.88	-134.61	
20 bytes	Min	-92.57	-184.20	-96.34
	Max	-89.28	-100.11	
	Avg	-90.98	-149.55	
26 bytes	Min	-120.25	-239.46	-130.66
	Max	-116.57	-141.06	
	Avg	-118.28	-194.44	
32 bytes	Min	-147.72	-290.26	-160.70
	Max	-143.46	-173.68	
	Avg	-145.57	-239.37	
34 bytes	Min	-156.83	-304.27	-171.74
	Max	-152.51	-186.65	
	Avg	-154.66	-254.32	
36 bytes	Min	-165.92	-322.43	-183.07
	Max	-161.51	-200.22	
	Avg	-163.77	-269.31	
40 bytes	Min	-184.09	-354.63	-206.18
	Max	-179.57	-228.27	
	Avg	-181.97	-299.27	

optimal thresholds for the various lengths of partial packets are listed in Table 1. When the length of a partial packet is 13 bytes, the two distributions overlap between -72.61 and -71.48, which results in misclassification. However, the experiments show that the overlap decreases as the length of partial packets increase because longer partial packets provide more information about the statistical patterns of the original ciphertext. The distributions of the two groups separate from one another for lengths longer than 20 bytes, allowing partial packets of 20 bytes or more to be detected with 100% accuracy.

V. SIMULATION AND EVALUATION

We evaluated the proposed scheme using trained models for each of the six cipher methods found to exhibit the studied ciphertext patterns. Models were constructed from a synthetic dataset generated using a fixed key as described in Section IV-B which yields the optimal dataset. The distributions of M were calculated by sampling the set of 95 characteristic messages with replacement 5000 times to simulate training the model with 5000 messages. To generate testing data, we encrypted 200 messages at various lengths for each of the ciphers: 100 of which were generated under a fixed key, while the remaining 100 were generated using random keys. For each evaluation, the model is tested with 600 messages at each length: 100 of which were generated by the same cipher as the model and 500 from the remaining five ciphers studied. We consider two sets of message lengths for testing, the first of which is used to establish the lower detection threshold and contains messages of 1 byte to 15 bytes in length. The second set is used for the remaining tests and is comprised of messages with lengths of 13, 16, 18, 20, 22, 24, 26, 30, 34, and 40 bytes. The evaluation to determine the minimum detectable message length is tested with 9000 total messages (1500 positive cases, 7500 negative cases) and the remaining tests are conducted with 6000 total messages (1000 positive cases, 5000 negative cases). The results of each of the following evaluations are detailed in the remainder of this section:

- determining the minimum detectable ciphertext length,
- determining detection time and accuracy dependent upon ciphertext length,
- distinguishing fixed-key ciphertexts from those generated at random,
- distinguishing ciphertexts generated from different ciphers.

A. MINIMUM DETECTABLE MESSAGE LENGTH

The proposed detection scheme uses the characteristic gaps in value distributions of ciphertext bytes to determine the likelihood that the ciphertext was generated by a given encryption pair. Thus, any one byte alone is insufficient to make an accurate, justifiable classification. In order to determine the point at which our scheme can confidently detect E_k, we calculated the message classification accuracy with lengths below the minimum detectable message length as presented in Table 2.

TABLE 2. **Detection accuracy below minimum detectable length.**

Length	RC4	CHA	SAL	OFB	CTR	GCM
1 byte	0.900	0.685	0.653	0.887	0.602	0.603
2 bytes	0.995	0.822	0.817	0.997	0.847	0.997
3 bytes	0.993	0.998	0.925	0.995	0.930	0.997
4 bytes	0.995	0.993	0.995	0.990	0.993	0.993
5 bytes	0.992	0.993	0.993	0.988	0.997	0.993
6 bytes	0.998	1.000	0.968	1.000	0.967	0.997
7 bytes	1.000	1.000	0.967	1.000	0.960	1.000
8 bytes	0.998	1.000	0.990	0.998	0.980	1.000
9 bytes	1.000	0.998	1.000	1.000	1.000	1.000
10 bytes	1.000	0.998	1.000	0.998	1.000	1.000
11 bytes	1.000	0.998	0.998	1.000	1.000	1.000
12 bytes	1.000	0.998	1.000	1.000	1.000	1.000
13 bytes	1.000	1.000	1.000	1.000	1.000	1.000
14 bytes	1.000	1.000	1.000	1.000	1.000	1.000
15 bytes	1.000	1.000	1.000	1.000	1.000	1.000

At these shorter lengths, the scheme demonstrates a high True Positive Rate (TPR) due to the overlap of the model's distributions with all other possibilities. The value expressed at an arbitrary byte of a given ciphertext is likely to be observed in many other configurations of E_k. In typical malware detection, minimizing Type I error is not a priority because it does not affect the detection of actual malware. However, because our scheme is designed to detect a ciphertext generated by E_k from random, we must ensure that the False Positive Rate (FPR) is zero. Each model has a slightly different minimum length at which it achieves 100% accuracy, but for consistency, we will consider a message length of 13 bytes to be the safe lower limit for classifiable messages.

B. DETECTION TIME AND ACCURACY

Using 13 bytes as the minimum detection length, we tested each model for accuracy and average detection time using

TABLE 3. Minimum and maximum likelihoods calculated at various message lengths.

Length	θ	$P(E_k)$	RC4	CHA	SAL	OFB	CTR	GCM
13 bytes	-72.09	Max	-57.99	-57.68	-58.02	-57.75	-57.85	-57.76
		Min	-70.16	-69.79	-65.34	-70.53	-70.64	-65.20
16 bytes	-88.72	Max	-71.28	-71.72	-71.47	-71.53	-71.47	-71.25
		Min	-88.67	-83.97	-83.36	-79.54	-84.68	-79.48
18 bytes	-99.81	Max	-80.41	-80.27	-80.77	-80.07	-80.40	-81.04
		Min	-88.17	-92.50	-93.63	-93.12	-88.51	-93.60
20 bytes	-110.90	Max	-89.84	-89.77	-89.42	-89.50	-89.77	-89.23
		Min	-101.28	-107.26	-102.24	-97.35	-101.64	-102.59
22 bytes	-121.99	Max	-98.86	-98.42	-98.42	-98.90	-98.43	-98.75
		Min	-111.35	-112.58	-110.76	-111.98	-110.11	-111.39
24 bytes	-133.08	Max	-107.18	-107.49	-108.14	-107.00	-107.23	-106.95
		Min	-126.40	-119.73	-126.06	-125.86	-119.81	-126.98
26 bytes	-144.17	Max	-116.47	-116.64	-116.65	-116.81	-116.06	-116.84
		Min	-130.37	-129.22	-135.45	-129.33	-130.78	-129.79
30 bytes	-166.36	Max	-135.04	-134.23	-134.20	-134.68	-134.38	-134.01
		Min	-149.05	-150.48	-147.96	-152.23	-152.50	-148.45
34 bytes	-188.54	Max	-153.31	-152.23	-153.03	-152.31	-152.56	-152.50
		Min	-170.42	-168.14	-164.74	-166.92	-172.69	-175.16
40 bytes	-221.81	Max	-179.35	-179.16	-179.46	-178.78	-179.75	-180.05
		Min	-205.20	-195.98	-198.91	-199.33	-195.62	-199.87

TABLE 4. Average detection times in μsec.

Length	RC4	CHA	SAL	OFB	CTR	GCM
13 bytes	17.63 ± 0.12	17.74 ± 0.25	17.77 ± 0.29	17.59 ± 0.21	17.81 ± 0.23	17.86 ± 0.33
16 bytes	22.36 ± 0.64	22.50 ± 0.85	22.92 ± 2.7	22.94 ± 2.3	24.38 ± 5.5	24.98 ± 6.5
18 bytes	24.49 ± 0.65	25.24 ± 2.4	24.87 ± 2.3	25.03 ± 3.2	25.83 ± 5.7	24.86 ± 3.1
20 bytes	27.05 ± 0.53	27.28 ± 1.5	26.66 ± 0.75	26.63 ± 1.7	27.20 ± 2.6	27.27 ± 2.7
22 bytes	29.13 ± 0.92	29.15 ± 1.5	29.32 ± 1.0	30.38 ± 5.5	29.86 ± 3.5	29.15 ± 1.1
24 bytes	31.44 ± 0.64	31.36 ± 1.3	31.70 ± 0.61	31.11 ± 2.1	31.40 ± 3.4	30.94 ± 1.1
26 bytes	33.83 ± 0.74	33.84 ± 3.3	34.52 ± 5.5	33.84 ± 3.3	34.10 ± 4.5	33.81 ± 3.7
30 bytes	37.83 ± 0.89	39.38 ± 6.0	38.34 ± 1.3	38.09 ± 1.9	38.39 ± 1.4	38.61 ± 3.4
34 bytes	42.39 ± 0.94	42.76 ± 2.4	42.48 ± 0.94	43.24 ± 1.6	43.28 ± 5.2	42.67 ± 3.7
40 bytes	50.10 ± 6.2	49.15 ± 3.4	49.04 ± 2.0	49.23 ± 2.6	48.84 ± 1.3	50.88 ± 8.9

the 100 messages generated by E_k at each message length, as seen in Table 3. Each model maintained 100% accuracy for all messages at the tested lengths and we have provided the minimum and maximum calculated likelihoods calculated for each. The minimum likelihood is important as it represents the message in the dataset which is closest to presenting Type II error. Messages shorter than 13 bytes may still be classified accurately by chance, but when the minimum calculated likelihood is greater than the threshold, we can confidently say that our model has made an accurate classification because the classes are separable. We calculate the average minimum likelihood for all models at lengths from 1 to 40 bytes and plot them in Figure 10 to demonstrate how the model's confidence increases with message length.

The times taken to complete detection at each length are provided in Table 4. Average times are listed in microseconds with two standard deviations. Because our algorithm uses each byte of the message to estimate likelihood, time increases proportionally with the length of the message.

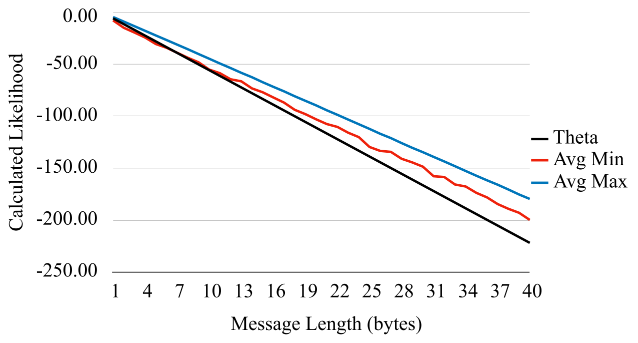


FIGURE 10. Average likelihoods calculated at various message lengths.

C. DISTINGUISHABILITY FROM RANDOM-KEY CIPHERTEXTS

Next, we show that a model can perfectly distinguish ciphertexts generated by the same encryption scheme under different keys. Table 5 displays the calculations of the maximum likelihood amongst the random-key ciphertexts. The model is able to detect these messages and maintain perfect accuracy. The maximum likelihood is the message that is closest to present Type I error. The model easily detects random-key cases as evidenced by the significant differences between the calculated maximum likelihoods and θ at each level.

D. DISTINGUISHABILITY FROM OTHER MODELS

Finally, we show that the models are distinguishable from one another. Using the fixed-key training set, each model is tested with 500 ciphertexts comprised of the 100 messages encrypted by the other five ciphers at each testing

TABLE 5. Distinguishability of ciphertexts generated under a random key by comparing the model's most-probable false positive prediction to theta.

Length	Theta	RC4	CHA	SAL	OFB	CTR	GCM
13 bytes	-72.09	-85.22	-81.14	-86.28	-80.26	-79.66	-80.51
16 bytes	-88.72	-99.96	-94.78	-94.53	-100.00	-105.84	-100.75
18 bytes	-99.81	-119.73	-119.74	-108.67	-114.55	-114.83	-109.39
20 bytes	-110.90	-128.64	-134.39	-129.29	-124.62	-124.43	-128.85
22 bytes	-121.99	-138.12	-143.47	-142.94	-144.80	-143.94	-148.42
24 bytes	-133.08	-153.58	-159.14	-158.42	-153.07	-158.15	-158.08
26 bytes	-144.17	-178.81	-178.32	-172.30	-173.18	-178.40	-177.98
30 bytes	-166.36	-185.81	-207.70	-201.25	-202.96	-196.81	-207.22
34 bytes	-188.54	-231.27	-221.27	-235.62	-225.73	-211.03	-228.77
40 bytes	-221.81	-279.83	-265.74	-284.18	-284.48	-284.14	-285.01

TABLE 6. Distinguishability between models for ciphers generated under the same key by comparing the relative most-probable false positive to theta.

Length	Theta	RC4	CHA	SAL	OFB	CTR	GCM
13 bytes	-72.09	-80.58	-90.75	-81.13	-80.61	-80.22	-86.25
16 bytes	-88.72	-104.76	-104.85	-105.16	-104.91	-105.25	-115.57
18 bytes	-99.81	-119.17	-119.53	-119.44	-119.86	-119.41	-125.40
20 bytes	-110.90	-134.04	-134.09	-134.44	-134.89	-128.95	-140.35
22 bytes	-121.99	-144.21	-148.72	-145.09	-154.41	-144.51	-159.25
24 bytes	-133.08	-158.27	-157.08	-157.86	-163.67	-158.21	-168.64
26 bytes	-144.17	-172.19	-178.45	-172.66	-183.04	-172.55	-188.56
30 bytes	-166.36	-212.87	-206.54	-207.56	-212.27	-217.81	-217.52
34 bytes	-188.54	-230.89	-242.89	-236.47	-242.84	-236.78	-248.59
40 bytes	-221.81	-277.54	-285.91	-285.58	-284.37	-274.76	-290.79

length. In this scenario, each model has implemented the same encryption key, so we can hold k constant as we change E. As shown in Table 6, the maximum likelihood amongst the 500 tests is provided to represent the message most likely to be classified as a ciphertext generated by the current cipher, i.e., presenting Type II error. The results of the evaluation indicate that the models are distinct for each cipher and key, preventing the misclassification of a ciphertext generated by one of the other five models using the same key.

VI. EXPERIMENTAL RESULTS WITH DarkComet

We have studied and presented a statistical weakness found in the ciphertexts generated by stream ciphers and proposed techniques that may be used to detect them in different scenarios. This section is dedicated to detailing how our scheme can be used to improve the detection of encrypted malware through a real-world experiment on DarkComet.

For ease of implementation, encrypted malware tends to use a pre-stored key to mitigate the difficulty of key sharing once it has been deployed. In this case, the stored key can be obtained via checking source code or through reverse engineering, allowing us to generate the optimal dataset using the known key. Conversely, other types of malware might exploit a technique such as code obfuscation which renders key acquisition almost impossible. Our training scenarios address both of these potential conditions. By providing a method to detect the use of a stream cipher, we can further expand the discrimination function used to identify potential malware traffic.

The RC4 stream cipher found wide use in secure technologies such as TLS, WEP, and WPA, among other applications. Though its use in industry has since been replaced by other encryption methods following the discovery of a multitude of vulnerabilities, RC4 is still used by malware programmers due to its ease of implementation. DarkComet, for example, is a RAT that gained popularity in the early 2010s which utilizes RC4 encryption under a fixed key to camouflage its activity. In the following subsections, we describe the process of how we can use our proposed algorithm to detect malware in a situation where it is actively sending information between a victim and an attacking machine using DarkComet.

A. DarkComet TESTING ENVIRONMENT

DarkComet 5.3.1 was installed in a secure testing environment configured on an offline machine. To study the malware traffic, we set-up two virtual machines (VM) with one acting as an attacker and the other as a victim. Both virtual machines were deployed using VMware Player running versions of Windows 10.

DarkComet infects a victim machine by deploying a *stub*. When opened on the machine, the stub drops the payload and initiates the connection with the attacker. To simulate this event, the stub was copied to the victim VM via USB and opened. Once connected, DarkComet begins to send its encrypted messages via Transmission Control Protocol (TCP). We used Wireshark on the attacking VM to capture communications between the two machines. Because we configured the testing environment offline, filtering the packets generated by DarkComet was trivial.

B. COLLECTION OF DarkComet PACKETS

Though it is now considered a malicious trojan, DarkComet was originally designed more generally as a remote administration tool and provides functions used by other similar applications. These tools include, but are not limited to: screen capture, access to the secondary machine's shell, and

TABLE 7. Classification results of detecting DarkComet using the proposed scheme.

Length	Theta	DarkComet Min	Benign Max	TPR	FPR	Accuracy
1 byte	-5.55	-4.90	-4.40	1.000	0.228	0.886
2 bytes	-11.09	-9.62	-8.86	1.000	0.098	0.951
3 bytes	-16.64	-14.27	-13.36	1.000	0.048	0.976
4 bytes	-22.18	-18.75	-17.82	1.000	0.012	0.994
6 bytes	-33.27	-28.02	-27.58	1.000	0.032	0.984
7 bytes	-38.82	-32.77	-37.53	1.000	0.010	0.995
8 bytes	-44.36	-37.31	-41.92	1.000	0.002	0.999
9 bytes	-49.91	-41.83	-46.51	1.000	0.002	0.999
10 bytes	-55.45	-46.48	-56.34	1.000	0.000	1.000
11 bytes	-61.00	-51.10	-61.07	1.000	0.000	1.000
12 bytes	-66.54	-55.64	-65.65	1.000	0.002	0.999
13 bytes	-72.09	-60.33	-75.65	1.000	0.000	1.000
14 bytes	-77.63	-64.87	-85.65	1.000	0.000	1.000
15 bytes	-83.18	-69.41	-90.69	1.000	0.000	1.000
16 bytes	-88.72	-73.98	-99.70	1.000	0.000	1.000

key-logging. It also has a collection of "Fun Functions" which provide features such as remote chat between the two machines, a text-to-speech dictator, a piano which plays sounds on the secondary machine, and options to show and hide items on the GUI (clock, taskbar, desktop) amongst others. We chose to primarily use these functions to generate data as they are the most easily repeatable.

To generate communication data, we used the functions within the DarkComet malware to send TCP packets between the attacking and victim VM machines, capturing these packets via Wireshark. Studying these messages, we found that a DarkComet TCP packet consists of two parts: a header that pertains to the function called and some appended text regarding what the function will execute. For example, sending the message "Hello, World!" via the remote chat function would produce a TCP packet with a payload containing "CHATOUTHello, World!" encrypted via RC4. This payload is what we used as the data for our detection methods, discarding other information found in the header of the packet.

To extract the payload, we exported the collected Wireshark data to a JSON file. Then, using a Python script, we filtered out unwanted information so that we were left with only TCP communications which contain a payload. The data is output as hexadecimal values representing each byte, so we wrote a method to convert the data to the appropriate ASCII characters and then finally to the decimal representation of each character.

C. DarkComet DETECTION USING THE DISCRIMINATION FUNCTION

As described in Section II, we know that DarkComet employs RC4 and uses a fixed key under which all messages are encrypted. Conveniently, we know the standard key for the version used in our simulation to be #KCMDDC51#-890 [38]. This allows us to generate models synthetically or via captured packets as proposed in Section IV-B. As we have demonstrated the scheme's efficacy using synthetic data via a known key in the previous section, we will simulate building the DarkComet model empirically.

We begin by collecting 10,000 malicious packets using the DarkComet fun functions to generate traffic between the two machines. Then, to complete our dataset, we obtained benign samples by capturing packets from casual web surfing on an uninfected machine. A testing set is generated using 500 packets from the web surfing data and another 500 from the DarkComet traffic, labeling each *benign* or *malignant* accordingly. The remaining DarkComet packets are allocated for training the detection model.

Each packet generated through DarkComet is prepended with a plaintext ID pertaining to the function used to generate the packet. Because this further reduces the variance in the values expressed over the first n bytes, it can cause our model to overfit itself to a dataset containing messages from an imbalanced collection of message sources. We demonstrate how we may use Algorithm 3 to detect a message where the first n bytes are ignored.

Results of testing the model with messages sliced over various lengths are provided in Table 7. Here, a perfect TPR is maintained over the selected message lengths. This is due to the positive class being generated from the same source as the training data and under the same conditions, as introduced in Section V. Since both training data and malware packets were encrypted using RC4 and the same key, they must share a unique distribution. There is no DarkComet packet that does not fall under the distribution, and therefore, the discrimination function retains the perfect TPR regardless of the message length. However, it is not true that all benign packets are generated from the distribution, and thus, the discrimination function identifies a non-negligible number of false positives when the length of the input is insufficient. Due to these reasons, we use the FPR to indicate the model's efficacy. Furthermore, the FPR also tells us how many bytes are necessary for the model to make accurate detection of a malicious packet at the point which there are no more false positives. Table 7 confirms our assertion that 13 bytes are the minimum detectable message length because we eliminate false positives at this level, and the two classes become linearly separable.

VII. CONCLUSION

In this paper, we have expanded upon a novel approach to detect the use of a stream cipher, showing that the findings of [1] in regards to RC4 can be extended to other stream ciphers or structurally-similar block cipher modes of operation. A weakness produced by the properties of stream ciphers and character encoding allowed us to create a unique fingerprint for a cipher and key. The proposed machine learning scheme uses these fingerprints to discriminate ciphertexts generated by the cipher and key from random. We evaluated the models, demonstrating the distinguishability ciphertexts generated by a stream cipher under different keys and also from ciphertexts generated by other ciphers using the same key. Experiments with the DarkComet RAT indicate that the trained model also accurately classifies packets generated by malware which uses a stream cipher to encrypt its communications. The model can detect such ciphertexts with 100% accuracy for messages longer than 13 bytes in approximately 17 μsec.

There is no unique solution to detect all types of malware. Nonetheless, our proposed solution effectively detects a specific type of malware that encrypts its traffic via a stream cipher and a fixed key through use of a discrimination function. Our discovery and proposed scheme will aid networking tools by providing a fast and accurate method to detect encrypted malware which will ultimately produce more robust network security solutions. In our future works, we intend to expand our experiments through access to other malware testing environments and extend our scheme to other encoding standards such as Unicode variants.

REFERENCES

[1] J. Son, E. Ko, U. B. Boyanapalli, D. Kim, Y. Kim, and M. Kang, "Fast and accurate machine learning-based malware detection via RC4 ciphertext analysis," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Feb. 2019, pp. 159–163.

[2] R. Verdult. (Aug. 2015). *Introduction to Cryptanalysis: Attacking Stream Ciphers*. [Online]. Available: http://www.cs.ru.nl/~rverdult/Introduction_to_Cryptanalysis-Attacking_Stream_Ciphers.pdf

[3] D. E. Denning, "The many-time pad: Theme and variations," in *Proc. IEEE Symp. Secur. Privacy*, Apr. 1983, pp. 23–30.

[4] A. Grosul and D. Wallach, "A related-key cryptanalysis of RC4," Dept. Comput. Sci., Rice Univ., Houston, TX, USA, Tech. Rep. TR00-358, 2000. [Online]. Available: https://scholarship.rice.edu/handle/1911/96275

[5] I. Mantin and A. Shamir, "A practical attack on broadcast RC4," in *Proc. Int. Workshop Fast Softw. Encryption*. Berlin, Germany: Springer-Verlag, 2001, pp. 152–164.

[6] C. E. Shannon, "Communication theory of secrecy systems," *Bell Syst. Tech. J.*, vol. 28, no. 4, pp. 656–715, Oct. 1949.

[7] S. W. Brim and B. E. Carpenter. (Feb. 2002). *Middleboxes: Taxonomy and Issues*. [Online]. Available: https://rfc-editor.org/rfc/rfc3234.txt

[8] B. Farinholt, M. Rezaeirad, P. Pearce, H. Dharmdasani, H. Yin, S. L. Blond, D. McCoy, and K. Levchenko, "To catch a ratter: Monitoring the behavior of amateur DarkComet RAT operators in the wild," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 770–787.

[9] Quequero. (2012). *DarkComet Analysis-Understanding the Trojan Used in Syrian Uprising*. [Online]. Available: https://resources.infosecinstitute.com/darkcomet-analysis-syria/

[10] Y. Dodis and J. Spencer, "On the (non) universality of the one-time pad," in *Proc. 43rd Annu. IEEE Symp. Found. Comput. Sci.*, Nov. 2002, pp. 376–385.

[11] F. Armknecht, "Algebraic attacks on stream ciphers," in *Proc. ECCOMAS-Eur. Congr. Comput. Methods Appl. Sci. Eng.*, Nov. 2004.

[12] P. McLaren, G. Russell, W. J. Buchanan, and Z. Tan, "Decrypting live SSH traffic in virtual environments," *Digit. Invest.*, vol. 29, pp. 109–117, Jun. 2019.

[13] B. Jungk and S. Bhasin, "Don't fall into a trap: Physical side-channel analysis of ChaCha20-Poly1305," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1110–1115.

[14] P. Jindal and B. Singh, "A survey on RC4 stream cipher," *Int. J. Comput. Netw. Inf. Secur.*, vol. 7, no. 7, pp. 37–45, Jun. 2015.

[15] A. Roos, "A class of weak keys in the RC4 stream cipher," Vironix Softw. Lab., Westville, South Africa, Tech. Rep. 1, 1995.

[16] S. R. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the key scheduling algorithm of rc4," in *Proc. 8th Annu. Int. Workshop Sel. Areas Cryptogr.* Berlin, Germany: Springer-Verlag, 2001, pp. 1–24.

[17] J. Chen and A. Miyaji, "A new practical key recovery attack on the stream cipher RC4 under related-key model," in *Information Security and Cryptology* (Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6584. Berlin, Germany: Springer, Oct. 2010, pp. 62–76.

[18] S. S. Gupta, S. Maitra, G. Paul, and S. Sarkar, "(Non-)random sequences from (non-)random permutations—Analysis of RC4 stream cipher," *J. Cryptol.*, vol. 27, no. 1, pp. 67–108, Jan. 2014.

[19] M. Vanhoef and F. Piessens, "All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS," in *Proc. USENIX Secur. Symp.*, 2015, pp. 97–112.

[20] L. Martignoni, M. Christodorescu, and S. Jha, "OmniUnpack: Fast, generic, and safe unpacking of malware," in *Proc. 23rd Annu. Comput. Secur. Appl. Conf. (ACSAC)*, Dec. 2007, pp. 431–440.

[21] R. Zhao, D. Gu, J. Li, and Y. Zhang, "Automatic detection and analysis of encrypted messages in malware," in *Information Security and Cryptology* (Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 8567. Cham, Switzerland: Springer, 2014, pp. 101–117.

[22] B. Anderson, S. Paul, and D. McGrew, "Deciphering malware's use of TLS (without decryption)," *J. Comput. Virol. Hacking Techn.*, vol. 14, no. 3, pp. 195–211, Aug. 2018.

[23] B. Anderson and D. McGrew, "Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. New York, NY, USA: Association for Computing Machinery, Aug. 2017, pp. 1723–1732, doi: 10.1145/3097983.3098163.

[24] P. Prasse, L. Machlica, T. Pevny, J. Havelka, and T. Scheffer, "Malware detection by analysing network traffic with neural networks," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, San Jose, CA, USA, May 2017, pp. 205–210.

[25] R. Z. J. Liu, Z. Tian, and L. Liu, "A distance-based method for building an encrypted malware traffic identification framework," *IEEE Access*, no. 7, pp. 100014–100028, 2019.

[26] J. Kohout and T. Pevny, "Unsupervised detection of malware in persistent Web traffic," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 1757–1761.

[27] H. Sultan, A. Khalique, S. I. Alam, and S. Tanweer, "A survey on ransomeware: Evolution, growth, and impact," *Int. J. Adv. Res. Comput. Sci.*, vol. 9, no. 2, pp. 802–810, 2018, doi: 10.26483/ijarcs.v9i2.5858.

[28] J. P. Tailor and A. D. Patel, "A comprehensive survey: Ransomware attacks prevention, monitoring and damage control," *Int. J. Res. Sci. Innov.*, vol. 4, pp. 116–121, Jun. 2017. [Online]. Available: https://www.rsisinternational.org

[29] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "CryptoLock (and drop It): Stopping ransomware attacks on user data," in *Proc. IEEE 36th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2016, pp. 303–312.

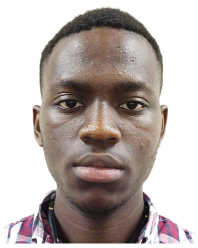
[30] N. Anderson, "How the FBI found miss teen USA's Webcam spy," Ars Technica, New York, NY, USA, Tech. Rep., 2013. [Online]. Available: https://arstechnica.com/tech-policy/2013/09/miss-teen-usas-webcam-spy-called-himself-cutefuzzypuppy/

[31] W. R. Marczak, J. Scott-Railton, M. Marquis-Boire, and V. Paxson, "When governments hack opponents: A look at actors and technology," in *Proc. 23rd USENIX Secur. Symp. (USENIX Security)*. San Diego, CA, USA: USENIX Association, Aug. 2014, pp. 511–525. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/marczak

[32] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang, "On the analysis of the zeus botnet crimeware toolkit," in *Proc. 8th Int. Conf. Privacy, Secur. Trust*, Aug. 2010, pp. 31–38.
[33] C. Park, H. Park, and K. Kim, "Realtime C&C zeus packet detection based on RC4 decryption of packet length field," *Adv. Sci. Technol. Lett.*, vol. 64, no. 14, pp. 55–59, 2014.
[34] A. A. Awad, S. G. Sayed, and S. A. Salem, "A host-based framework for RAT bots detection," in *Proc. Int. Conf. Comput. Appl. (ICCA)*, Sep. 2017, pp. 336–342.
[35] V. C. Craciun, A. Mogage, and E. Simion, "Trends in design of ransomware viruses," in *Innovative Security Solutions for Information Technology and Communications* (Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 11359. Cham, Switzerland: Springer, 2019, pp. 259–272.
[36] P. McLaren, W. J. Buchanan, G. Russell, and Z. Tan, "Discovering encrypted bot and ransomware payloads through memory inspection without *a priori* knowledge," 2019, *arXiv:1907.11954*. [Online]. Available: http://arxiv.org/abs/1907.11954
[37] A. Adamov, A. Carlsson, and T. Surmacz, "An analysis of LockerGoga ransomware," in *Proc. IEEE East-West Design Test Symp. (EWDTS)*, Sep. 2019, pp. 1–5.
[38] L. Aylward, "Malware analysis-dark comet rat," Context/Accenture Secur., Essen, Germany, Tech. Rep. 1, 2011. [Online]. Available: https://www.contextis.com/en/blog/malware-analysis-dark-comet-rat

WILLIAM STONE received the B.S.E. degree in biomedical engineering from Mercer University, Macon, GA, USA, in 2013. He is currently pursuing the M.S. degree in computer science from Kennesaw State University, Marietta, GA, USA. He is working under the supervision of Dr. J. Son as a Research Assistant with Kennesaw State University. His current research interests include the domain of artificial intelligence and machine learning with a focus on, but not limited to, the implementation of data science tools to solve cybersecurity problems. His other interests include data privacy, social networking, game theory, and econometrics.

DAEYOUNG KIM received the B.S. degree in electronic and computer engineering and the M.S. degree in computer science and engineering from Hanyang University, South Korea, in 2005 and 2007, respectively, and the Ph.D. degree in computer science from Rutgers University, in 2019. He is currently a limited-term Assistant Professor of Computer Science with Kennesaw State University. His research interests include computer security and privacy and mobile computing.



VICTOR YOUDOM KEMMOE received the B.Sc. degree in computer science from Kennesaw State University, GA, USA, in 2018, where he is currently pursuing the M.Sc. degree in computer science. He is also a Graduate Research Assistant with the Information and Intelligent Security Laboratory, Kennesaw State University. His research interests include applied cryptography, cybersecurity, blockchain and smart contracts, machine learning in cybersecurity, and quantum computing.

MINGON KANG received the B.E. degree in computer engineering from Hanyang University, Ansan, South Korea, and the M.S. and Ph.D. degrees in computer science from The University of Texas at Arlington, Arlington, TX, USA, in 2010 and 2015, respectively. He is currently an Assistant Professor with the Department of Computer Science, University of Nevada Las Vegas. He is the author of over 55 journal and conference papers. His research interests include bioinformatics, machine learning, data mining, and big data analytics.

JUNGGAB SON (Member, IEEE) received the B.S.E. degree in computer science and engineering from Hanyang University, Ansan, South Korea, in 2009, and the Ph.D. degree in computer science and engineering from Hanyang University, Seoul, South Korea, in 2014.

From 2014 to 2016, he was a Postdoctoral Research Associate with the Department of Mathematics and Physics, North Carolina Central University. From 2016 to 2018, he was a Research Fellow and a limited-term Assistant Professor with the Department of Computer Science, Kennesaw State University, where he has been an Assistant Professor with the Department of Computer Science, since 2018. His research interests include applied cryptography, cybersecurity, privacy protection, blockchain and smart contract, and datascience in cybersecurity. He is an Associate Editor of IEEE Access.