Summer 7-2020

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

William T. Stone
Kennesaw State University

Junggab Son

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

A Thesis Presented to

The Faculty of the Computer Science Department

by

William T. Stone III

In Partial Fulfillment

of Requirements for the Degree

Master of Science, Computer Science

Kennesaw State University

July 2020

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

Approved:

DocuSigned by:
Junggab Son
A197BD0D60714A7...
July 23, 2020

Dr. Junggab Son – Advisor

DocuSigned by:
Coskun Cetinkaya
028A87E4965A4EE...
July 23, 2020

Dr. Coskun Cetinkaya – Department Chair

DocuSigned by:
Jon Preston
FD9E8EC07752427...
July 23, 2020

Dr. John Preston – Dean

In presenting this thesis as a partial fulfillment of the requirements for an advanced degree from Kennesaw State University, I agree that the university library shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from, or to publish, this thesis may be granted by the professor under whose direction it was written, or, in his absence, by the dean of the appropriate school when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from or publication of, this thesis which involves potential financial gain will not be allowed without written permission.

William T. Stone III

Notice To Borrowers

Unpublished theses deposited in the Library of Kennesaw State University must be used only in accordance with the stipulations prescribed by the author in the preceding statement.

The author of this thesis is:

William T. Stone III

1100 South Marietta Parkway

Marietta, GA, USA 30060

The director of this thesis is:

Junggab Son

1100 South Marietta Parkway

Marietta, GA, USA 30060

Users of this thesis not regularly enrolled as students at Kennesaw State University are required to attest acceptance of the preceding stipulations by signing below. Libraries borrowing this thesis for the use of their patrons are required to see that each user records here the information requested.

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

An Abstract of

A Thesis Presented to

The Faculty of the Computer Science Department

by

William T. Stone III

B.S. Biomedical Engineering, Mercer University, 2013

In Partial Fulfillment

of Requirements for the Degree

Master of Science, Computer Science

Kennesaw State University

July 2020

Abstract

Encryption key use is a critical component to the security of a stream cipher: because many implementations simply consist of a key scheduling algorithm and logical exclusive or (XOR), an attacker can completely break the cipher by XORing two ciphertexts encrypted under the same key, revealing the original plaintexts and the key itself. The research presented in this paper reinterprets this phenomenon, using repeated-key cryptanalysis for stream cipher identification. It has been found that a stream cipher executed under a fixed key generates patterns in each character of the ciphertexts it produces and that these patterns can be used to create a fingerprint which is distinct to a certain stream cipher and encryption key pair. A discrimination function, trained on this fingerprint, optimally separates ciphertexts generated through an enciphering pair from those which are generated by any other means. The patterns were observed in the Rivest Cipher 4 (RC4), ChaCha20-Poly1305, and Salsa20 stream ciphers as well as block cipher modes of operation that perform similarly to stream ciphers, such as: Counter (CTR), Galois/Counter (GCM), and Output feedback (OFB) modes. The discriminatory scheme proposed in this study perfectly detects ciphertexts of a fixed-key stream cipher with or without explicit knowledge of the key which may be utilized to detect a specific type of malware that exploits a stream cipher with a stored key to encrypt or obfuscate its activity. Finally, using real-world example of this type of malware, it is shown that the scheme is capable of detecting packets sent by the DarkComet remote access trojan, which utilizes RC4, with 100% accuracy in about 36 μs, providing a fast and highly accurate tool to aid in detecting malware using encryption.

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

A Thesis Presented to

The Faculty of the Computer Science Department

by

William T. Stone III

In Partial Fulfillment

of Requirements for the Degree

Master of Science, Computer Science

Advisor: Junggab Son

Kennesaw State University

July 2020

Acknowledgment

To my advisor, Dr. Junggab Son who was willing to give me a chance: thank you for your guidance, sharing your knowledge with me, and your tremendous patience. Your mentorship has helped open my future to new possibilities. Though I have always been a bit slow, the opportunity you provided me has helped me better focus on my studies and without it, I'm unsure if I would have graduated with a true sense of accomplishment. Thank you to Dr. Daeyoung Kim for your friendship, contribution to the project, and as an added benefit, your help maturing my coding skills. Thank you to the rest of my friends and fellow members of the Information and Intelligent Security Lab at KSU who I was able to share many lunches and laughs with. And thank you to the faculty at KSU for your time and helping to rekindle a love of learning that I lost long ago and to The Graduate College who helped ease the financial burden of higher education.

Thank you to everyone who has helped me be who I am today. In particular, I would like to thank my Mom, Dad, and my sister and best-friend BeAna who have always supported me and believed in me at times when I didn't believe in myself. To my late grandfather who paved the way, my Nana and uncle Steve who I could have not made it this far without, my uncle DK who has fueled my love of technology and learning. To my uncle Peter and rest of my family who I can't list because there are too many to thank: thank you for your love and for teaching me the power of kindness and the importance of family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The stream cipher is a method of symmetric encryption modeled after the One-time Pad. It is susceptible to well-documented attacks such as *related-key*, *divide-and-conquer*, *malleability*, among others (Verdult, 2015). One critical vulnerability of stream ciphers lies in the reuse of the encryption key. A Many-time Pad can be broken easily by an adversary who obtains two messages that were encrypted under the same key: by performing a simple exclusive-or (XOR) of the two messages, he can uncover the respective original messages and finally, the key itself (Denning, 1983). This vulnerability is used for many cryptanalytic studies focused on attacking the cipher and uncovering sensitive information, however, this study will use repeated-key encryption for the purpose of ciphertext identification.

The process of stream cipher encryption is performed via a direct mapping of one plaintext character to one ciphertext character, repeated encryption under a fixed key will produce identical mappings with each iteration. Using this property which is inherent to stream ciphers which employ an exclusive-or operation along with the ASCII encoding scheme which is used to define the characters in the language of the encrypted messages, a deterministic mapping can be obtained which represents every possible value at each index across all possible ciphertexts generated by the stream cipher and encryption key pair. Together, these deterministic mappings can be used to

create a model which is distinct for a given encryption pair. The distinct model serves as a type of fingerprint which can be used to identify ciphertexts generated by the pair.

Presented in this work is a ciphertext discrimination function that, when trained on a given fingerprint, is capable of identifying *all* ciphertexts generated by the cipher and key pair which produced the fingerprint. The discrimination function is formed using Bayesian statistic methods to determine the likelihood that a given message is a ciphertext generated by the same pair as the fingerprint, accomplished by taking the probability of the value at each byte of the tested message in the positional distributions of the fingerprint pattern. Because the key is fixed during this analysis, the discrimination function can be determined with or without explicit knowledge of the encryption key. Through simulation, it is shown that this detection scheme is effective at classifying ciphertexts from the following stream ciphers: Alleged Rivest Cipher 4 (RC4), ChaCha20-Poly1305, and Salsa20. Furthermore, it also works well with a subset of block cipher modes of operation that operate as a stream cipher, such as Counter (CTR), Galois/Counter (GCM), and output feedback (OFB) modes.

Though fixed key encryption is known to be insecure and is avoided in practice, one potential application for this scheme is use as a tool to detect *encrypted malware*. Encrypted malware is a type of malicious software which utilizes encryption to hide itself or obfuscate its malicious activities from detection. Malware authors may even utilize encryption techniques supported in secure networking protocols, rendering traditional antivirus or malware detection tools, such as network monitoring, under-equipped against these forms of invasion. Messages passed by encrypted malware will appear as regular, benign communication to an unintelligent monitor. Among the most popular implementations of encryption in malware is RC4, which was used in the development of Zeus, Citadel, and Dridex, encompassing a variety of malware genres such as remote access trojans (RAT), ransomware, and botnets (Binsalleeh et al., 2010; Milletary, 2012; Adamov, Carlsson, & Surmacz, 2019). However, other stream ciphers have found use in malicious applications as well. For instance, Salsa20 was used in the development of the original Petya malware family, Advanced

Encryption Standard (AES) with GCM mode used to encrypt files for the Tycoon ransomware, and AES with Counter (CTR) mode in the Locky ransomware application (Protection, 2020; Sinitsyn, 2016).

Decryption of each message is not a feasible solution for the monitoring tool as this would decrease the security for and integrity of truly benign messages shared over the network, is both time and computationally expensive, and impossible in cases when the encryption parameters are not known. However, in many encrypted malware applications the author of the software will use a pre-stored key in order to mitigate the difficulty of sharing the key once the the malicious payload is deployed. The ciphertext discrimination function proposed in this paper offers a promising solution for the detection of encrypted malware packets as it enables a monitoring tool to classify encrypted packets from malware without necessitating decryption, providing the notable benefit of fast and accurate packet inspection. While there is no generic solution that is capable of detecting all malware, the proposed scheme can effectively and efficiently detect a specific type of malware which transfers packets encrypted using a stream cipher under a fixed key.

Experiments with real-world malware, the DarkComet RAT which encrypts its traffic using RC4, demonstrate the efficiency and effectiveness of the proposed scheme with empirical data. The proposed scheme achieves accuracy of 100% when the key is known, taking about 350 milliseconds (*ms*) for model creation and 36 microseconds (μs) for packet detection on an Intel Core (TM) i7-8700 processor. The scheme also achieves 100% accuracy when the key is not known, training the detection model and classifying messages in approximately 11 seconds (*sec*) and 32 μs, respectively.

The remainder of this paper is organized as follows: first, a brief literature review of related studies is provided in Chapter 2, Chapter 3 details the stream cipher mechanism and the source of the statistical weakness used for the detection scheme, in Chapter 4 the specifications of the discrimination function that can identify messages generated by the stream cipher and encryption key pair are described, Chapter 5 presents a real-world simulation of the proposed scheme and its results

using the remote access trojan DarkComet. Finally, concluding remarks and a brief discussion of future works are presented in Chapter 6.

Chapter 2

Literature Review

Though the proposed scheme is not suitable for detecting all malware, it may be useful in detecting a specific type. Presented below are works dedicated to the study of the type of malware this scheme may be effective against.

2.1 Notable Weaknesses of and Attacks on Stream Ciphers

In (Armknecht, 2004), F. Armknecht presented 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 (McLaren, Russell, Buchanan, & Tan, 2019), the authors exposed a vulnerability of OpenSSH and OpenSSL that allows for the discovery of cryptographic artefacts 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 (Jungk & Bhasin, 2017).

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 (Jindal & Singh, 2015). 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) (Mantin & Shamir, 2001), the initial byte generated by the Key Scheduling Algorithm is highly related to a few bytes from the key (weak keys) discovered by Roos in 1995 (Roos, 1995), the Fluhrer, Mantin, and Shamir study which showed that only a few keys may determine the output state and many output bits with significant probability (weak keys) (Fluhrer, Mantin, & Shamir, 2001). A practical key recovery attack is described in (Chen & Miyaji, 2011). 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 (Sen Gupta, Maitra, Paul, & Sarkar, 2014) 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 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 (Vanhoef & Piessens, 2015). The authors break WPA-TKIP by using biases which they detect empirically through statistical analysis, allowing them to uncover the TKIP MIC key.

2.2 Use of Encryption in Malware

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 *OmniUnpack* 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 (Martignoni, Christodorescu, & Jha, 2007). 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 (Zhao, Gu, Li, & Zhang, 2014), 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 (Anderson, Paul, & McGrew, 2018). Through this work, a random forest classifier was ultimately created which distinguishes TLS flow generated by malware (Anderson & McGrew, 2017). In (P. Prasse & Havelka, 2017), 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 (J. Liu & Liu, 2019) 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 which analyzes persistent communications, instead of the presence of anomalies within the persistent communication itself, is offered in (Kohout & Pevny, 2015).

Encryption also plays a significant role in 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 (Sultan, Khalique, Alam, & Tanweer, 2018) and (Tailor & Patel, 2017), including details on the different families of ransomware, notable attacks, and prevention techniques. An early-warning scheme which allows for the detection of potential malware was developed in (Scaife, Carter, Traynor, & Butler, 2016) 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.

2.3 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. The Zeus malware, which emerged in the late 2000s, is a trojan horse malware that became a significant threat in the banking industry. Binsalleeh et. al. provided an in-depth analysis of the Zeus botnet in their paper (Binsalleeh et al., 2010). 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 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 (Park, Park, & Kim, 2014). A framework for detection of the DarkComet RAT was developed in (Awad, Sayed, & Salem, 2017), 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 mal-

ware development may be found in crypto-ransomware applications. A survey of trends in ransomware development is given in (Craciun, Mogage, & Simion, 2019); 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 example. A novel approach to discover the artefacts of malware communications using crypto-libraries found in Microsoft Windows is described in (McLaren, Buchanan, Russell, & Tan, 2019). 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 (Adamov et al., 2019) found the scheme used to encrypt the affected data was AES with CTR mode.

Chapter 3

Ciphertext Patterns Generated by Stream Ciphers

In this chapter, the statistical weakness found in the ciphertexts generated by stream cipher encryption is presented in more detail, including supporting information describing the mechanism of the stream cipher and its notable weaknesses.

3.1 Stream Cipher Secrecy

Cryptography is a technique used for secure message sharing; encryption is the process of converting a *plaintext* message to a secure *ciphertext*; the reverse operation is known as decryption. The process of encrypting the message m can be represented symbolically as: $E(k,m) = c$, where E is the encryption mechanism or *cipher*, k is the *key* used by the cipher, and c is the ciphertext generated by the procedure. It may also be denoted as $E_k(m) = c$. E is considered public, a standardized algorithm that is known amongst all parties, while k is private and only known between privileged parties. The goal of encryption is to conceal the true meaning of some plaintext message so that when it is shared, only the approved entities are able to uncover the contents of the original mes-

sage. While E provides the actual function of converting plaintext to ciphertext and vice versa, it is k that actually provides the security to the message and allows c to be decrypted by an entity who has access to the key. There are many forms of encryption, varying in complexity and promised security, an implementer must balance the different needs of their application in order to ensure their cipher is the best solution.

Symmetric key encryption, in particular, is a basic enciphering method in which privileged parties use the same cryptographic key for both encryption and decryption. There are two paradigms of symmetric encryption, determined by the process through which the cipher divides characters of the message. One technique, referred to as a block cipher, processes a message in blocks of data in pre-defined lengths. In contrast, the *stream cipher* processes the message as a stream of data (i.e. bit-by-bit or character-by-character), encrypting the message m by processing each character m_i such that $0 \leq i < L$ and $m_0 \leq m_i < m_L$, where L is the length of m. Consider the simple example provided in Figure 3.1, here the plaintext message "HELLOWORLD" is combined with the secret key (pad) "BLETCHLEYP" via an XOR operation on a per-character basis to obtain the final ciphertext. To compute the XOR, each character must first be converted to a numerical value: this is done via a *character encoder*. Character encoding is a method computers use to interpret non-binary values. Such characters are converted to a standardized binary value used to represent the character. Examples of such encoding schemes include the American Standard Code for Information Interchange (ASCII) and the Unicode Transformation Format (UTF-X).

The stream cipher is modeled after the One-time Pad, an encryption technique first introduced by a banker named Frank Miller in 1882, however, the form that is most widely known today can be attributed to Gilbert Vernam. In 1919, Vernam designed an encryption device that would perform an XOR operation between the characters in a plaintext message and a stream of characters that were generated randomly (Bellovin, 2011). The security of the Vernam Cipher, as it is sometimes known, is theoretically *absolute* or *perfect* when the key is generated completely at random and never reused. The One-time Pad was formally proven to be perfectly secure by the mathematician

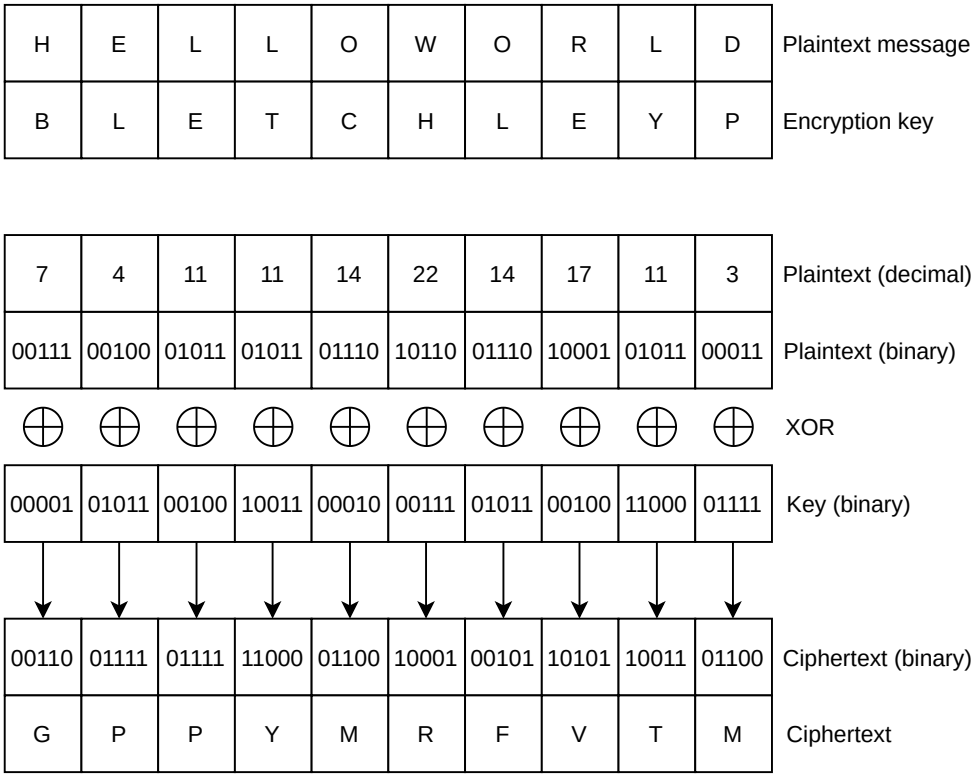


Figure 3.1: Elementary depiction of a stream cipher executing XOR operation.

Claude Shannon in 1946 (Shannon, 1949). In summary, his definition of perfect security is that upon interception of some encrypted message c, an adversary is unable to learn anything or make any deductions given just c. Suppose there are infinitely many possible m's and the *a priori* probability of any m is $P(m)$, then $P(m)$ and the *a posteriori* probability of m given c must be equal (i.e. $P(m) = P(m|c)$). Conversely, the statement $P(c) = P(c|m)$ must also hold true. This can be justified using Bayes' Theorem, which takes the general form:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}. \tag{3.1}$$

And the likelihood of c being generated by the encryption of m may be expressed as

$$P(c|m) = \frac{P(m)P(m|c)}{P(c)}, \tag{3.2}$$

where:

- $P(c|m)$ is the posterior likelihood of c given m,
- $P(m)$ is the prior probability of m among all possible m's,
- $P(m|c)$ is the sum of probabilities of all encryptions of m which generate c, and
- $P(c)$ is the probability of c across all possible c's.

Though it provides theoretic perfect security, the One-time Pad is not viable for real world application due to the stipulations regarding the key. First, the key must be as long as the messages it encrypts/decrypts which makes the key sharing process cumbersome and costly. Secondly the key must never be reused, so sending a new key with every message is redundant because if the participants have a method of sharing the key privately, then they can also use this method to share messages securely. To mitigate these issues surrounding the key, modern stream ciphers are implemented using a keystream generated by a *key scheduling algorithm*. Instead of explicitly constructing a pad the length of the message, a random keystream can be produced using a *pseudorandom generator* (PRG) taking k as seed (Boneh, n.d.). Finally, the pseudorandomly-generated pad is XORed with the plaintext or ciphertext to perform encryption or decryption, respectively.

3.2 Vulnerabilities Stemming from Key Generation

Security of the One-time Pad is facilitated through use of a uniformly distributed and truly random secret key. Because the pad $G(k)$ produced by a PRG is not truly random, security of the cipher relies on the unpredictability of the PRG. In the same way that a truly random key creates indiscriminate mappings of m to c, a PRG should generate $G(k)$ such that each byte of the pad is statistically independent, i.e. $P(k_i|k_j) = P(k_j|k_i) = 0$ holds true for all i-th and j-th bytes of $G(k)$, and the potential values of each i-th byte are well-distributed. Creating unpredictability is hard to

accomplish and if not implemented carefully, gives rise to various vulnerabilities which propagate into the encryption scheme.

3.3 Discriminatory Patterns in Ciphertexts

Son, et. al. discovered a statistical weakness in the ciphertexts generated by RC4 in (Son et al., 2019). Using this we weakness, the authors proposed a machine learning application capable of identifying RC4 ciphertext generated under a fixed key and later, the solution was used to detect encrypted malware communication. The authors found that ASCII-encoded ciphertexts generated under these conditions produce a discernible pattern which allows a trained classifier to identify other ciphertexts generated under the same conditions when compared to those which were generated through other means. An example of these patterns can be found in Figure 3.2.

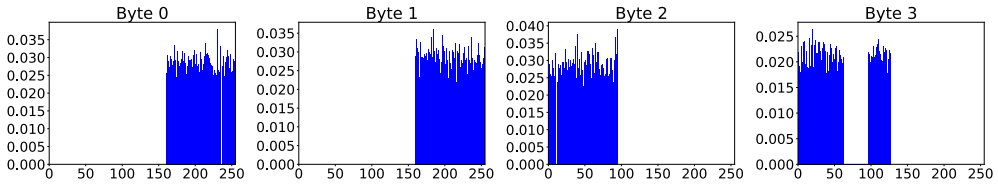


Figure 3.2: The distribution of bytes over the first four bytes of RC4 ciphertexts.

Figures 3.2 and 3.3 depict the first four bytes of ciphertexts generated from 10,000 encryptions using RC4 and AES with Cipherblock Chaining (CBC) mode under a fixed key, respectively. Each byte (i.e. character) of the ciphertext has 256 possible values, the dependent variable in the plots of Figures 3.2 through 3.4 is the probability that each value appears at that position of the ciphertext.

AES-CBC, which operates as a traditional block cipher, creates no obvious pattern. Even though the probabilities of values across each byte are not perfectly uniform, all values are well-represented and given an infinite number of samples, the probabilities should approach a uniform distribution. However, in the case of RC4, only 95 of the possible 256 values are ever expressed regardless of how many encryptions are performed. Using the two figures as examples, the non-uniformity and

gaps left in the ciphertexts generated via RC4 compared with those of a well-distributed function are what facilitate the proposed discrimination function.

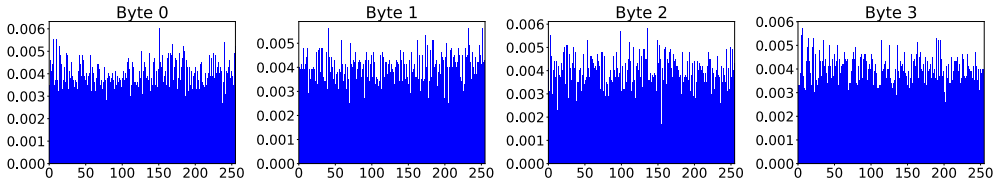


Figure 3.3: The distribution of bytes over the first four bytes of CBC ciphertexts.

3.3.1 The Effect of Character Encoding on Pattern Generation

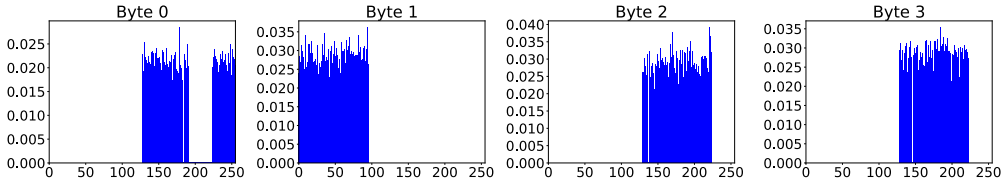
Human language is defined over a set of possible characters, with different languages being defined over different sets. The common English language, for example, includes the alphabetical characters in both upper and lower cases, the Arabic numerals, and a collection of special characters used for punctuation, mathematical expressions, etc. Together, these characters sum to a set of size 95. This set, known as *printable characters*, are those which can be found on the standard keyboard and can be printed to the screen on input. Computers understand binary, requiring an encoder to interpret all characters other than 0 and 1. In the simple case of ASCII, characters are interpreted using a binary number the size of one byte. This allows for all values that it can encode to be represented as 1 of 256 possible values, however, the printable characters do not occupy the entire range. This limited range of expressed values gives rise to the gaps found in Figure 3.2.

3.3.2 Encryption Schemes Displaying Ciphertext Patterns

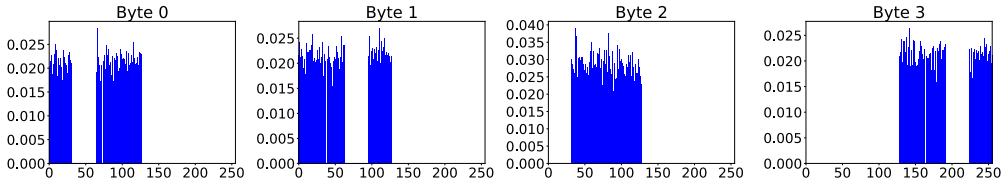
Using this knowledge in conjunction with the mechanism of the stream cipher, it is possible to use this discrepancy as a statistical weakness when analyzing the cipher. Stream ciphers encrypt on a per-character basis via a direct mapping of a plaintext character to a ciphertext character. For any

given encryption, there is only ever one mapping of a plaintext character to a ciphertext character at each byte of the message and both must exist in this truncated range of just 95 possibilities.

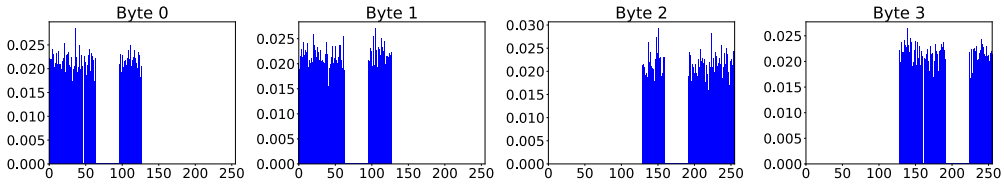
Other stream ciphers and block ciphers whose mode of operation causes it to perform like a stream cipher were tested have been examined in addition to RC4, which also generate similar statistical patterns. The study was extended to the following schemes: ChaCha20-Poly1305, Salsa20, and to the CTR, OFB, and GCM modes of operation. Figure 3.4 displays the patterns generated under repeated fixed-key encryptions of 10,000 random messages containing the full array of printable ASCII values.



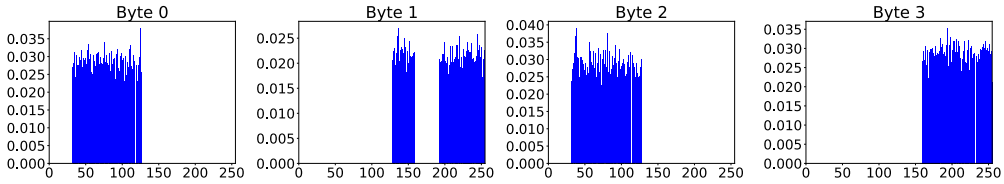
(a) ChaCha20-Poly1305



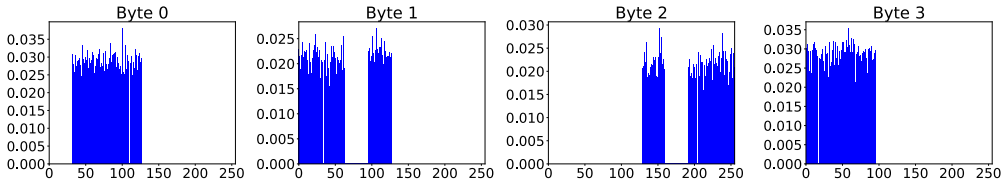
(b) Salsa20



(c) AES with CTR mode



(d) AES with OFB mode



(e) AES with GCM mode

Figure 3.4: The distribution of values over the first four bytes of ciphertexts generated by various stream ciphers.

Chapter 4

A Proposed Solution for Detecting Stream Ciphers

In the previous chapter, a statistical weakness in the ciphertexts generated by stream ciphers and block cipher modes of operation which operate similar to stream ciphers was presented. Because this weakness produces distinct patterns which are unique to a given encryption scheme and key, they can be used as a fingerprint for the detection of the encrypting pair. Then, the patterns can be used to construct a model capable of predicting the likelihood that a message was encrypted by the given scheme under the specified key. The remainder of this chapter details the formation of a discrimination function which, when trained, is capable of classifying ciphertexts encrypted by E_k from those that are not.

4.1 Representing the Discovered Patterns Mathematically

Using the results from the generated ciphertexts, the patterns can be represented using the two-dimensional matrix $M =$

$$\begin{bmatrix} x_{0,0} & x_{0,1} & \dots & x_{0,255} \\ x_{1,0} & x_{1,1} & \dots & x_{1,255} \\ \vdots & \vdots & \vdots & \vdots \\ x_{l,0} & x_{l,1} & \dots & x_{l,255} \end{bmatrix}$$

where x_{ij} is each cell of M, i is the byte (position) in the ciphertext, j is the decimal representation of the value found at c_i ($0 \leq j \leq 255$), and $l = L - 1$. Consider the simplified example in Figure 4.1 where there are only 10 characters in an arbitrary alphabet, so $0 \leq j \leq 9$. The current message is "HELLO", in which $i_0 =$ 'H', $i_1 =$ 'E', etc, below each character is its respective decimal representation, j, in the truncated range. Since the example includes one message, its length is used to determine $l = 5 - 1$.

H	E	L	L	O
2	1	4	4	7

$$\begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & x_{0,4} & x_{0,5} & x_{0,6} & x_{0,7} & x_{0,8} & x_{0,9} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} & x_{1,6} & x_{1,7} & x_{1,8} & x_{1,9} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} & x_{2,6} & x_{2,7} & x_{2,8} & x_{2,9} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} & x_{3,5} & x_{3,6} & x_{3,7} & x_{3,8} & x_{3,9} \\ x_{4,0} & x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} & x_{4,5} & x_{4,6} & x_{4,7} & x_{4,8} & x_{4,9} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Figure 4.1: Construction of M from one message.

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 for multiple calculations such as 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.

In this paper, we propose machine learning-based schemes that can detect ciphertexts generated by

stream ciphers. First, it is shown how the model can be trained with or without knowledge of the encryption key. Then, two detection scenarios are presented: (i) when a complete set of ciphertext is available and (ii) when only a partial ciphertext is available.

4.2 Constructing a Detection Model

The discrimination model is trained based on the two following scenarios:

- the key is known and available or obtainable through some means of investigation,
- a fixed key is used for encryption, but its true value is unknown.

Because these patterns are generated under a fixed key, the key is not necessary to perform the analysis if it is assumed to be constant. Training the models in either scenario is the same, the only step that changes is the data used to train the model. In the first scenario, knowledge of the key allows the model to be determined theoretically, yielding the optimal dataset by populating M using perfect knowledge of the encryption system. The key can be used to determine all possible encryptions deterministically, making it possible to deduce precisely the range of bytes which the cipher will map to. However, in the second scenario when the key is unknown, the patterns must be determined empirically, populating M with data collected from the source of encryptions.

In the case where the key is known, a type of chosen plaintext analysis of the cipher is performed in which every possible printable value is encrypted. 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 $32_{10} \leq j < 127_{10}$. Each message consists of the repeated character x_j, for example: when x_j = 'A', the encrypted message is 'AAAA...' for some length L. This is performed for each character (95 times), determining the encryption of each possible character at each position over L. Finally,

for each c in the collection of calculated ciphertexts, M is populated by iterating over each c to determine c_{ij} and incrementing the value of M at the ij-th bin.

Without access to the encryption key, the same test can not be completed in indistinguishable time. Instead, assume that a sufficiently large set of ciphertexts generated under the specified conditions can be captured. Then, using these samples, iterate over each character in each message of the set, populating M using 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.

Algorithm 1 Proposed Training Model

Generate training data:

if 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

else

Generate D, through network monitoring. Captured suspicious messages are formatted such that each character in the message is represented as its respective decimal value

end if

Construct training model:

$M = \sum_{i=0}^{L} \sum_{j=0}^{255} a_{ij} = 0$

for message $\in$ **D do**

    $l = 0$

    for character $\in$ message **do**

        $M[l][character]++$

        $l++$

    end for

end for

Calculate probability distributions:

for position $\in$ **M do**

    $sum = \sum_{i=0}^{L} position[i]$

    for val $\in$ position **do**

        $val = val \div sum$

    end for

end for

4.2.1 Calculating Probabilities Using the Ciphertext Patterns

M is now a collection of encryptions of all printable ASCII characters generated by E_k. Due to the nature of stream ciphers, the encrypted value can only exist as one of 95 values and because the encrypted messages are generated using a fixed key, a one-to-one relationship can be observed between x and c. So using the patterns created by M, the probability that a given message was generated by E_k can be calculated.

First, the elements of M are converted from sums of observations to the respective frequencies by dividing each element by the total number of messages which were analyzed at each c_i. Now, each element in M is equivalent to $P(c_{ij}|E_k)$, or the expected probability of the observed value at each byte given the cipher and encrypting key. In order to determine the probability that a ciphertext was generated by the encrypting pair E_k, the a priori likelihood is calculated using Bayes' Theorem (recall Equation 3.1 defined in Chapter 3). Once $P(E_k|c)$ is calculated, the discrimination function (4.2) is used to determine the binary classification of the result, where θ is the classification rule also referred to as the *threshold*.

$$P(E_k|c) = \frac{P(c|E_k) \cdot P(E_k)}{P(c)}. \tag{4.1}$$

$$P(E_k|c) \underset{\neg E_k}{\overset{E_k}{\gtrless}} \theta \tag{4.2}$$

The independent probabilities of $P(E_k)$ and $P(x)$ are constant: $P(E_k) = P(\neg E_k)$ and assuming a uniform distribution of the possible byte-values of c, any ciphertext c_1 is just as likely to exist as another ciphertext c_2. Equation (4.1) can be simplified to the proportional relationship:

$$P(E_k|c) \propto P(c|E_k). \tag{4.3}$$

That is, the probability of E_k given c is equivalent to the probability of c given E_k. This relationship can be used in conjunction with 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:

$$P(c|E_k) = \prod_{i=1}^{L} P(c_i|E_k). \tag{4.4}$$

4.2.2 Determining a Classification Threshold

The model considers a simple binary classification: a message is generated by E_k or it is not. Since the binary classifier is trained using labeled data, θ is determined as a point on a linear line which perfectly separates the two classes depending on the length of the message due to the variable length of the captured information. An optimal threshold can be determined by calculating the probability c was generated truly randomly (ideal case).

Let us consider a ciphertext c with length L. In Chapter 3 it was described how the ASCII encoding scheme uses one byte to represent all possible characters which, in theory, provides 256 possible values to be represented by a single character. Each character of the ciphertext c, c_i, can be considered to have probability $P(c_i) = 1/256$. Because a stream cipher encrypts byte-by-byte, each c_i can be considered as an individual output. For a number to be truly randomly generated, each output event must be independent of another. With this consideration, the probability of any c can be calculated as the product of the independent probabilities of each c_i:

$$P(c) = \prod_{i=0}^{L} P(c_i). \tag{4.5}$$

Because the calculated probabilities are small, the log-likelihood is taken as:

$$P(c) = \prod_{i=0}^{L} \ln(P(c_i)). \tag{4.6}$$

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

$$P(c) = \prod_{i=0}^{L} \frac{1}{256} \; or \; \frac{1}{256} \times L. \tag{4.7}$$

However, the described weakness of stream ciphers 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 only has the probability of $P(c_i) = 1/95$ 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 $P(c_i), 0 \leq c_i < 95$, an optimal threshold can be calculated which perfectly separates both classifications.

The optimal threshold may be found by calculating (4.7) for the 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.

It is also possible 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, these types of repeated messages can be taken into account while 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 all messages in the set are expected to be generated

under the same conditions, they can be ordered 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, the probability of the message given a uniform distribution of bytes at each element of the message is calculated using (4.7) and denoted as P^-. Finally, the threshold is calculated 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, θ must be defined as a function of message length: as the length of the message increases, more information is gained which affects the likelihood that it was produced by E_k. 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, the simple linear equation: $\theta = mL + \theta_0$ is used to calculate θ as a function of L, where m is the slope of the line and θ_0 is the intercept.

4.3 Detecting Ciphertexts Generated by E_k

Once the model is created, it can then be used to make predictions about whether a ciphertext was generated by the pair E_k or not. A ciphertext can be obtained in two conditions which will affect how the threshold is determined:

- the ciphertext is completely available
- the ciphertext is only partially available.

4.3.1 Using a Complete Ciphertext

First, consider the scenario when as much information as possible is available: the complete ciphertext c generated by E_k. We denote the likelihood that E_k was used to generate c as $P(E_k|c)$ which we defined to be proportional to $P(c|E_k)$ in (4.3). Thus, by calculating the independent

probabilities of each byte in the ciphertext c given the stream cipher E_k, $P(E_k|c)$ can be determined as:

$$P(E_k|c) \propto \prod_{i=1}^{L} P(c_i|E_k). \tag{4.8}$$

Presence of a stream cipher may be detected using the discriminant function found in (4.2). Here, the likelihood that the given stream cipher was used to generate the ciphertext is evaluated against the threshold (θ) to determine whether c can be classified as a ciphertext generated by E_k or not. The steps taken to calculate (4.8) and appropriately classify c are provided in Algorithm 2.

Algorithm 2 Detection of a complete ciphertext

Training model:
Use the training model in Algorithm 1
Calculate probability:
$p = 0$
$i = 0$
for $x_i \in x$ **do**
    $p_i = M[i][x_i]$
    $p = p + \ln(p_i)$
    $i++$
end for
Determine threshold
$l = length(x)$
$\theta = \ln(1/256) \times l$
Detection
if $p > \theta$ **then**
    return $\mathbf{E}_k$
else
    return $\neg\mathbf{E}_k$
end if

4.3.2 Using a Partial Ciphertext

The detection scheme also considers the scenario in which a ciphertext is only partially available. In the previous section, it was described how the proposed scheme detects ciphertexts encrypted

by stream cipher when the positions of values in 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 level of data loss. Since the patterns observed in the collected ciphertexts are position-dependent, the proposed scheme may not function properly when presented with only partial information.

Suppose that only a slice of a network packet is analyzed, $s = \{s_i | 1 \leq i \leq L_s\}$ and $s \subset c$, where the size of the partial packet is much shorter than that of the ciphertexts gathered previously, i.e., $L_s \ll L$. $P(E_k|s)$ must be determined, which can then be used to infer if $c = E(k,x)$. The first step is to determine 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:

$$\ln P(i|s,E_k) = \sum_{j=1}^{M} \ln P(x_{i+j-1} = s_j|E_k), \tag{4.9}$$

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:

$$P(x_i|E_k) = \frac{c_i(x_i) + \alpha}{\sum_{m=0}^{255} c_i(m) + \alpha} \tag{4.10}$$

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:

$$i^* = \underset{i}{\operatorname{argmax}} \ln P(i|s,E_k), \tag{4.11}$$

where $1 \leq i \leq L - L_s + 1$. An example of the log-likelihood of a partial packet is illustrated in Figure 4.2. Equation 4.9 is used to 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 4.2, 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.

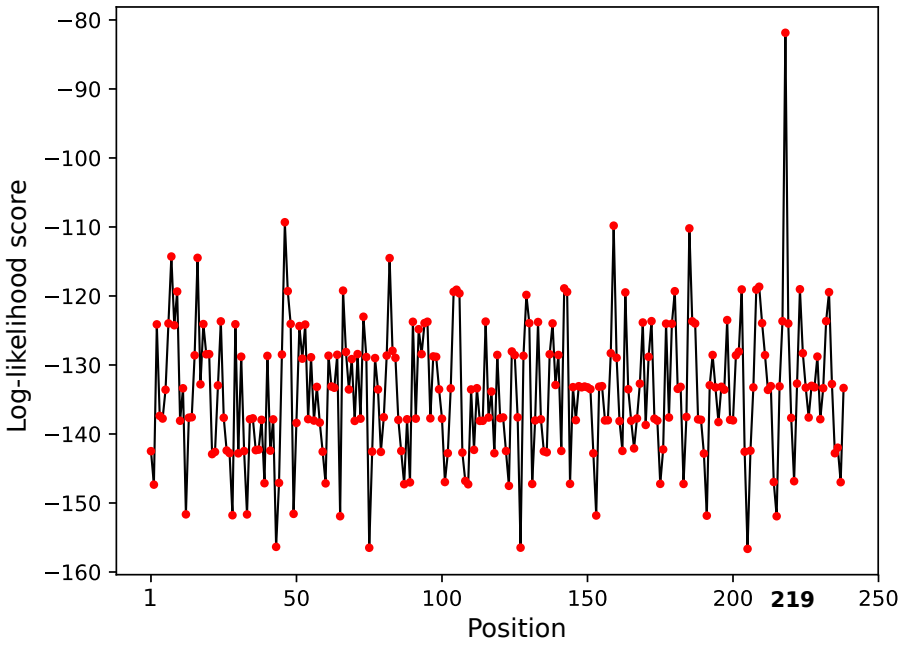


Figure 4.2: Log-likelihood of a partial packet across a network packet.

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:

$$\begin{aligned}\mathcal{L}(s,k) &= \ln P(E_k|s) \\ &\propto \ln P(i^*|s,E_k) \\ &= \sum_{j=1}^{M} \ln P(x_{i^*+j-1} = s_j|E_k). \end{aligned} \tag{4.12}$$

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

$$\mathcal{L}(s,k) = \sum_{j=1}^{M} \ln P(x_{i^*+j-1} = s_j|E_k) \underset{\neg E_k}{\overset{E_k}{\gtrless}} \zeta. \tag{4.13}$$

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** $\neg\mathbf{E}_k$

4.4 Evaluation of the Proposed Scheme

The scheme proposed in this chapter was evaluated in regards to the following criteria:

- minimum detectable ciphertext length,

- detection time and accuracy,
- distinguishability from ciphertexts generated using a random key, and
- distinguishability from patterns produced by other ciphers.

Evaluations were conducted using models for each of the six enciphering schemes. The models were constructed from synthetic data with a fixed key as described previously in Section 4.2 of this chapter. Then, testing data was generated by encrypting 200 messages at various lengths for each encryption scheme. The messages were split in half, with 100 messages encrypted under a fixed key (the same used to create the model) and the other 100 under randomized keys. Results are provided in the following subsections.

4.4.1 Determining the Minimum Detectable Message Length

We begin by finding the minimum length at which the scheme can discriminate ciphertexts with confidence. Because the proposed scheme uses the gaps in the distribution of expressed values at different bytes over a ciphertext, it must be presented with sufficient information to make a justifiable assessment. The probability at any given byte alone is not sufficient to determine the likelihood of E_k as many different configurations may produce distributions which overlap the distributions of values at the same bytes of E_k. The results of classifying ciphertexts of lengths 1 to 15 bytes are presented in Tables 4.1 and 4.2.

As expected, the model cannot achieve perfect accuracy using only a few bytes and performance is very poor in a few cases. Unsurprisingly, the *true positive rate*, or TPR, is relatively high even in these short testing cases because it is likely that the value is expressed at the current byte, but the frequency of its expression may not be high enough to overcome the threshold. The *false positive rate* (FPR) affects the model's accuracy most significantly: insufficient data presents *Type I error* as it is likely for the distributions of values to have some overlap, causing $\neg E_k$ ciphertexts to be

falsely classified as E_k. All models have slightly different lengths at which they achieve 100% accuracy, but for the remainder of this paper we will consider a message length of 16 bytes to be a safe lower limit for classifiable messages.

Table 4.1: Minimum detectable ciphertext lengths for RC4, ChaCha20, and Salsa20 ciphers.

		RC4			CHA			SAL		
Length	θ	TPR	FPR	Accuracy	TPR	FPR	Accuracy	TPR	FPR	Accuracy
1 byte	-5.55	0.990	0.118	0.900	1.000	0.378	0.685	0.990	0.414	0.653
2 bytes	-11.09	0.970	0.000	0.995	0.970	0.208	0.822	0.970	0.214	0.817
3 bytes	-16.64	0.960	0.000	0.993	0.990	0.000	0.998	0.950	0.080	0.925
4 bytes	-22.18	0.970	0.000	0.995	0.960	0.000	0.993	0.970	0.000	0.995
5 bytes	-27.73	0.950	0.000	0.992	0.960	0.000	0.993	0.960	0.000	0.993
6 bytes	-33.27	0.990	0.000	0.998	1.000	0.000	1.000	0.990	0.036	0.968
7 bytes	-38.82	1.000	0.000	1.000	1.000	0.000	1.000	1.000	0.040	0.967
8 bytes	-44.36	0.990	0.000	0.998	1.000	0.000	1.000	1.000	0.012	0.990
9 bytes	-49.91	1.000	0.000	1.000	0.990	0.000	0.998	1.000	0.000	1.000
10 bytes	-55.45	1.000	0.000	1.000	0.990	0.000	0.998	1.000	0.000	1.000
11 bytes	-61.00	1.000	0.000	1.000	0.990	0.000	0.998	0.990	0.000	0.998
12 bytes	-66.54	1.000	0.000	1.000	0.990	0.000	0.998	1.000	0.000	1.000
13 bytes	-72.09	1.000	0.000	1.000	1.000	0.000	1.000	1.000	0.000	1.000
14 bytes	-77.63	1.000	0.000	1.000	1.000	0.000	1.000	1.000	0.000	1.000
15 bytes	-83.18	1.000	0.000	1.000	1.000	0.000	1.000	1.000	0.000	1.000

Table 4.2: Minimum detectable ciphertext lengths for OFB, CTR, and GCM modes.

		OFB			CTR			GCM		
Length	θ	TPR	FPR	Accuracy	TPR	FPR	Accuracy	TPR	FPR	Accuracy
1 byte	-5.55	0.990	0.134	0.887	0.990	0.476	0.602	1.000	0.476	0.603
2 bytes	-11.09	0.980	0.000	0.997	0.980	0.180	0.847	0.980	0.000	0.997
3 bytes	-16.64	0.970	0.000	0.995	0.990	0.082	0.930	0.980	0.000	0.997
4 bytes	-22.18	0.940	0.000	0.990	0.960	0.000	0.993	0.960	0.000	0.993
5 bytes	-27.73	0.930	0.000	0.988	0.980	0.000	0.997	0.960	0.000	0.993
6 bytes	-33.27	1.000	0.000	1.000	0.990	0.038	0.967	0.980	0.000	0.997
7 bytes	-38.82	1.000	0.000	1.000	1.000	0.048	0.960	1.000	0.000	1.000
8 bytes	-44.36	0.990	0.000	0.998	1.000	0.024	0.980	1.000	0.000	1.000
9 bytes	-49.91	1.000	0.000	1.000	1.000	0.000	1.000	1.000	0.000	1.000
10 bytes	-55.45	0.990	0.000	0.998	1.000	0.000	1.000	1.000	0.000	1.000
11 bytes	-61.00	1.000	0.000	1.000	1.000	0.000	1.000	1.000	0.000	1.000
12 bytes	-66.54	1.000	0.000	1.000	1.000	0.000	1.000	1.000	0.000	1.000
13 bytes	-72.09	1.000	0.000	1.000	1.000	0.000	1.000	1.000	0.000	1.000
14 bytes	-77.63	1.000	0.000	1.000	1.000	0.000	1.000	1.000	0.000	1.000
15 bytes	-83.18	1.000	0.000	1.000	1.000	0.000	1.000	1.000	0.000	1.000

4.4.2 Detection Time and Accuracy

Following the determination of 16 bytes as the minimum message length, models were evaluated to verify their detection accuracy, as well as to calculate an average detection time at each length. Table 4.3 shows θ at each test message length along with two values for each encryption scheme: the maximum and minimum likelihoods calculated from the test messages at each L for the respective cipher. The *min(P)* denotes the likelihood of the message most probable to be classified *false negative*, i.e. it would be marked as $\neg E_k$ when it should actually be classified as E_k. The times taken to determine the classification are provided in Table 4.4. Times are listed in microseconds (μs) with two standard deviations.

Table 4.3: Minimum and maximum likelihoods calculated at various message lengths.

Length	θ	$P(E_k)$	RC4	CHA	SAL	OFB	CTR	GCM
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

4.4.3 Distinguishing Random-key Ciphertexts

Next, it is shown that a model can perfectly distinguish ciphertexts generated under a random key. Table 4.5 displays the calculations of the maximum likelihood amongst 100 ciphertexts, representing the most likely to be *false positive*. The proposed scheme separates the data well and the likelihoods of cases closest to the threshold diverge as more characters are taken into account, indicating the proposed scheme can distinguish fixed-key ciphertexts from those generated using a random key under the same cipher.

Table 4.4: Average detection times in μs.

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

Table 4.5: Results of testing the models with ciphertexts generated under a random key.

Length	Theta	RC4	CHA	SAL	OFB	CTR	GCM
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

4.4.4 Distinguishability Between Models

Finally, we show that the model for each cipher can be distinguished from another. Using the fixed-key training set, each model is tested with the 100 ciphertexts of the other five encryption schemes at each respective length. As shown in Table 4.6, the maximum likelihood amongst the 500 tests is listed, representing the message most likely to be classified as a ciphertext generated by the current cipher, i.e. a *false positive*. The results 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.

Table 4.6: Distinguishing ciphers using relative most-probable false positive.

Length	Theta	RC4	CHA	SAL	OFB	CTR	GCM
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

Chapter 5

Simulating Malware Detection Using the Proposed Solution

The previous chapters have presented a statistical weakness found in the ciphertexts generated by a stream cipher and proposed a probabilistic technique which may be used to detect such ciphertexts in different scenarios. This chapter details a simulation in which the discrimination function presented in this paper is used to detect malicious communications across a network.

Some encrypted malware are implemented with a pre-stored encryption key in order to mitigate the difficulty of key sharing upon deployment to the victim machine. In this case, it would be possible to obtain the stored key via source code-checking or reverse engineering. The optimal detection model can be determined if the key is uncovered, however, other malware might exploit a technique such as code obfuscation, hiding the key and rendering its acquisition almost impossible. In this case, the model would be determined empirically; both scenarios can be addressed by the proposed discrimination scheme. Through implementation of this scheme, current network monitoring tools can be strengthened by using the technique to identify this type of encrypted malware.

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 widely used by malware programmers due to its speed, ease of implementation, and legacy support in the aforementioned secure technologies. 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. The following sections describe the process of how the proposed detection algorithm could be used to detect malware in a situation where it is actively sending information between a victim and attacking machine using DarkComet.

5.1 The DarkComet Testing Environment

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

DarkComet infects a victim machine through use of a *stub* which, when opened on the machine, installs the payload and initiates the connection with the attacker. The stub was copied to the victim VM via USB and opened. Once connected, DarkComet begins to send its encrypted messages via TCP: the Wireshark network monitoring tool was used on the attacking VM to capture communication between the two machines. Because the testing environment was set-up offline, filtering the packets generated by DarkComet was trivial.

5.2 Generating Data Using DarkComet

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

including, but not limited to: screen capture, access to the secondary machine's shell, and 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 which uses Microsoft Reader to read input text on the primary machine and play the audio on the secondary machine, a piano which plays sounds on the secondary machine, and options to show and hide GUI elements (clock, taskbar, desktop). These functions were the primary method of data generation as they are the most easily repeatable.

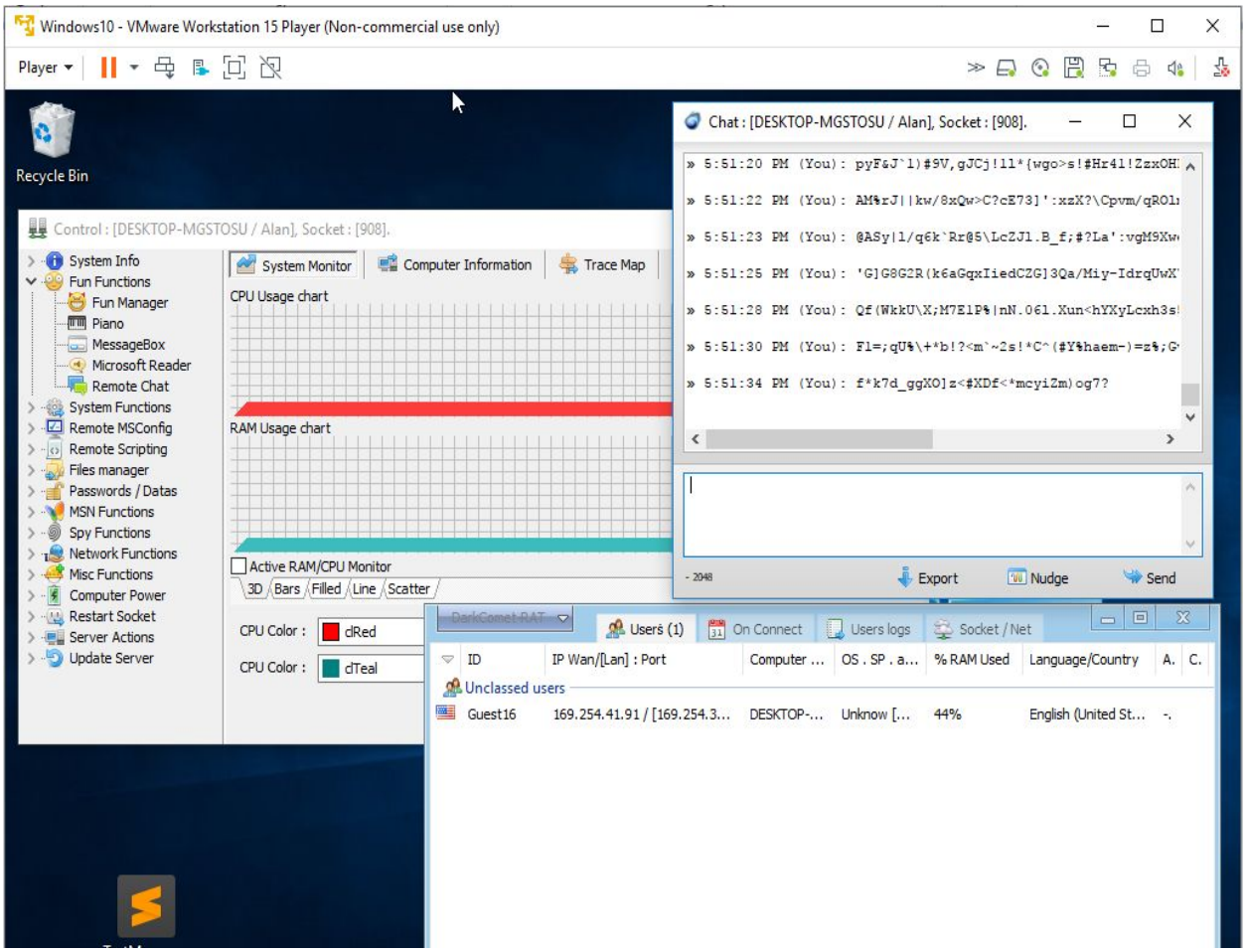


Figure 5.1: Screenshot of the DarkComet GUI including the remote chat box.

To generate communication data, the functions within the DarkComet malware were used to send Transmission Control Protocol (TCP) packets between the attacking and victim VMs; the packets were captured via Wireshark. Studying these messages, it was found that a DarkComet TCP packet consists of a header which pertains to the function used and some appended text which is

used to execute the function on the receiving machine. 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 was used as the data to train the discrimination model of the proposed scheme. Other information such as IP addresses found in the header of the packet was discarded.

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

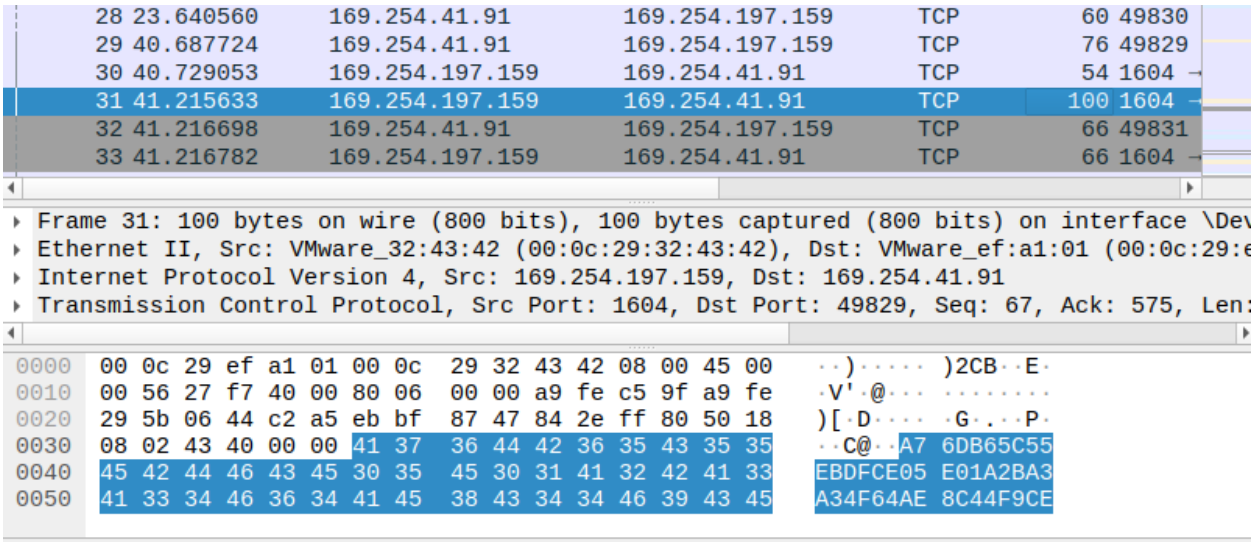


Figure 5.2: Screenshot of Wireshark intercepting a DarkComet packet with its respective payload.

5.3 Detecting DarkComet Through Packet Analysis

As mentioned previously, it is known that DarkComet employs RC4 and uses a fixed key under which all messages are encrypted and conveniently, the standard key for the version used for this simulation is known to be #KCMDDC51#-890, providing an example for the schemes proposed

in Chapter 4 to be tested on. Both methods of model construction (where the key is either known or unknown) were executed, the latter was simulated by withholding the key and training under the pretense that the only information known prior to training is which packets belong to the malware's communication traffic. Following the data generation process outlined earlier in this section, 10,000 DarkComet packets were collected and the results of the simulation are presented in the remainder of this chapter.

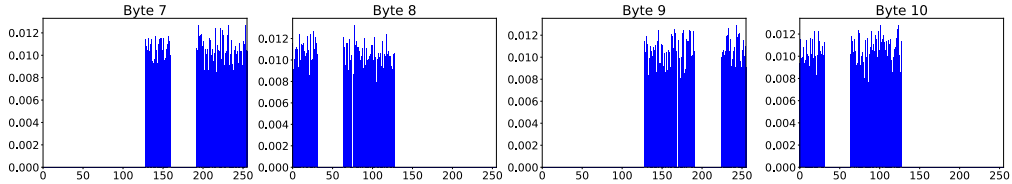
To test the models, a test message dataset was created to include collected DarkComet packets as well as benign packets. The benign packets were obtained from casual web-surfing on an uninfected machine, using Wireshark to capture the UDP packets between the machine and various web servers. User Datagram Protocol (UDP) packets contain a payload similar to TCP, so the same filtering technique can be used in order to obtain the encrypted data sent in the packet. The method used to encrypt the benign packets is not important, the messages only serve as a source of data that we can confidently label as non-malicious (i.e. not produced by DarkComet). The final testing dataset included 1,000 entries and was evenly split between DarkComet and benign packets.

5.3.1 Simulating Packet Detection

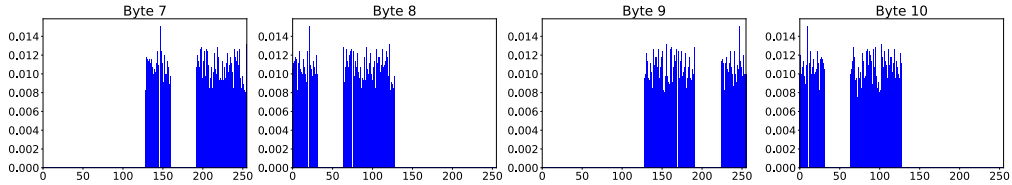
To show that the model can be effective in detecting packets produced by DarkComet, we present two sets of value distributions. Figure 5.3.a shows the distributions of four bytes of captured DarkComet packets, Figure 5.3.b shows the distributions of four bytes of RC4 ciphertexts generated using the same key used by DarkComet. Bytes 7 through 10 were chosen because the first seven bytes of DarkComet data were just the encrypted function ID which results in only one value being expressed at each byte.

As covered previously, using empirical data creates an imperfect model because it is unlikely that all possible encryptions will be observed, causing the frequencies associated with each value to be slightly different between the models. However, the patterns remain the same which allows the

scheme to maintain detection accuracy.



(a) DarkComet packets



(b) RC4 ciphertexts

Figure 5.3: The distribution of values over four bytes of DarkComet packets and ciphertexts generated by RC4 using the DarkComet Key.

5.3.2 Detecting DarkComet Packets

Using the complete array of printable ASCII characters to train the model when the key is known, it was found that the model could be trained in about 350 *ms*. Because testing this model is identical to the evaluation provided in Chapter 4, we tested the model by investigating the effects of the character set on detection time. By decreasing the number of characters available to create the model, it was found that the model maintains perfect accuracy from the complete set of 95 down to 50 characters and as expected, the training time decreased to about 230 *ms* as there was less data to be trained on.

Results of testing the model when the key is unknown (or simulated to be unknown) are given in Figure 5.1. The model was executed 100 times with each iteration of the model evaluated using 5-fold cross-validation. Training completed with an average execution time of 11 sec. Training time increases due to the importation of data, formatting the data appropriately, and training on a larger dataset which is compounded by the cross-validation scheme.

Table 5.1: Results of detecting DarkComet.

Length	Theta	Min $P(E_k)$	Detection Time (μs)
16 bytes	-88.72	-84.00	17.96 ± 2.9
18 bytes	-99.81	-97.94	20.22 ± 4.2
20 bytes	-110.90	-98.45	21.93 ± 2.5
22 bytes	-121.99	-111.50	24.81 ± 5.2
24 bytes	-133.08	-119.98	25.91 ± 2.1
26 bytes	-144.17	-136.15	28.36 ± 4.6
30 bytes	-166.36	-154.10	32.44 ± 3.8
34 bytes	-188.54	-174.00	36.50 ± 3.4
40 bytes	-221.81	-199.80	43.15 ± 1.6

5.4 Detecting Partial Packets

The results presented in Figure 5.1 use complete packets to detect DarkComet. However, as mentioned previously, each DarkComet message is prepended with a function ID. This provides for easier detection as the first bytes of any DarkComet message will always be one value of an even smaller subset of the 95 presented throughout this paper, but this also skews the model results.

Consider Table 5.2, the testing messages are all prepended with the ID *"CHATOUT"*. Because the DarkComet model has been trained on messages which share the same first 7 bytes, the messages are detected perfectly. Because it is unlikely that a byte from the benign message's first 7 bytes maps to these values (a $1/256$ chance per byte), we cannot establish a robust model by including such outliers. Observe that the minimum likelihood for DarkComet packets does not change until a message length of 8 bytes (i.e. the byte after the prepended function ID), at which point we are able to begin properly calculating $P(DarkComet)$.

Now, if the first 7 bytes are ignored and calculations start at the 8^{th} byte, we confront the issue of insufficient information (recall the evaluation of the minimum length limit in Section 4.4). Now that the true 95 values may be expressed at this byte, it is more likely that the well-distributed UDP data will collide with the DarkComet distribution. Since the most significant factor in the proposed scheme's calculation of $P(E_k)$ are the gaps in values expressed at each byte; with messages

Table 5.2: Results of detecting DarkComet at shorter message lengths.

Length	Theta	DarkComet Min	Benign Max
1 byte	-5.55	0.00	-10.00
2 bytes	-11.09	0.00	-20.00
3 bytes	-16.64	0.00	-20.00
4 bytes	-22.18	0.00	-30.00
5 bytes	-27.73	0.00	-40.00
6 bytes	-33.27	0.00	-50.00
7 bytes	-38.82	0.00	-60.00
8 bytes	-44.36	-4.83	-64.49

of shorter lengths, the algorithm does not posses enough information to make accurate classifications as can be observed in Table 5.3. Using the same test message set as before, each message is classified using just the $[8, L)$ characters. While the model maintains a perfect true positive rate (TPR), the insufficient information presents *Type I error* as observed during the theoretical evaluation.

Table 5.3: Results of detecting DarkComet at shorter message lengths after message header.

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

We can consider a partial ciphertext of 16 bytes or more and calculate the most probable position

of the partial message using Algorithm 3. To show that the first 7 bytes can be ignored, we use the partial packet ($m = [8, L)$) testing data to find the most probable starting position of the message in the model. Figure 5.4 shows that the most-likely first byte of the partial packets is point 8 in the complete ciphertext, which supports our assumption that the prepended message may be ignored so long as the resulting partial packet is at least 16 bytes long.

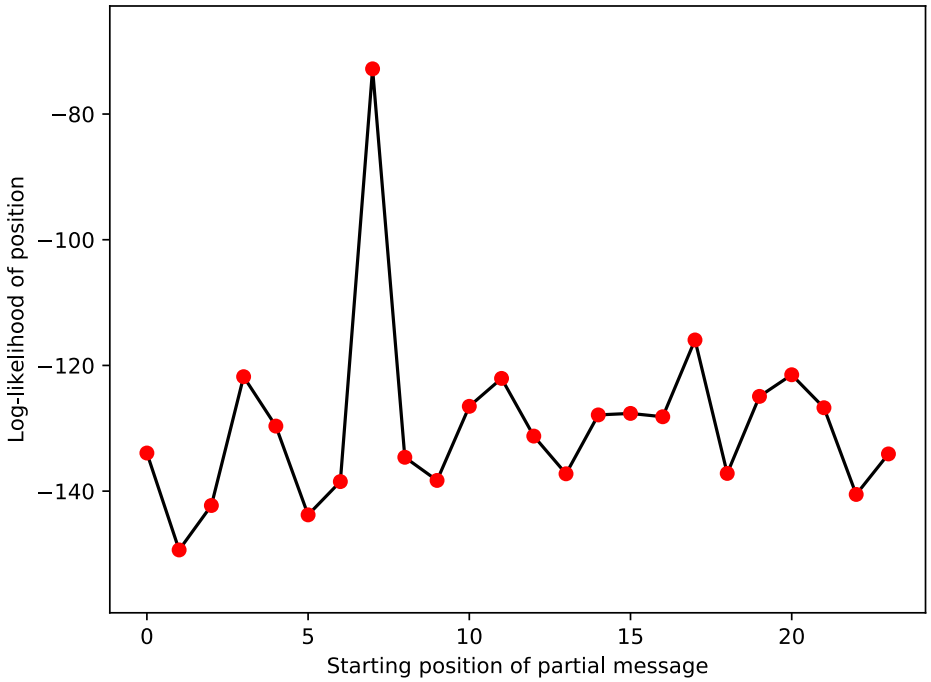


Figure 5.4: Finding the most probable starting point of a partial ciphertext in a trained model.

Chapter 6

Conclusion

This thesis expanded upon and studied a novel machine learning approach to detect the use of a stream cipher from captured ciphertexts. Through use of the many-time pad, a new statistical weakness was uncovered which is produced by typical character encoding which creates a pattern of gaps in the expressed values of generated ciphertexts. This weakness allows an analyzer to create a fingerprint for a given stream cipher and key pair from the patterns they generate as they are unique to the pair. The fingerprint may then be used to calculate the likelihood that a message was produced by the cipher and key versus a truly random generator.

The detection scheme is 100% effective in detecting ciphertexts generated by a stream cipher when the message is at least 16 bytes long. The scheme was tested with both hypothetical and real-world samples, showing that that the discrimination function can be used to effectively identify potentially malicious communications encrypted via stream cipher. Specifically, the scheme is 100% effective in detecting ciphertexts generated by RC4 with detection times as fast as 22 μs using a theoretical model and equally effective at detecting packets generated by the DarkComet RAT which employs RC4 as its encryption mechanism in about 17 μs. The quicker real-world time is attributed to the training set being explicitly trained on other collected DarkComet packets. Furthermore, it was shown that the position of a partial ciphertext may be probabilistically located

within the original ciphertext and can be used to detect the enciphering pair with perfect accuracy as long as the 16 byte minimum length is satisfied.

Network monitoring tools must constantly adapt to the increasingly complex malware environment, current tools are under-equipped to detect malware which utilizes encryption to obfuscate its malicious intentions. This discovery, along with the 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.

Bibliography

Adamov, A., Carlsson, A., & Surmacz, T. (2019). An analysis of lockergoga ransomware. *2019 IEEE East-West Design and Test Symposium, EWDTS 2019*, 1–5. doi: 10.1109/EWDTS .2019.8884472

Anderson, B., & McGrew, D. (2017). Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity. In *Proceedings of the 23rd acm sigkdd international conference on knowledge discovery and data mining* (p. 1723–1732). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3097983.3098163 doi: 10.1145/3097983.3098163

Anderson, B., Paul, S., & McGrew, D. (2018). Deciphering malware's use of tls (without decryption). *Journal of Computer Virology and Hacking Techniques*, *14*, 195-211.

Armknecht, F. (2004). Algebraic attacks on stream ciphers. *ECCOMAS 2004 - European Congress on Computational Methods in Applied Sciences and Engineering*(November).

Awad, A. A., Sayed, S. G., & Salem, S. A. (2017). A host-based framework for RAT bots detection. *2017 International Conference on Computer and Applications, ICCA 2017*, 336–342. doi: 10.1109/COMAPP.2017.8079775

Bellovin, S. (2011, 07). Frank miller: Inventor of the one-time pad. *Cryptologia*, *35*, 203-222. doi: 10.1080/01611194.2011.583711

Binsalleeh, H., Ormerod, T., Boukhtouta, A., Sinha, P., Youssef, A., Debbabi, M., & Wang, L. (2010). On the analysis of the Zeus botnet crimeware toolkit. *PST 2010: 2010 8th International Conference on Privacy, Security and Trust*, 31–38. doi: 10.1109/PST.2010.5593240

Boneh, D. (n.d.). *Cryptography i [mooc]*. Coursera. Retrieved from https://www.coursera.org/learn/crypto/

Chen, J., & Miyaji, A. (2011). A new practical key recovery attack on the stream cipher RC4 under related-key model. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6584 LNCS*(October 2010), 62–76. doi: 10.1007/978-3-642-21518-6_5

Craciun, V. C., Mogage, A., & Simion, E. (2019). Trends in design of ransomware viruses. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11359 LNCS*, 259–272. doi: 10.1007/978-3-030-12942-2_20

Denning, D. E. (1983, April). The many-time pad: Theme and variations. In *Ieee symposium on security and privacy* (p. 23-30).

Fluhrer, S. R., Mantin, I., & Shamir, A. (2001). Weaknesses in the key scheduling algorithm of rc4. In *Revised papers from the 8th annual international workshop on selected areas in cryptography* (p. 1–24). Berlin, Heidelberg: Springer-Verlag. doi: 10.5555/646557.694759

Jindal, P., & Singh, B. (2015). A Survey on RC4 Stream Cipher. *International Journal of Computer Network and Information Security*, *7*(7), 37–45. doi: 10.5815/ijcnis.2015.07.05

J. Liu, R. Z., Z. Tian, & Liu, L. (2019). A distance-based method for building an encrypted malware traffic identification framework. *IEEE Access*(7), 100014-100028.

Jungk, B., & Bhasin, S. (2017). Don't fall into a trap: Physical side-channel analysis of ChaCha20-Poly1305. *Proceedings of the 2017 Design, Automation and Test in Europe, DATE 2017*, 1110–1115. doi: 10.23919/DATE.2017.7927155

Kohout, J., & Pevny, T. (2015). Unsupervised detection of malware in persistent web traffic. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, *2015-August*, 1757–1761. doi: 10.1109/ICASSP.2015.7178272

Mantin, I., & Shamir, A. (2001). A practical attack on broadcast rc4. In *Revised papers from the 8th international workshop on fast software encryption* (p. 152–164). Berlin, Heidelberg:

Springer-Verlag. doi: 10.5555/647936.741069

Martignoni, L., Christodorescu, M., & Jha, S. (2007). OmniUnpack: Fast, generic, and safe unpacking of malware. *Proceedings - Annual Computer Security Applications Conference, ACSAC*, 431–440. doi: 10.1109/ACSAC.2007.15

McLaren, P., Buchanan, W. J., Russell, G., & Tan, Z. (2019). *Discovering encrypted bot and ransomware payloads through memory inspection without a priori knowledge.* Retrieved from http://arxiv.org/abs/1907.11954

McLaren, P., Russell, G., Buchanan, W. J., & Tan, Z. (2019). Decrypting live SSH traffic in virtual environments. *Digital Investigation*, *29*, 109–117. doi: 10.1016/j.diin.2019.03.010

Milletary, J. (2012). *Citadel trojan malware analysis* (Report). Dell SecureWorks. Retrieved from https://www.botnetlegalnotice.com/citadel/files/Patel_Decl_Ex20.pdf

Park, C., Park, H., & Kim, K. (2014). *Realtime C&C Zeus Packet Detection Based on RC4 Decryption of Packet Length Field* (Vol. 64) (No. Security). doi: 10.14257/astl.2014.64.14

P. Prasse, T. P., L. Machlica, & Havelka, J. (2017). Malware detection by analysing network traffic with neural networks. *2017 IEEE Security and Privacy Workshops (SPW), San Jose, CA*, 205-210.

Protection, E. (2020). *Threat spotlight: Tycoon ransomware targets education and software sectors* (Report). The Blackberry Research and Intelligence Team. Retrieved from https://blogs.blackberry.com/en/2020/06/threat-spotlight-tycoon-ransomware-targets-education-and-software-sectors

Roos, A. (1995). *A class of weak keys in the rc4 stream cipher.* (Posts in sci.crypt)

Scaife, N., Carter, H., Traynor, P., & Butler, K. R. (2016). CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. *Proceedings - International Conference on Distributed Computing Systems*, *2016-August*, 303–312. doi: 10.1109/ICDCS.2016.46

Sen Gupta, S., Maitra, S., Paul, G., & Sarkar, S. (2014). (Non-)random sequences from (Non-)random permutations - Analysis of RC4 stream cipher. *Journal of Cryptology*, *27*(1), 67–108. doi: 10.1007/s00145-012-9138-1

Shannon, C. E. (1949). Communication theory of secrecy systems. *The Bell system technical journal*, *28*(4), 656–715.

Sinitsyn, F. (2016). *Locky: the encryptor taking the world by storm* (Report). Kapersky Labs. Retrieved from https://securelist.com/locky-the-encryptor-taking-the-world-by-storm/74398/

Son, J., Ko, E., Boyanapalli, U. B., Kim, D., Kim, Y., & Kang, M. (2019, Feb). Fast and accurate machine learning-based malware detection via rc4 ciphertext analysis. In *2019 international conference on computing, networking and communications (icnc)* (p. 159-163). doi: 10.1109/ICCNC.2019.8685644

Sultan, H., Khalique, A., Alam, S. I., & Tanweer, S. (2018). a Survey on Ransomeware: Evolution, Growth, and Impact. *International Journal of Advanced Research in Computer Science*, *9*(2), 802–810. Retrieved from http://dx.doi.org/10.26483/ijarcs.v9i2.5858 doi: 10.26483/ijarcs.v9i2.5858

Tailor, J. P., & Patel, A. D. (2017). A Comprehensive Survey: Ransomware Attacks Prevention, Monitoring and Damage Control. *International Journal of Research and Scientific Innovation (IJRSI)*, *4*(VIS), 116–121. Retrieved from www.rsisinternational.org

Vanhoef, M., & Piessens, F. (2015). All Your Biases Belong to Us: Breaking RC4 in WPA-TKIP and TLS. *USENIX Security*, 97–112.

Verdult, R. (2015). *Introduction to cryptanalysis: Attacking stream ciphers* (Report). Institute for Computing and Information Sciences Radbound University Nijmegen, Netherlands.

Zhao, R., Gu, D., Li, J., & Zhang, Y. (2014). Automatic detection and analysis of encrypted messages in malware. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8567*(61103040), 101–117. doi: 10.1007/978-3-319-12087-4_7