Global Information Assurance Certification Paper

Learning Cryptography by Doing It Wrong: Cryptanalysis of the Vigenère Cipher

GIAC (GCIH) Gold Certification

Author: Jeremy Druin, jdruin@gmail.com
Advisor: Christopher Walker, CISSP, CCISO, GCED, GWEB
Accepted: 2/1/2018

Abstract

When studying complex ideas, it may help to begin with a simpler example to better understand its concepts. Modern cryptography and cryptanalysis are exceptionally complex, so a case study from classical cryptography can aid understanding. The Vigenère Cipher is a good example. Vigenère was widely considered to be a secure cipher for three centuries. It is non-trivial to cryptanalyze, offering a stretch goal for beginners, but not impossible to comprehend. Vigenère provides practice of multiple techniques such as statistical analysis, histograms, and Index of Coincidence. Statistical properties of files before and after encryption can be compared to show attributes that allow encrypted files to be detected. A method of detecting the encryption key length for a Vigenère cipher will be introduced. Ultimately, a strategy to recover the key for JPEG encrypted files will be demonstrated. To help the reader follow this analysis, open source software will be provided that performs encryption, decryption, and cryptanalysis. Besides learning about classical ciphers and having fun, we will reinforce the importance of proper cipher choice for the modern InfoSec professional.

1. Introduction

Cryptography, the art and science of secret writing (Merriam-Webster, 2017), is a vast and complex topic. Cryptography includes cryptanalysis techniques, the practice of deciphering or decoding encrypted messages (Collins English Dictionary, 2012). Some practical understanding of cryptanalysis techniques is important for cyber security practitioners. While advanced cryptanalysis is difficult to learn without significant time and talent, classical cryptographic ciphers are within reach of the non-professional. Specifically, simple substitution and polyalphabetic ciphers provide an introduction into cryptanalysis and demonstrate the importance of proper implementation.

Modern ciphers are significantly more advanced than the classical Vigenère cipher. A study of Vigenère is not an analog to taking on strong ciphers such as Advanced Encryption Standard (AES). The underlying cryptosystem, its mechanisms and therefore approach to attack are different (NOVER, 2004). However, Vigenère provides a good opportunity to explore and practice cryptanalysis techniques. Vigenère is non-trivial to cryptanalyze and was once widely considered unbreakable (Norman, 2017). Yet by combining multiple techniques and employing computers, Vigenère can be broken by persistent amateurs.

Two tools are provided to allow the reader to follow the examples provided in this paper. One provides encryption and decryption of files using a Vigenère cipher. The second tool, a frequency analyzer, performs statistical analysis, can determine the Vigenère encryption key length, and decrypt a file given the key. The analysis begins by comparing files before and after encryption. Statistic properties that indicate potentially weak encryption are noted. It will be shown how a Vigenère cipher can be broken if the length of the encryption key is found. Further investigation will present a method to determine the key length. Finally, the key will be recovered.

2. Overview

Monoalphabetic ciphers pick a key from a single alphabet. A single key is used to encrypt each character of plaintext. This leaves the ciphertext vulnerable to cryptanalysis via frequency analysis (Daniel Rodriguez-Clark, 2013). A form of these simple substitution ciphers was used

by the armies of Julius Caesar as early as 50 B.C. (Simmons, 2009). The monoalphabetic cipher with key = 3 is called a "Caesar Cipher" as a result.

Polyalphabetic ciphers such as Vigenère use an independent key to encrypt each character of plaintext until the end of the key is reached. At this point, the key is reused for the next block of plaintext characters (Math Explorers Club, 2003). The Vigenère cipher, a polyalphabetic cipher, is attributed to Blaise de Vigenère (Simmons, 2009) in 1586 but may have been reinvented multiple times (Norman, 2017). Leon Battista Alberti documented the general polyalphabetic cipher in 1467 although this version did not switch encryption alphabets with each letter of plaintext (University of Babylon, 2011). Giovan Battista Bellaso is credited with another variant as early as 1564 (Bellaso, 1552).

Although Vigenère was considered unbreakable for some time (Simmons, 2009), it was cracked well before computers became available (Singh, 2001). Charles Babbage decrypted a letter enciphered in Vigenère in 1846 and later solved a double-enciphered message (Kahn, 1967). However, Babbage's work was not made public (Salomon, 2003). An officer in the Prussian army, Kasiski, published a method similar to Babbage's for breaking Vigenère in 1863 (Kahn, 1967). Nonetheless, the cipher was considered secure in the popular press as late as 1917 (The Oxford Math Center, 2018).

2.1. Taxonomy of Vigenère Ciphers

Given a cryptographic key, a cipher can change "plaintext" into "ciphertext" (***Figure 1***). The term "text" is used loosely to refer to any data. Ciphers may either use a single key for both encryption and decryption operations; or uses pairs of keys--one each for encryption and decryption. Symmetric ciphers reuse the same key for both encryption and decryption while asymmetric ciphers require a pair. Symmetric ciphers can use substitution or transposition cipher primitives to transform the data. Substitution ciphers replace characters in the plaintext to generate ciphertext. Examples include "s-boxes" in the Data Encryption Standard (DES) (Gargiulo, 2002) and rotors in the

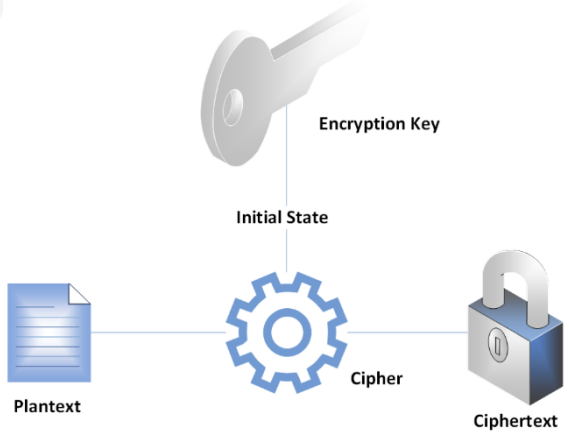

Figure 1: Ciphers transform plaintext into ciphertext and vice versa

German Enigma machine (Rijmenants, 2016). Transposition only permutes the plaintext thus "shuffles the deck". Enigma plug boards (Rijmenants, 2016) and DES "p-boxes" (Forouzan, 2007) are examples of transposition. Vigenère only uses substitution.

2.1.1. Substitution Cipher Primitives

In simple substitution, the shift of the plaintext is consistent throughout the encryption process. The Caesar cipher with its key of 3 shifts each letter of plaintext forward three positions in the alphabet. "a" becomes "d", "b" becomes "e" and so forth. The alphabet forms a ring such that "x" loops back around to "a", "y" to "b", etc.[1]. Another example using key = 1 is given (***Figure 2)***.

```
Key:            11111 11111
Plaintext:      hello world
                ----- -----
Ciphertext:     ifmmp xpsme
```

Figure 2: Using a key of 1, the phrase is encrypted by simple substitution

These ciphers can be broken using frequency analysis (Daniel Rodriguez-Clark, 2013). Since each character is shifted the same amount, the relative popularity of characters is preserved in the ciphertext. This is true for overall frequency and contextual frequency. Assuming an English alphabet encrypted with a Caesar cipher, "e" is statistically the most likely character of plaintext overall so "h" will tend to show up most frequently in the ciphertext. A cryptanalyst studying some ciphertext may assume the most popular ciphertext character translates to "e", the second most popular "t", third "a", and so on (Oxford Math Center, 2018). Context also matters. If position is preserved in the cipher text (i.e., spaces are not removed), then "t" is the most likely character to start a word (Oxford Math Center, 2018).

Also, if any ciphertext character is deciphered, then all other occurrences of that character are immediately known. In ***Figure 2***, the letter "m" appears three times in the ciphertext. If any one of these is determined to be "l" (lowercase L), then all three are instantly recognized.

[1] An online substitution cipher tool is available at https://learncryptography.com/tools/caesar-cipher

2.1.2. Polyalphabetic Cipher Primitives

Polyalphabetic ciphers like Vigenère improve upon simple substitution by using a multi-character key. Polyalphabetic ciphers can be thought of as several simple substitution ciphers used in rotation. The length of the key determines how many characters of plaintext can be encrypted before the key must be reused. In the following example, the key "sans" is highlighted in alternating colors to show the application of the key[2].

Keyword (sans):	sanssanssanssanssanssanssanssanssanssans
Plaintext:	the spy is driving a little red corvette
Ciphertext:	lhr kpl as vjiiafg s lvlllr jeq uoenwtgw

Figure 3: Example of polyalphabetic encryption with key "sans"

Table 1: Key encoding used in example

Key Encoding												
a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*
0	1	2	3	4	5	6	7	8	9	10	11	12
n	*o*	*p*	*q*	*r*	*s*	*t*	*u*	*v*	*w*	*x*	*y*	*z*
13	14	15	16	17	18	19	20	21	22	23	24	25

In contrast to the previous "hello world" example (***Figure 2)***, the plaintext in the next example is the same (***Figure 4***), but the five-character key is ***12345***. Therefore, each of the first five characters of plaintext is shifted independently according to the key for that column. Because the key is five characters in length, it must be reused after the first five characters of plaintext. In this example, the key is reused for "world" (***Figure 4***).

[2] In this example, spaces are not encrypted for simplicity

```
Key:          12345 12345
Plaintext:    hello world
              ----- -----
Ciphertext:   igopt xqupi
```

Figure 4: Using a key of 12345, the phrase is encrypted with polyalphabetic substitution

This method is more secure than the simple substitution cipher. In ***Figure 2***, all characters including the three occurrences of "l" (lowercase L) were shifted the same amount. The polyalphabetic key *12345* in ***Figure 4*** shifts the first occurrence of "l" (lowercase L) to "o" (lowercase O) and the second occurrence to "p". This behavior reduces the chance that popular letters will form highly predictable patterns in the ciphertext. However, it is important to note that the second "l" (lowercase L) in "hello" was shifted 4 spots to "p" exactly the same as the "l" (lowercase L) in "world". This is a result of both letters being in the same position relative to the key; the fourth position. This coincidence foreshadows the Vigenère cipher's demise.

2.1.3. One-time pad

Assuming the cryptanalyst has only the ciphertext, there exists an unbreakable cipher. If a random polyalphabetic key at least as long as the plaintext is used only once, a cryptanalyst cannot recover the plaintext[3] (RIJMENANTS, 2016). A cryptanalyst can discover some key that decrypts the ciphertext into an intelligible statement, but then find other keys that produce even more statements. There is no way to determine which message is correct since all are equally likely (RIJMENANTS, 2016).

It is theoretically possible to generate Vigenère cipher keys that fulfill the requirements of a one-time pad, but these may not be practical. Putting aside the issues of generating truly random keys, sharing keys securely and having enough key material, transporting the key itself becomes overwhelming. Recall the key must be at least as long as the plaintext. A mobile phone

[3] If the cryptanalyst is allowed other forms of attack such as coercion of an adversary or collusion with a dishonest agent, the assumption does not hold.

alone generates about 4GB of traffic per month (Heuveldop, 2017). Imagine the problems if this amount is doubled by the use of one-time pad.

2.2. Software and Test Files

To encrypt, decrypt and analyze data using a variety of ciphers, python[4] scripts are provided on GitHub in the "cryptography" repository[5]. The repository includes software needed to perform all operations and cryptanalysis demonstrated in this paper. Also, test files can be found in the "test-files" directory. The project includes scripts for many different ciphers and related utilities. Visionary.py and Freak.py are most applicable to this study.

2.2.1. Visionary.py

Script *visionary.py* encrypts/decrypts using a Vigenère cipher[6]. Unlike previous examples that were simplified for clarity, visionary.py works on all input bytes (i.e. a space character will be encrypted the same as any other). Character, binary and base64 input/output formats are supported. Input can be read from standard input[7] or a file. Important arguments include the following.

-h, --help	Show help
-e, --encrypt	Encrypt INPUT. This option requires a KEY.
-d, --decrypt	Decrypt INPUT. This option requires a KEY.
-k KEY, --key KEY	Encryption/Decryption key
-v, --verbose	Enables verbose output
-i INPUT_FILE, --input-file INPUT_FILE	Read INPUT from an input file

These are some examples of usage[8].

[4] Python 3 is required and available at https://www.python.org/downloads/
[5] https://github.com/webpwnized/cryptography
[6] The original cipher used only characters in the alphabet for keys. visionary.py operates on bytes so values 0 - 255 are possible.
[7] More information on standard input is available at http://www.linuxdevcenter.com/pub/a/linux/lpt/13_01.html
[8] Scripts were tested with Python 3.6.0 on Windows 10 version 1709, Python 3.6.4rc1 on Kali Linux 4.14.0 and Python 3.5.2 on Ubuntu 4.4.0

Encrypt the phrase "helloworld" with key 12345 (*Figure 5*)

python visionary.py --encrypt --key 12345 "helloworld"

```
C:\Users\Jeremy\Documents\GitHub\cryptography>python visionary.py
 --encrypt --key 12345 "helloworld"
igoptxqupi
```

Figure 5: Encrypt the phrase "helloworld" with key 12345

Decrypt the phrase "igoptxqupi" with key 12345 (*Figure 6*)

python visionary.py --decrypt --key 12345 "igoptxqupi"

```
C:\Users\Jeremy\Documents\GitHub\cryptography>python visionary.py
 --decrypt --key 12345 "igoptxqupi"
helloworld
```

Figure 6: Decrypt the phrase "igoptxqupi" with key 12345

Take input from binary file, encrypt, then redirect output to binary file (*Figure 7*)

python visionary.py --encrypt --key Password1 --input-format=binary --output-format=binary --input-file=funny-cat-1.jpg > enc-funny-cat-1.bin

```
C:\Users\Jeremy\Documents\GitHub\cryptography\test-files>python
../visionary.py --encrypt --key Password1 --input-format=binary
--output-format=binary --input-file=funny-cat-1.jpg > enc-funny-
cat-1.bin
```

Figure 7: Take input from binary file, encrypt, then redirect output to binary file

2.2.2. Freak.py

Freak.py is a utility that calculates statistics useful for analysis. The script supports character, binary and base64 input formats. Analysis is performed on each byte of input. Like visionary.py, input can be read from standard input[9] or a file and four types of analysis are supported: histogram, statistics, index of coincidence and polyalphabetic columns (columnar)[10].

A histogram is a graphical representation of the frequency of elements in a data set (Collins English Dictionary - Complete & Unabridged 10th Edition, 2010). Presented as a bar chart on a set of characters, the relative frequency of each character is denoted by the height of the bar.

The following options are helpful:

```
-c, --show-count          Show count for each byte of input
-p, --show-percent        Show percent representation for each byte of input
-m, --show-histogram Show histogram for each byte of input
-a, --show-ascii          Show ASCII representation for each byte of input
-t TOP_FREQUENCIES, --top-frequencies TOP_FREQUENCIES
                          Only display top X frequencies
```

Generally, the most common switches used with the histogram feature are count, percentage, show-histogram and top-frequencies. The following example counts the occurrences of each byte-value in a file and creates a histogram to display relative popularity. Byte-values range from 0 (0X00) to 255 (0XFF). In the example below, byte value 0 is the most frequent character by a large margin so the respective bar is relatively longer.

Considering each byte in file funny-cat-1.jpg, show count, percent and histogram for top 10 most common bytes (sorted by popularity) (*Figure 8*)

```
python freak.py -cpm -v -t 10 -i funny-cat-1.jpg
```

[9] More information on standard input is available at http://www.linuxdevcenter.com/pub/a/linux/lpt/13_01.html

[10] "Polyalphabetic columns" are discussed more once the relevance of the key length is established

```
C:\Users\Jeremy\Documents\GitHub\cryptography>python freak.py -cpm -v -t 10 -i test-files\funny-cat-1.jpg

Analysis of Input
Byte: 0         394     (1.03%) ####################
Byte: 204       220     (0.58%) ###########
Byte: 212       220     (0.58%) ###########
Byte: 50        217     (0.57%) ###########
Byte: 1         212     (0.56%) ###########
Byte: 16        211     (0.55%) ###########
Byte: 220       211     (0.55%) ###########
Byte: 89        200     (0.52%) ##########
Byte: 196       200     (0.52%) ##########
Byte: 44        196     (0.51%) ##########
```

Figure 8: Considering each byte in file funny-cat-1.jpg, show count, percent and histogram for top 10 most common bytes (sorted by popularity)

Calculating statistics is intuitive since the names of the arguments align directly with their function.

-mean, --show-mean	Show Arithmetic Mean (Average)
-median, --show-median	Show Median
-mode, --show-mode	Show Mode (Most popular byte)
-antimode, --show-anti-mode	Show Anti-Mode (Least popular byte)
-variance, --show-variance	Show Variance
-stddev, --show-standard-deviation	Show Standard Deviation
-e, --show-entropy	Show Shannon entropy
-stats, --show-statistics	Show mean, median, mode, variance, standard deviation and Shannon entropy for each byte of input

Generally, freak.py is used to compute all available statistics on a file. As with other operations, calculations are made with respect to byte-value. The script also calculates Shannon entropy with a range of 0.0 to 8.0. This value is a relative indicator of how random the data appears (Ueltschi, 2006). Data that is encrypted well should be reasonably indistinguishable from random data. A strong cipher produces an even histogram and very high entropy (Lawlor, 2013). The histogram, mode and entropy are correlated. For data to appear random, the popularity of

each byte must be the same (on average). This results in a flat histogram and high entropy (VandenBrink, 2016).

For each byte in file encrypted-funny-cat-1.jpg, show all statistics: mean, median, mode, anti-mode, variance, standard deviation, and Shannon entropy (*Figure 9*)

python freak.py --show-statistics --verbose -i funny-cat-1.jpg

```
C:\Users\Jeremy\Documents\GitHub\cryptography>python freak.py
--show-statistics --verbose -i test-files\funny-cat-1.jpg

Arithmetic Mean (Average): 123.53

Median (Middle element): 123

Mode (Most populous): 0 (Count: 394 Ratio: 2.64)

Anti-Mode (Least populous): 215 (Count: 95 Ratio: 0.64)

Variance: 5469.60

Standard Deviation: 73.96

Entropy: 7.98
```

Figure 9: For each byte in file encrypted-funny-cat-1.jpg, show all statistics: mean, median, mode, anti-mode, variance, standard deviation, and Shannon entropy

Index of Coincidence (IOC) indicates when bytes at a common offset are related. This statistical measure and related features of freak.py will be explained during the analysis of the key length. The IOC can be calculated using the -ioc, --show-ioc flag as seen in the following.

Determine the index of coincidence of file encrypted-funny-cat-1.jpg (*Figure 10*)

python freak.py -ioc --verbose --input-file=encrypted-funny-cat-1.bin

```
C:\Users\Jeremy\Documents\GitHub\cryptography>python freak.py -ioc -
-verbose --input-file=test-files\encrypted-funny-cat-1.jpg
Byte Offset: 01 [1.09] ####################
Byte Offset: 02 [1.06] ####################
Byte Offset: 03 [1.03] ###################
Byte Offset: 04 [0.94] #################
Byte Offset: 05 [1.00] ###################
Byte Offset: 06 [1.16] ######################
Byte Offset: 07 [0.87] ################
Byte Offset: 08 [1.87] ##################################
Byte Offset: 09 [0.96] ##################
Byte Offset: 10 [0.97] ##################
```

Figure 10: Determine the index of coincidence of file encrypted-funny-cat-1.jpg

2.2.3. Test Files

Four JPEG files are supplied for demonstration[11]. Three files are taken from photos of cats (***Figures 11-13***). The fourth is a bi-chromatic image of Twitter, Inc. logo Larry Bird[12] (***Figure 14***). The fourth image is provided as a counter-example. The cryptanalysis feature of freak.py does not perform well with images that are only 1-2 colors. Analysis will show the extreme lack of color variation throws off what is usually the most popular byte.

[11] https://github.com/webpwnized/cryptography/tree/master/test-files
[12] The small, blue bird ironically appears to have its name revealed in a tweet (https://twitter.com/rsarver/statuses/174205732776976385)

Figure 11: funny-cat-1.jpg

Figure 12: funny-cat-2.jpg

Figure 13: funny-cat-3.jpg

Figure 14: twitter-bird.jpg

An encrypted copy of funny-cat-1.jpg is included with the other test files. This file will be used in remaining demonstrations that require an encrypted file. The command used to encrypt was the following. The password is intentionally obfuscated to allow the reader to attempt to learn the password using techniques presented[13].

[13] The command with clear text password is available at https://github.com/webpwnized/cryptography#visionary

python visionary.py --encrypt --key [password]--input-format=binary --output-format=binary --input-file=test-files\funny-cat-1.jpg > test-files\encrypted-funny-cat-1.jpg

visionary.py encrypts by iterating over each byte of the plaintext. As it proceeds, plaintext bytes are shifted into ciphertext bytes. Because the cipher is polyalphabetic, the amount of shift varies from byte to byte according to the password. Each byte of the key is used in succession from left to right to determine the shift for the respective plaintext byte. When the key bytes are exhausted, the key bytes are reused starting at the left most key byte again (***Figure 15***).

```
def do_encrypt(pByte: int, pKey: int) -> int:
    #e(x) = (x + k) % n
    return (pByte + pKey) % MODULUS


def do_decrypt(pByte: int, pKey: int) -> int:
    #d(x) = (x - k) % n
    return (pByte - pKey) % MODULUS
```

Figure 15: Encryption function

To ensure the encryption process worked well, the file can be decrypted then compared to the original (***Figure 16***).

python visionary.py --decrypt --key [password] --input-format=binary --output-format=binary --input-file=test-files\encrypted-funny-cat-1.jpg > test-files\decrypted-funny-cat-1.jpg

fc /B test-files\funny-cat-1.jpg test-files\decrypted-funny-cat-1.jpg[14] [15]

[14] fc.exe is included with Windows. diff works well in Linux.
[15] /B compares file in binary mode

```
C:\Users\Jeremy\Documents\GitHub\cryptography>python visionary.py --
decrypt --key rocky329 --input-format=binary --output-format=binary
--input-file=test-files\encrypted-funny-cat-1.jpg > test-files\decry
pted-funny-cat-1.jpg

C:\Users\Jeremy\Documents\GitHub\cryptography>python visionary.py --
decrypt --key rocky329 --input-format=binary --output-format=binary
--input-file=test-files\encrypted-funny-cat-1.jpg > test-files\decry
pted-funny-cat-1.jpg

C:\Users\Jeremy\Documents\GitHub\cryptography>fc /B test-files\funny
-cat-1.jpg test-files\decrypted-funny-cat-1.jpg
Comparing files TEST-FILES\funny-cat-1.jpg and TEST-FILES\DECRYPTED-
FUNNY-CAT-1.JPG
FC: no differences encountered
```

Figure 16: To ensure the encryption process worked well, the file can be decrypted then compared to the original

2.3. A possible cryptanalysis of Vigenère cipher

2.3.1. An aside: Traffic Analysis

Before cryptanalysis can take place, reconnaissance is necessary to determine what has been encrypted. Context such as the original file format, the cipher used and other information is needed to attack the encryption (or perhaps to know a cryptanalysis attack is not likely to work). The communication patterns between systems can provide this background (Northcutt, 2007). Traffic Analysis seeks to answer questions such as "What encoding and encryption is usually used?", "What language is spoken?" and "What type of file is being transferred?". These and other conversational metrics may reveal the file type and encryption in play.

2.3.2. Potentially Useful Statistics: Unencrypted JPEG Files

JPEG is a binary format (Hamilton, 1992). Byte-values range from 0 (0X00) to 255 (0xFF). If JPEG files contain no distinguishing characteristics with respect to frequency of byte-values, the expected value of the mean and median is 127.5. All byte-values would be the mode[16] and the Shannon Entropy would be 8.0. A series of statistics can be run on the test files to see if

[16] Mode: Most popular value in a set

the byte values of JPEG files tend to have a prominent characteristic. freak.py is able to calculate mean, median, mode, standard deviation and entropy on the top 10 most popular bytes. The command is demonstrated here for file funny-cat-1.jpg and the same is run on the other three test files to generate the statistics. The results are summarized in ***Table 2***.

python freak.py -cpme -mean -median -mode -stddev -t 10 -v -i funny-cat-1.jpg

```
Analysis of Input
Byte: 0 394     (1.03%) ####################
Byte: 204       220     (0.58%) ###########
Byte: 212       220     (0.58%) ###########
Byte: 50        217     (0.57%) ###########
Byte: 1 212     (0.56%) ###########
Byte: 16        211     (0.55%) ###########
Byte: 220       211     (0.55%) ###########
Byte: 89        200     (0.52%) ##########
Byte: 196       200     (0.52%) ##########
Byte: 44        196     (0.51%) ##########

Arithmetic Mean (Average): 123.53

Median (Middle element): 123

Mode (Most populus): 0

Standard Deviation: 73.96

Entropy: 7.98
```

Figure 17: Statistics for funny-cat-1.jpg

python freak.py -cpme -mean -median -mode -stddev -t 10 -v -i funny-cat-2.jpg

```
Analysis of Input
Byte: 0 88      (1.76%) ###################################
Byte: 45        62      (1.24%) ########################
Byte: 21        39      (0.78%) ###############
Byte: 1 37      (0.74%) ##############
Byte: 26        36      (0.72%) ##############
Byte: 43        36      (0.72%) ##############
Byte: 16        33      (0.66%) #############
Byte: 114       33      (0.66%) #############
Byte: 49        32      (0.64%) ############
Byte: 51        32      (0.64%) ############

Arithmetic Mean (Average): 119.68

Median (Middle element): 117.5

Mode (Most populus): 0

Standard Deviation: 75.11

Entropy: 7.91
```

Figure 18: Statistics for funny-cat-2.jpg

python freak.py -cpme -mean -median -mode -stddev -t 10 -v -i funny-cat-3.jpg

```
Analysis of Input
Byte: 0 96      (1.60%) ################################
Byte: 45        63      (1.05%) ####################
Byte: 1 51      (0.85%) #################
Byte: 46        42      (0.70%) ##############
Byte: 23        40      (0.67%) #############
Byte: 146       40      (0.67%) #############
Byte: 150       40      (0.67%) #############
Byte: 92        39      (0.65%) #############
Byte: 2 38      (0.63%) ############
Byte: 70        38      (0.63%) ############

Arithmetic Mean (Average): 121.92

Median (Middle element): 120.5

Mode (Most populus): 0

Standard Deviation: 74.81

Entropy: 7.93
```

Jeremy Druin, jdruin@gmail.com

Figure 19: Statistics for funny-cat-3.jpg

python freak.py -cpme -mean -median -mode -stddev -t 10 -v -i twitter-bird.jpg

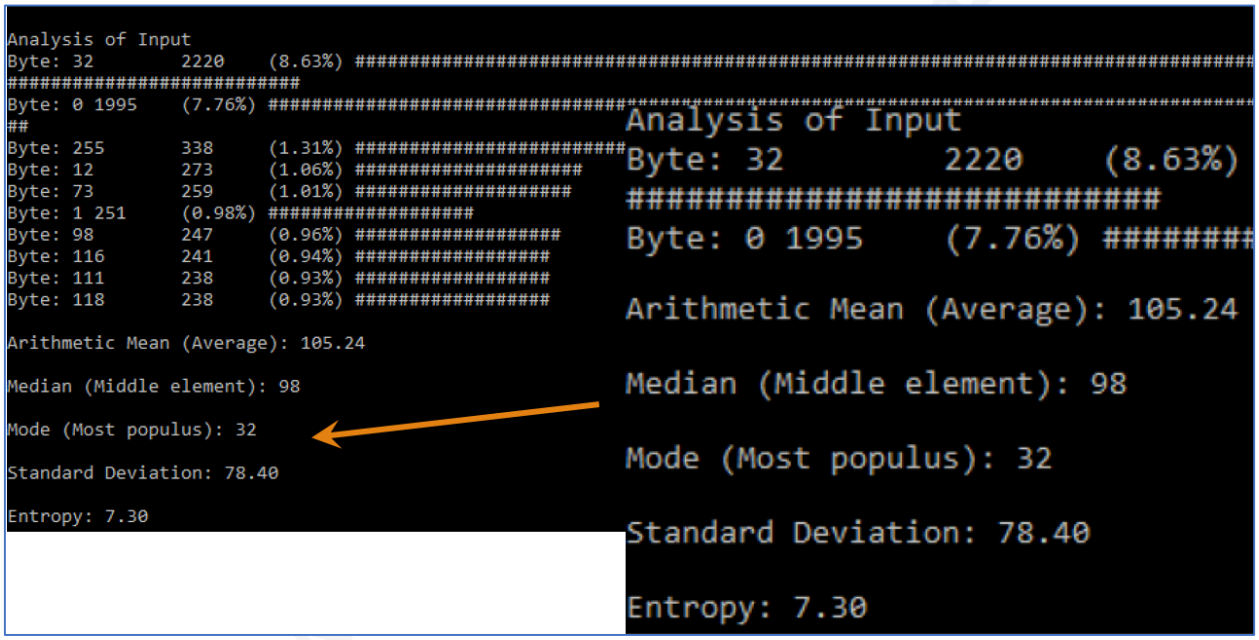

Figure 20: Statistics for twitter-bird.jpg

Table 2: Statistics for test files

	Funny Cat 1	Funny Cat 2	Funny Cat 3	Larry Bird
Mean	123.53	119.68	121.92	105.24
Median	123	117.5	120.5	98
Mode	0	0	0	32
Standard Deviation	73.96	75.11	74.81	78.40
Entropy	7.98	7.91	7.93	7.30

The figures show a byte-value of 0 is pronounced in all of the JPEG except Larry Bird. However, even in this counter-example, byte-value 0 makes up 7.76% of the file. This is well

above the 0.39% expected if there were no prominent byte-values[17]. Also, even though byte-value 0X32 was the mode at 8.63%, byte-value 0 was a close second (***Figure 20***).

2.3.3. Potentially Useful Statistics: JPEG Files Encrypted with Vigenère

The same statistics can be run on the encrypted version of funny-cat-1.jpg, then the results compared against those in ***Figure 17*** and ***Table 2 - Funny Cat 1***. The command to calculate the data is the following.

```
python freak.py -cpme -mean -median -mode -stddev -t 10 -v -i encrypted-funny-cat-1.jpg
```

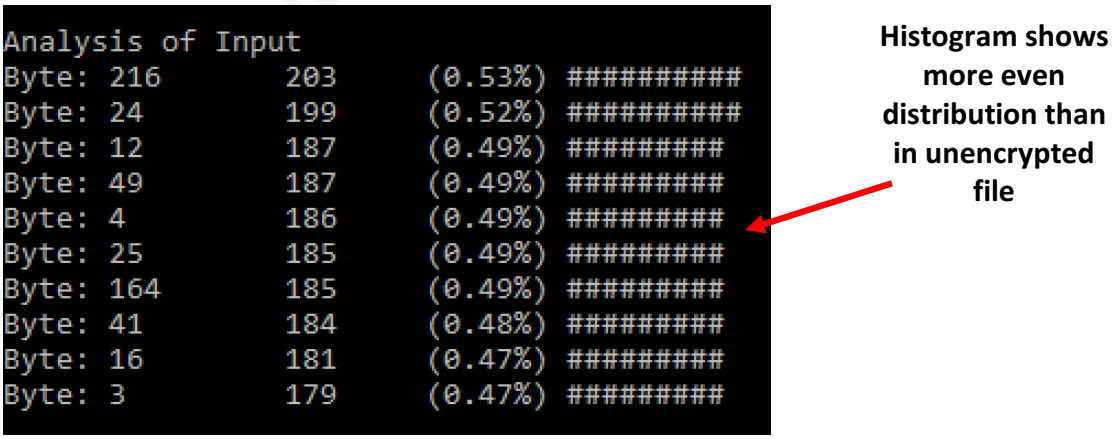

Figure 21: Statistics of encrypted-funny-cat-1.jpg

[17] Assuming all bytes are equally likely, any randomly chosen byte should be present about 1 / 256 * 100 = 0.39% of the time.

	funny-cat-1.jpg	encrypted-funny-cat-1.jpg
Mean	123.53	125.14
Median	123	124
Mode	0	216
Mode Count (Ratio)	394 (2.64)	203 (1.36)
Anti-Mode	215 } 2.00	188 } 0.57
Anti-Mode (Ratio)	95 (0.64)	117 (0.79)
Standard Deviation	73.96	73.96
Entropy	7.98	7.99

Figure 22: Comparison between encrypted and unencrypted JPEG (Vigenère encryption)

Vigenère has suppressed some of the identifying characteristics of the JPEG file, but the mean and median are still off the expected value of is 127.5 for a well encrypted JPEG. Vigenère flattened the histogram by spreading out the popular bytes more evenly over the range. This is also observed in the difference in the count of bytes that make up the mode and anti-mode[18]. The mode count ratio is the number of bytes for the mode divided by the number of bytes expected for a random distribution. For funny-cat-1.jpg, the mode count ratio is 2.64 which says there are more than 2.5 times as many "0" than expected if the distribution of byte-values were perfectly even. Vigenère reduces the magnitude of the mode and the anti-mode as it distributes these extremes over the rest of the range. As a result, the difference between mode and anti-mode ratios shrinks from 2.00 to 0.57. However, Vigenère does not transform the JPEG file into one that appears to be made of random bytes. There is still a prominent mode suggesting the Vigenère cipher can be broken.

2.4. Cryptanalysis of Vigenère-encrypted JPEG files

[18] Anti-mode: Least popular byte

Recall a Vigenère cipher is effectively a series of substitution ciphers. Using a key length of 5 as an example (***Figure 23***), the first byte of the key is used to encrypt both plaintext character 1 and 6 because the key must be reused. This pattern holds for the other bytes in the key. In ***Figure 24***, the ciphertext is arranged so characters encrypted with the same key byte are aligned in columns. Each of these columns is a substitution cipher encrypted by the respective key byte a.k.a. the "column key".

```
Key:          12345 12345
Plaintext:    hello world
              ----- -----
Ciphertext:   igopt xqupi
```

Figure 23: Example encryption with Vigenère

```
Key:          12345
              -----
Ciphertext:   igopt
              xqupi
```

Figure 24: Ciphertext arranged into columns under the key byte used to encrypt those respective characters

Substitution ciphers can be broken using frequency analysis (Lyons, 2012). In this technique, the most popular character in the ciphertext is identified. If the mode of the ciphertext is calculated, the value is probably the most popular letter in the plaintext encrypted with the key. Since the encrypted byte and the plaintext byte are known, the key can be recovered by adjusting the ciphertext by the value of the plaintext. The result will be the key most of the time.

Ciphertext created by Vigenère ciphers can be rearranged so that the columns form substitution ciphers (***Figure 24***). For any given column of ciphertext, the mode can be found. Statistical analysis showed JPEG files tend to have byte 0 as mode in the plaintext. Therefore,

the mode of the ciphertext is likely the key for that column[19]. However, in order to arrange the ciphertext bytes into columns with respect to the key byte that encrypted that column, the key-length must be known.

2.4.1. Index of Coincidence

The value of a character of ciphertext is determined by two inputs: the value of the plaintext and the column key. For example, the first character of ciphertext in ***Figure 25*** is "i" because the first plaintext character "h" was shifted one letter by the key "1". The last character of ciphertext is also "i" but was generated when plaintext character "d" was shifted 5 spots by key "5".

```
Key:          12345 12345
Plaintext:    hello world
              ----- -----
Ciphertext:   igopt xqupi
```

Figure 25: Example encryption with Vigenère

Note there are also two characters with value "p" in the ciphertext (***Figure 25***). These characters were both generated from plaintext character "l" (lowercase L) shifted 4 positions. In the former case of ciphertext letter "i", the key value was different but, in this latter, the key value was the same. The probability of getting these "doubles" is greatly influenced if the key used to encrypt the characters is identical.

Some plaintext letters are more popular than others. For example, the chance of randomly choosing the letter "e" out of a text is about 12% (Cornell University, 2004). Without encryption, the chance of choosing two letter "e" is 1.44%[20]. When encrypted with Vigenère, those two letter "e" would not necessarily be encrypted with the same key. This would only happen if the key value in the respective key positions happened to be the same. The chances of hitting doubles is

[19] Normally the column key is calculated by subtracting the mode of the plaintext from the ciphertext, but the mode of the plaintext happens to be 0, so no adjustment is needed.
[20] 12% * 12%

lessened since popular letters only have about a 1/(key-length) chance of being encrypted by the same value[21].

When the ciphertext characters are in the same position with respect to the key, the probability of the key is not relevant. Plaintext letters from the same "column" (***Figure 24***) are guaranteed to be shifted the same amount. To get doubles for two ciphertext characters from the same column, it only matters that the same plaintext letters were encrypted. Popular letters regain their influence. Getting doubles is easier when observing two ciphertext characters from the same column (***Figure 26***).

Figure 26: Ciphertext arranged into columns under the key byte used to encrypt those respective characters

Imagine ciphertext is arranged in columns such that all characters in the respective column are encrypted with the same key-value (***Figure 26***). The chance of drawing doubles from two different columns is about 0.038 (3.8%)[22]. However, if values are picked within the same column, the chance is almost twice as good at 0.065 (6.5%) (Christensen, 2015). If we check for doubles every other character, then every third character, then every fourth character, etc., the chance of getting doubles will spike once the spacing or "index" is aligned with the key length. This *index of coincidence (IOC)* is calculated with the following formula (***Figure 27***).

[21] "About" because the key itself might have duplicate letters
[22] Presuming a "sufficiently long key" or "as the key length goes to infinity"

$$\mathbf{IC} = \frac{\sum_{j=1}^{N}[a_j = b_j]}{N/c},$$

Figure 27: Formula for Index of Coincidence (Wikipedia, 2017)

One method to determine IOC is to make a copy of the ciphertext under itself (***Figure 28***). The ciphertext is shifted one position and the index of coincidence is calculated. Then the IOC is recalculated after shifting two positions. This process continues until number of shifts exceeds the length of the key. An educated guess of key length can be made from traffic analysis or reconnaissance. Alternatively, since computers can automate the calculations, brute-force of several dozen shifts could provide enough coverage[23]. There is no penalty for trying too many shifts. The value of the IOC spikes each time the index is a multiple of the key length.

Figure 28: Checking for doubles after shifting ciphertext 1,2,3,... positions

For the phrase "hello world" encrypted with key 12345, the IOC would tend to be maximal when the amount of shift was a multiple of 5. The freak.py script can be used to examine encrypted-funny-cat-1.jpg.

```
python freak.py -ioc -i encrypted-funny-cat-1.jpg
```

[23] freak.py has MAX_SHIFTS_TO_ANALYZE = 50 by default. This constant can be changed in function get_kappa_index_of_coincidence().

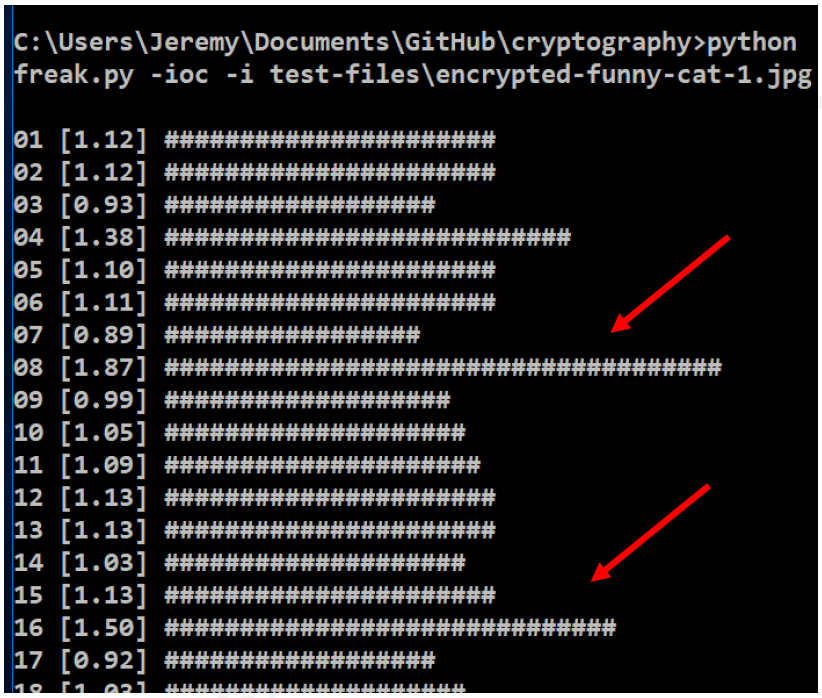

Figure 29: IOC calculation for shifts 1 - 49

For funny-cat-1, spikes in the value of IOC found at shifts 8,16, 24, etc. indicate the key length is 8 (***Figure 29***).

2.4.2. Key recovery

Since the key length is known, the ciphertext can be broken into blocks of 8 characters so that characters encrypted with the same key byte are aligned in columns (***Figure 30***).

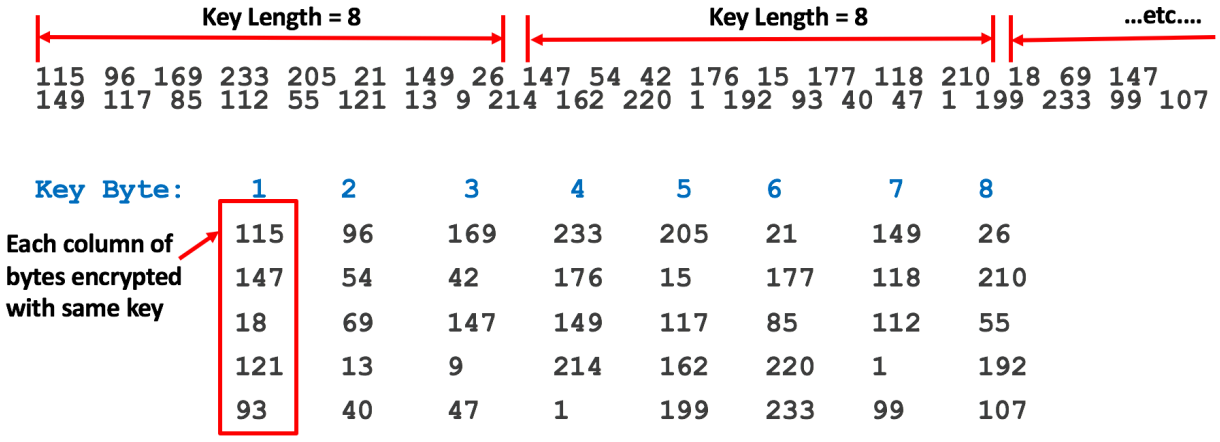

Figure 30: Ciphertext from encrypted-funny-cat-1.jpg in rows of 8 bytes

The key byte for each column can be recovered using techniques prescribed for a substitution cipher. Prior analysis shows the most popular byte in the plaintext probably has a value of zero. Therefore, the mode of each column should be the value of the key byte[24]. The entire key is recovered by repeating this process for each column. The "-g" causes freak.py to guess the key bytes. "-col 8" provides the length of the key (***Figure 31***).

python freak.py -c -g -t 1 -col 8 --input-file=encrypted-funny-cat-1.jpg

```
C:\Users\Jeremy\Documents\GitHub\cryptography>python freak.py -c
-g -t 1 -col 8 --input-file=test-files\encrypted-funny-cat-1.jpg
Analysis of Column 1

Analysis of Input

Best guess      Lowercase: r    Uppercase: R    Numeric: A
17      58

Analysis of Column 2

Analysis of Input

Best guess      Lowercase: o    Uppercase: O    Numeric: >
14      36

Analysis of Column 3

Analysis of Input

Best guess      Lowercase: c    Uppercase: C    Numeric: 2
2       45

Analysis of Column 4
```

Figure 31: Output of "guess" feature of freak.py

[24] Referring to ***Figure 15***, the formula to encrypt the plaintext byte is "value of plaintext byte" + "value of key byte". Since the "value of plaintext byte" is 0 for the most popular bytes, the most popular ciphertext bytes are probably the value of the key.

```
Best guess      Lowercase: r    Uppercase: R    Numeric: A
Best guess      Lowercase: o    Uppercase: O    Numeric: >
Best guess      Lowercase: c    Uppercase: C    Numeric: 2
Best guess      Lowercase: k    Uppercase: K    Numeric: :
Best guess      Lowercase: y    Uppercase: Y    Numeric: H
Best guess      Lowercase: d    Uppercase: D    Numeric: 3
Best guess      Lowercase: c    Uppercase: C    Numeric: 2
Best guess      Lowercase: j    Uppercase: J    Numeric: 9
```

Figure 32: Output of freak.py "guess" with extraneous lines removed

Because visionary.py normalizes the key bytes before encryption, information about whether the key phrase was uppercase, lowercase or numeric is lost[25]. freak.py displays all three possibilities (***Figure 32***). "Rocky329", "ROCKY329" and "rocky329" are reasonable guesses. Trying "rocky329" successfully decrypts the plaintext[26].

2.5. Opportunities for Improvement

JPEG files were specifically chosen for demonstration because the pronounced mode makes for easier analysis. To cryptanalyze English text, frequency analysis can be used (K.W. Lee, 2006) but freak.py does not implement these techniques. A modular implementation that uses character frequencies based on the language would allow languages besides English to be supported.

The image of the Twitter, Inc. logo showed that redundant color schemes can cause the mode of a JPEG file to be a value other than 0x00. freak.py might be improved in this respect in a couple of ways. If the script used the top three most popular bytes, the user could see which outputs reasonable guesses. Alternatively, the user could be allowed to enter which byte they prefer based on the results of the statistical analysis. Finally, the program could run the statistical analysis itself then automatically use the mode found.

[25] Reference function do_derive_key() in visionary.py
[26] For implementations of Vigenère that do not normalize the key, freak.py would simply output the password. Even though visionary.py makes analysis more difficult, the author stayed with the implementation because it mimics a cipher encountered in "real life".

Other tools already possess some enhanced features. FeatherDuster[27] by Daniel Crowley can automate much of this process for multiple types of ciphers (Crowley, 2016). The python-based tool can detect weak ciphers and improper use of cryptographic mechanisms. It includes code breaking features. FeatherDuster can analyze ciphertext, recommend an attack module or try all attacks with "autopwn"[28].

3. Conclusion

Vigenère cipher makes a good starting point to learn about cryptography and cryptanalysis. Vigenère gives an opportunity to explore substitution ciphers, polyalphabetic ciphers and multiple statistical techniques. The study shows some components of cryptanalysis: traffic analysis, identification of encrypted data, an introduction to applying statistics and an opportunity to write helpful scripts. Vigenère is a classical cipher. Defeating the cipher is not a gateway into professional code-breaking but gives the amateur insight into the subject of cryptanalysis.

For security professionals who do not specialize in cryptography, an introduction to cryptanalysis provides an appreciation for the important subject and highlights a valuable lesson. Cryptanalysis can be difficult even for a centuries-old, obsolete cipher. It is that much harder for modern ciphers. Failure to envision a way to break a cipher does not prove the cipher is invincible (Schneier, 2011). To have confidence in an encryption system, it is important to use proven ciphers that have been targeted by multiple, competent cryptanalysts over a significant period of time.

[27] https://github.com/nccgroup/featherduster

[28] According to http://schd.ws/hosted_files/issw2016/a6/Cryptanalib_and_FeatherDuster.pdf, a hat tip to Metasploit.

Jeremy Druin, jdruin@gmail.com

References

Bellaso, G.-B. (1552, 12 22). *La cifra nuovamente da lui ridotta a grandissima brevita etc.* Retrieved from Google Play Books: https://play.google.com/store/books/details/Giovan_Battista_Bellaso_La_cifra_nuovamente_da_lui?id=GbZRAAAAcAAJ

Christensen, C. (2015). *Cryptanalysis of the Vigenère Cipher: The Friedman Test.* Retrieved from NKU: https://www.nku.edu/~christensen/1402%20Friedman%20test%202.pdf

Collins English Dictionary - Complete & Unabridged 10th Edition. (2010). *histogram*. Retrieved from Dictionary.com: http://www.dictionary.com/browse/histogram

Collins English Dictionary. (2012). *Cryptanlysis*. Retrieved from Collins English Dictionary - Complete & Unabridged 10th Edition: http://www.dictionary.com/browse/cryptanalysis

Cornell University. (2004). *English Letter Frequency (based on a sample of 40,000 words).* Retrieved from Cornell University: https://www.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html

Crowley, D. ". (2016). *Cleaning up Magical Crypto Fairy Dust with Cryptanalib and FeatherDuster.* Retrieved from SCHED: http://schd.ws/hosted_files/issw2016/a6/Cryptanalib_and_FeatherDuster.pdf

Crypto Museum. (2015, 8 28). *One-Time Pad*. Retrieved from Crypto Museum: http://cryptomuseum.com/crypto/otp/index.htm

Daniel Rodriguez-Clark. (2013). *Frequency Analysis.* Retrieved from Crypto Corner: http://crypto.interactive-maths.com/frequency-analysis-breaking-the-code.html

Forouzan, B. A. (2007, 2 28). *Data Encryption Standard (DES).* Retrieved from Cryptography and Network Security: http://highered.mheducation.com/sites/dl/free/007070208x/877405/Chapter_06_Data_Encription_Standard.pdf

Gargiulo, J. (2002, 7 25). *S-Box Modifications and Their Effect in DES-like Encryption Systems.* Retrieved from SANS Institute InfoSec Reading Room: https://www.sans.org/reading-room/whitepapers/vpns/s-box-modifications-effect-des-like-encryption-systems-768

Hamilton, E. (1992, 9 1). *JPEG File Interchange Format.* Retrieved from W3.org: https://www.w3.org/Graphics/JPEG/jfif3.pdf

Heuveldop, N. (2017, 6 1). *Ericsson Mobility Report.* Retrieved from Ericsson: https://www.ericsson.com/assets/local/mobility-report/documents/2017/ericsson-mobility-report-june-2017.pdf

Jeremy Norman. (2017, 10 22). *Blaise de Vigenère Describes What is Later Known as the Vigenère Cipher*. Retrieved from HistoryofInformation.com: http://www.historyofinformation.com/expanded.php?id=2011

K.W. Lee, C. T. (2006). *Semantic Scholar.* Retrieved from DECRYPTING ENGLISH TEXT USING ENHANCED FREQUENCY ANALYSIS: https://pdfs.semanticscholar.org/5013/213db36162f6e5ae6d2f1c72fe8240adb567.pdf

Kahn, D. (1967). *The Codebreakers.* New York: Macmillian.

Lawlor, D. O. (2013). *Information Theory, Entropy, and Key Strength*. Retrieved from CS 463 Lecture: https://www.cs.uaf.edu/2013/spring/cs463/lecture/01_25_entropy.html

Jeremy Druin, jdruin@gmail.com

Lyons, J. (2012). *Simple Substitution Cipher* . Retrieved from Practical Cryptography: http://practicalcryptography.com/ciphers/simple-substitution-cipher/

Math Explorers Club. (2003, 3 8). *Polyalphabetic Substitution Ciphers*. Retrieved from Cornell University: https://www.math.cornell.edu/~mec/2003-2004/cryptography/polyalpha/polyalpha.html

Merriam-Webster. (2017, 12 3). *Cryptography*. Retrieved from Merriam-Webster: https://www.merriam-webster.com/dictionary/cryptography

Northcutt, S. (2007, 5 16). *Traffic Analysis*. Retrieved from SANS: https://www.sans.edu/cyber-research/security-laboratory/article/traffic-analysis

NOVER, H. (2004). *ALGEBRAIC CRYPTANALYSIS OF AES: AN OVERVIEW.* Retrieved from semanticscholar.org: https://pdfs.semanticscholar.org/9924/59258de4ff113e84e1813a53e3bad96fcc1a.pdf

Oxford Math Center. (2018). *Letter Frequencies in English*. Retrieved from Oxford Math Center: http://www.oxfordmathcenter.com/drupal7/node/353

Rijmenants, D. (2016). *CIPHER MACHINES AND CRYPTOLOGY*. Retrieved from Technical Details of the Enigma Machine: http://users.telenet.be/d.rijmenants/en/enigmatech.htm#plugboard

Rijmenants, D. (2016). *Technical Details of the Enigma Machine*. Retrieved from CIPHER MACHINES AND CRYPTOLOGY: http://users.telenet.be/d.rijmenants/en/enigmatech.htm#rotorencryption

RIJMENANTS, D. (2016, 1 22). *THE COMPLETE GUIDE TO SECURE COMMUNICATIONS WITH THE ONE TIME PAD CIPHER.* Retrieved from users.telenet.be: http://users.telenet.be/d.rijmenants/papers/one_time_pad.pdf

Salomon, D. (2003). *Data Privacy and Security: Encryption and Information Hiding.* Springer Science & Business Media.

Schneier, B. (2011, 4 15). *"Schneier's Law"*. Retrieved from Schneier on Security: https://www.schneier.com/blog/archives/2011/04/schneiers_law.html

Simmons, G. J. (2009, 6 22). *Substitution cipher*. Retrieved from Encyclopedia Britannica: https://www.britannica.com/topic/substitution-cipher

Simmons, G. J. (2009, 7 22). *Vigenère cipher*. Retrieved from Encyclopædia Britannica: https://www.britannica.com/topic/Vigenere-cipher

Singh, S. (2001). *The Code Book.* New York, New York: Delacorte Press.

The Oxford Math Center. (2018). *The Vigenere Cipher*. Retrieved from The Oxford Math Center: http://www.oxfordmathcenter.com/drupal7/node/169

Ueltschi, D. (2006). *Introduction to Statistical Mechanics*. Retrieved from http://www.ueltschi.org: http://www.ueltschi.org/teaching/chapShannon.pdf

University of Babylon. (2011). *Vigenère cipher.* Retrieved from University of Babylon: http://www.uobabylon.edu.iq/eprints/paper_11_1363_649.pdf

University of Rhode Island. (2013). *Classical Cryptography.* Retrieved from University of Rhode Island: https://www.cs.uri.edu/cryptography/classicalsubstitution.htm

University of Rhode Island. (2013). *Classical Cryptography.* Retrieved from University of Rhode Island: https://www.cs.uri.edu/cryptography/classicalvigenere.htm

Jeremy Druin, jdruin@gmail.com

VandenBrink, R. (2016, 8 8). *SANS ISC InfoSec Forums*. Retrieved from Internet Storm Center: https://isc.sans.edu/forums/diary/Using+File+Entropy+to+Identify+Ransomwared+Files/21351/

Wikipedia. (2017, 7 7). *Index of coincidence.* Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Index_of_coincidence

Jeremy Druin, jdruin@gmail.com

Last Updated: April 18th, 2018

Upcoming Training

Baltimore Spring 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Baltimore, MD	Apr 23, 2018 - Apr 28, 2018	vLive
SANS Seattle Spring 2018	Seattle, WA	Apr 23, 2018 - Apr 28, 2018	Live Event
Mentor Session - AW SEC504	Alexandria, VA	Apr 27, 2018 - May 04, 2018	Mentor
SANS Riyadh April 2018	Riyadh, Saudi Arabia	Apr 28, 2018 - May 03, 2018	Live Event
Community SANS Toronto SEC504	Toronto, ON	Apr 30, 2018 - May 05, 2018	Community SANS
Automotive Cybersecurity Summit & Training 2018	Chicago, IL	May 01, 2018 - May 08, 2018	Live Event
SANS SEC504 in Thai 2018	Bangkok, Thailand	May 07, 2018 - May 12, 2018	Live Event
SANS Security West 2018	San Diego, CA	May 11, 2018 - May 18, 2018	Live Event
Community SANS Columbia SEC504	Columbia, MD	May 14, 2018 - May 19, 2018	Community SANS
SANS Melbourne 2018	Melbourne, Australia	May 14, 2018 - May 26, 2018	Live Event
Mentor Session - AW SEC504	Chicago, IL	May 16, 2018 - May 23, 2018	Mentor
SANS Northern VA Reston Spring 2018	Reston, VA	May 20, 2018 - May 25, 2018	Live Event
Community SANS Phoenix SEC504	Phoenix, AZ	May 21, 2018 - May 26, 2018	Community SANS
Mentor Session - SEC504	Dulles, VA	May 24, 2018 - Jun 28, 2018	Mentor
SANS Amsterdam May 2018	Amsterdam, Netherlands	May 28, 2018 - Jun 02, 2018	Live Event
SANS Atlanta 2018	Atlanta, GA	May 29, 2018 - Jun 03, 2018	Live Event
SANS London June 2018	London, United Kingdom	Jun 04, 2018 - Jun 12, 2018	Live Event
SANS Rocky Mountain 2018	Denver, CO	Jun 04, 2018 - Jun 09, 2018	Live Event
Mentor Session - SEC504	Milwaukee, WI	Jun 06, 2018 - Jul 25, 2018	Mentor
Mentor Session - SEC504	San Francisco, CA	Jun 06, 2018 - Aug 01, 2018	Mentor
SANS Oslo June 2018	Oslo, Norway	Jun 18, 2018 - Jun 23, 2018	Live Event
SANS Cyber Defence Japan 2018	Tokyo, Japan	Jun 18, 2018 - Jun 30, 2018	Live Event
SANS Crystal City 2018	Arlington, VA	Jun 18, 2018 - Jun 23, 2018	Live Event
SANS Philippines 2018	Manila, Philippines	Jun 18, 2018 - Jun 23, 2018	Live Event
SANS Minneapolis 2018	Minneapolis, MN	Jun 25, 2018 - Jun 30, 2018	Live Event
SANS Vancouver 2018	Vancouver, BC	Jun 25, 2018 - Jun 30, 2018	Live Event
SANS Cyber Defence Canberra 2018	Canberra, Australia	Jun 25, 2018 - Jul 07, 2018	Live Event
Minneapolis 2018 - SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling	Minneapolis, MN	Jun 25, 2018 - Jun 30, 2018	vLive
SANS Paris June 2018	Paris, France	Jun 25, 2018 - Jun 30, 2018	Live Event
SANS London July 2018	London, United Kingdom	Jul 02, 2018 - Jul 07, 2018	Live Event
Community SANS Madrid SEC504 (in Spanish)	Madrid, Spain	Jul 02, 2018 - Jul 07, 2018	Community SANS