AES-128 Encryption Validity and Statistical Analysis (December 2017)

Laurie Harris, Tim McWilliams, and Jack Nelson, *Data Science Graduate Students, Southern Methodist University*

***Abstract* — Significant advances in computing power have taken place since the original implementation of the Advanced Encryption Standard (AES)-128 bit algorithm in 2001. Randomness through both diffusion and confusion play an integral factor in its success in addition to its avalanche properties, the characteristic that making minor input changes compounds on one another resulting in a very different ciphertext result. The assurance that the AES-128 encryption algorithm is still relatively secure with the current technological landscape would continue to show its robustness after over a decade and a half of implementation. No different than any algorithm, regularly updating keys and keeping keys in use concealed is an imperative factor in secure communications. Compromising a key's confidentiality adversely affects the effectiveness of the AES-128 encryption, leaving the door open for an attacker to use cryptanalysis to intercept and decipher confidential material.**

A statistical analysis was conducted on the validity of the Advanced Encryption Standard (AES)-128 algorithm on blocks of plaintext using randomness tests in the test suite defined by the National Institute of Standards and Technology (NIST). Python's Jupyter Notebook application was leveraged to carry out the results of this paper. More specifically, the PyCrypto package generated ciphertext blocks acting as our inputs to the analysis. Two scenarios with respect to the manner 128-bit plaintext blocks and 128-bit keys are generated and used uncover a potential non-random tendency when the AES-128 algorithm is used in Electronic Codebook mode (ECB).

***Index Terms*—AES-128, block cipher, cryptanalysis, cryptography, deciphering, encryption, NIST, randomness, python.**

I. Introduction

Encrpytion acts as a linchpin in upholding security and secrecy as it pertains to message transmissions that occur over the internet or any other telecom network. Concealing information regarding bank transactions, medical records, or government agency informants only scratch the surface of sensitive information that can result in significant damage if placed into the wrong hands.

The raft of potential repercussions from the integrity of a compromised encryption algorithm poses a need for continually validating its authenticity. Validating involves both the confirmation that the algorithm produces random ciphertext fending off cryptanalysis and all possible keys are ambiguous enough where brute-force attacks are unrealistic.

The widely used AES-128 algorithm is regarded as a secure cipher against both cryptanalysis and brute-force. It undergoes a series of transformations after its inputs, both keys and plaintext, are structured into 4x4 byte matrices. These transformations are conducted several times in ten separate rounds of encryption where an original 128-bit key is used to generate eleven round keys to be used in separate encryption rounds. The key generation process is called Key Expansion or Key Expansion Operation. Encryption rounds are composed of the following transformations: substitute bytes, shift rows, mix columns, and add round key [1]. Later sections in the paper explain these transformations as well as the Key Expansion Operation in detail.

A series of randomness tests from the National Institute of Standards and Technology (NIST) test package grade the performance of the AES-128 encryption ciphertext blocks. The 15 recommended tests, which will be described in detail, will examine randomness in frequency, frequency within a block, runs on uninterrupted sequences, run length in a block, and other linear cumulative sum, excursion and complexity testing.

This paper follows a series of steps to validate the AES-128 encryption algorithm from a cryptanalysis standpoint. It is assumed the 128-bit key size used in the cipher is secure against brute-force. [2] After explaining the AES-128 encryption design in greater depth in Section II, a detailed description of the randomness tests used for the analysis defined by NIST takes place in Section III. A brief summary of related work uncovered during research is done in Section IV. Section V compromises the scenarios used in the analysis and the methods carried out to generate 1,000 128-byte ciphertext blocks. The analysis results and the interpretation of the results that were uncovered are explained in Sections VI and VII, respectively. Finally, Section VIII will provide key conclusions made from the findings and suggest potential future research that can build upon the results of this AES-128 cipher analysis.

This paper, submitted for review December 2017, underwent intermittent milestone reviews over the course of the fall 2017 semester. The work is part of the requirements for MSDS 7349: Data and Network Security, a course within the online Master of Data Science program at Southern Methodist University.

L. L. Harris is a graduate student at Southern Methodist University, Mansfield, TX 76063 USA (e-mail: llharis@smu.edu).

T. M. McWilliams is a graduate student at Southern Methodist University, Williamstown, NJ 08094 USA (e-mail: tmcwilliams@smu.edu).

J. A. Nelson is a graduate student at Southern Methodist University, Royal Oak, MI 48067 USA (e-mail: nelsonjohn@mail.smu.edu).

II. AES-128 Encryption Design

AES was introduced by NIST in 1997 as a solution to increased attacks on its predecessor, Data Encryption Standard (DES). In 2000, following a competitive process, the Rijndael cipher was selected as the proposed algorithm for AES. The Rijndael cipher was among five algorithms which were chosen as finalists for AES. Randomness of the output was such a strong consideration that NIST researchers tested such and described the results in a 2000 publication by Juan Soto and Lawrence Bassham. The researchers tested randomness for each round of the five algorithms (Mars, RC6, Serpent, Twofish and Rijndael) and concluded that all algorithms appeared to have no detectable deviations from randomness. [3]

After successful use by the U.S. Government, AES became widely used in the private sector and it is regarded as a secure cipher, resistant to both brute-force attacks and cryptanalysis. [4] Using the AES-128 cipher, messages are encrypted in 128-bit blocks with ten rounds of substitution and transposition to produce secure ciphertext.

The AES-128 algorithm is composed of ten rounds of encryption. Within each round, a plaintext transforms through using the corresponding cipher key. It is important to note that this is not a Feistel structure. The AES processes the entire block as a single matrix using substitutions and permutations for each of the ten rounds. [2] The key and the state, also called the input data, structure into a 4x4 matrix of bytes. [1] There are four distinct words that represent 128 bits. These four distinct words serve as a round key for each of the ten rounds.

Four different transformations are used within each round. One function is a permutation while the remaining three are substitutions. [2] When encrypting AES uses forward transformations. When decrypting AES uses inversed transformations, except the add row key transformation. This is because "the inverse add round key transformation is identical to the forward add round key transformation where the XOR operation is its own inverse." [2]

Substitute bytes (forward and inverse): This transformation uses an S-box to perform a byte-by-byte substitution of the block. The S-box is a 16 x 16 matrix of bytes that contains a permutation of all possible 256 8-bit values. [2]

Shift rows (forward and inverse): In this transformation, the first row of the state is not altered. However, a 1-byte circular left shift is performed on the second row. A 2-byte circular left shift is performed on the third row. A 3-byte circular shift is performed on the fourth row. The inverse transformation performs the circular shifts in the opposite direction, to the right instead of to the left. [2]

Mix columns (forward and inverse): This transformation is performed on each column individually. Each of the column's bytes are mapped into a new value that is a function of all four of the bytes in that individual column. This transformation combined with the shift row transformation ensures that once a few rounds have passed, all of the output bits depend on all of the input bits. [2]

Add round key (forward and inverse): For this transformation forward and inverse are identical. This is because the XOR operation is its own inverse. The 128 bits of the state is bitwise XORed with the 128 bits of the round key. [2] This transformation is a byte-level operation. It is an operation between the 4 bytes of the state column and one word of the round key.

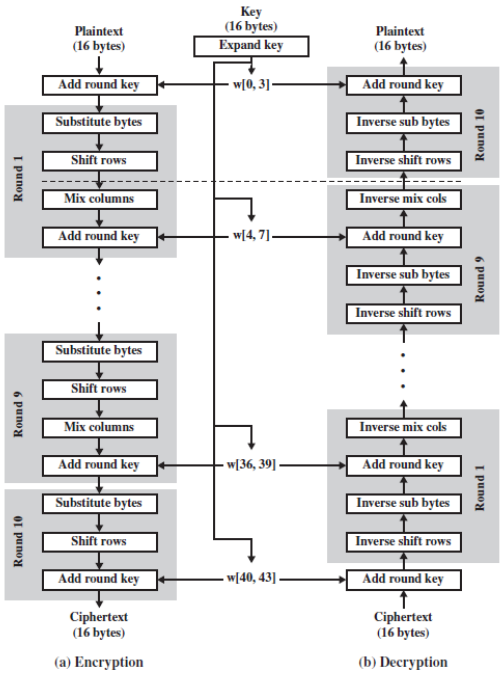

FIGURE I
The structure of AES encryption and decryption. [2]

The Key Expansion is comprised of two separate steps for computing round key (n+1) from round key (n) [1]. Step 1 begins by computing the new first column of the next round key. Then, substitution takes place for bytes in the old fourth column using the sub-bytes Operation. The bytes in the old fourth column shift vertically by one-byte position using the shift rows operation. Finally, it is XORed to the old first column. Step 2 involves calculating columns 2 thru 4 of the new round key. The calculations are as follows: "[new second column] = [new first column] XOR [old second column], [new third column] = [new second column] XOR [old third column], and [new fourth column] = [new third column] XOR [old fourth column]" [1].

In order to maintain confidence using the AES-128 algorithm, the code should be unbreakable, or at least computationally secure. To provide this assurance, patterns in frequency of letters and words used in the plaintext should be obscured by the algorithm. Randomness can be assessed by performing statistical tests on the generated encryption blocks.

III. NIST Randomness Tests Description

There are fifteen statistical tests prescribed by NIST. These tests will be utilized in our analysis to assess the randomness of the ciphertext block. [5] They can be broken down into five separate types of tests: Frequency, Runs, Linear Independence, Patterns and Excursions. All fifteen randomness tests and the inputs that are required to execute the test are shown in Table I. Descriptions for each input symbol is also provided for clarity when explaining each test in detail and to reduce redundancy from test to test.

The output of each test is a statistical p-value indicating the probability that the test statistic generated from the bit string tested is due to random variation. Each p-value is generated from a separate test statistic depending on the test that is then measured against its respective distribution. A test cannot deem a bit string random but can deem a bit string non-random if the p-value less than 0.01 as defined by NIST. [6]

Brief explanations of each of the fifteen tests will follow along with the input needed to conduct the test, the value chosen for our analysis, and the statistical distribution each test is based on. Note that full instructions with equations are not specified in all fifteen tests and one should reference NIST documentation if step-by-step details are desired. [6]

TABLE I
NIST RANDOMNESS TESTS [6]

Number	Name	Inputs
1	Frequency (Monobit) Test	n, ε
2	Frequency Test within a Block	M, n, ε
3	Runs Test	n, ε
4	Test for the Longest-Run-of-Ones in a Block	n, M, N, ε
5	Binary Matrix Rank Test	n, M*, Q*, ε
6	Discrete Fourier Transform (Spectral) Test	n, ε
7	Non-overlapping Template Matching Test	m, n, B, M, N, ε
8	Overlapping Template Matching Test	m, n, B, K, M, N, ε
9	Maurer's "Universal Statistical" Test	n, L*, Q*, ε
10	Linear Complexity Test	M, n K, ε
11	Serial Test	M, n, ε
12	Approximate Entropy Test	M, n, ε
13	Cumulative Sums (Cusums) Test	n, ε, mode*
14	Random Excursions Test	n, ε
15	Random Excursions Variant Test	n, ε

Input Definitions:
n = length in bits of the bit string, ε = input bit string of 1s and 0s,
M = length in bits of a bit string block; N = number of M-bit blocks,
M* = number of rows in matrix, Q* = number of columns in matrix,
m = length in bits of template, B = bit string of m-bit template,
K = degrees of freedom, L*= length in bits of a block,
Q* = number of blocks in initialization sequence, mode* = test indicator
*indicates the parameter is specific only to that particular test

A. *Frequency*

Frequency tests take a more basic approach where the probability of a 1 or 0 occurring is equivalent to a coin flip. A shift in this uniformity can cascade down to many or the tests following it. For this reason, NIST Tests 1 and 2 should be performed before as a non-random determination will likely lead to non-random determinations for more complex tests. The frequency can be measured as a whole or blocks at a specified length.

Frequency (Monobit) Test

The frequency test, also referred to as the monobit test, looks at the entire bit string as a whole or one single block. It will determine the overall proportion of zeros to ones by summing ones and zeros that have been converted to {1, -1} respectively and dividing the sum by the square root of the bit string length. The test statistic, the absolute value of the value calculated, is based on a normal distribution (z).

Frequency Test within a Block

Frequency tests within a block serve the same purpose but the bit string is separated into M-bit block for a series of frequency test. These proportions are made base on the proportion of ones and measure against what is expected for a perfectly random bit string, 0.5. The test statistic involves the use of the sum of variance in the proportions and follows a χ^2 distribution. NIST recommends the value of M to be above 100 and this analysis will use 128 since that is the block size bits are being generated in.

B. *Runs*

There are two tests for runs which focus on uninterrupted sequences of zeros and ones. One which looks at the total number of runs in the entire bit string and the other which only measures the longest run within a specified block. These tests focus on the aspect of unpredictability where the previous bit should not skew the 0.5 probability of either a one of a zero in the next bit.

Runs Test

The runs test counts the total number of runs across an entire bit string and measures that value against the number of runs that is expected in the length of the bit string. A run being defined as consecutive ones or zeros in a row. The test statistic is defined as the total number of runs but is slightly more complex than the previous two tests so a detailed description of the equation will not be noted. However, the statistic is referenced against a χ^2 distribution.

Test for Longest-Run-of-Ones in a Block

The test for the longest-run-of-ones in a block follows the runs concept but at a much different angle. A bit string is separated into M-bit blocks where the longest run of ones is measure in each block. The statistic is generated based on the frequency of the longest run length observed in each block and follows a χ^2 distribution. Note that it is assumed that a non-random run length of ones in a block will result in a non-random run length of zeros in a block.

C. *Linear Dependence*

Linear independence is an important component of the structure of random number sequences. It is desirable for a randomly generated sequence to display minimal linear dependence among the vector components. The following two tests identify deviations from linear independence in the generated sequences.

Binary Matrix Rank Test

The binary matrix rank test is applied to fixed length matrices of the generated sequence. The rank for each matrix is calculated and the rank distribution is aggregated. A χ^2 test statistic is computed from those values. Large χ^2 values would indicate deviation from a rank distribution generated by a random sequence. The input, M and Q, signify the rows and columns in a matrix. Since ciphertext is being generated in 128-bit blocks, matrices are right-sized as closely as possible for 128-bits. This entails making M and Q a value of 16, or two 128-bit blocks per matrix.

Linear Complexity Test

The linear complexity test utilizes the length of a linear feedback shift register (LFSR) to examine complexity of the sequence. Longer LFSRs are consistent with randomly generated sequences. The sequence is partitioned into blocks of a specified bit length and the test is applied to the blocks. The test statistic is measured against a χ^2 distribution and user specified inputs follow the values provided in NIST's test examples: M, length of a block, is 500 bits, and K, degrees of freedom is 6. [6]

D. Patterns

The detection of patterns is a cornerstone of cyrptanalysis. Although some patterns can be observed in randomly generated sequences, it is important that the patterns do not mimic what one would expect from examining a specific alphabet structure. Likewise, the space between any observed patterns is also worthy of review. The following tests specifically address patterns, repetition and sequence compressibility.

Discrete Fourier Transform (Spectral) Test

The discrete Fourier transform (spectral) test examines periodic features of the sequence. The ones and zeros in the sequence are transformed to {1, -1}. From the transformation, a sequence of peak heights are generated. The sequence of peak heights represents identified patterns, which is tested to determine if it deviates from what would be expected from a random sequence. There are no user defined parameters for the test and it follows a normal distribution (z).

Non-overlapping Template Matching Test

The non-overlapping template matching test examines occurrences of non-periodic patterns. To apply this test, the sequence is examined for specific m-bit patterns. Because this test is non-overlapping, if the determined pattern is detected, the search resumes with the bit following the observed pattern. The observed patterns are then compared to what would be expected from a randomly generated sequence. The entire bit string is separated into substrings prior to the pattern search. N substring blocks and the B matching template input parameters follow the examples provided within NIST documentation, 8 and '000000001'. The test follows a χ^2 distribution. [6]

Overlapping Template Matching Test

Similar to the non-overlapping template matching test, the overlapping template matching test is used to examine per-determined patterns of m-bit length. The difference occurs when a pattern is detected. In the non-overlapping test the search would resume following the detected pattern; however, with the overlapping test, the search will resume by shifting the examination window by only one bit. Similarly, the results will be compared with what would be expected from a randomly generated sequence. The test follows the same distribution as the non-overlapping version, χ^2, and input parameters were hard coded to NIST's recommendation: block length, m, equal to 1032, matching template, B, equal to '111111111', and degrees of freedom, K, equal to 5. [6]

Maurer's "Universal Statistical" Test

The universal statistical test (Maurer's test) is utilized to examine the number of bits found between matching patterns. This test is applied to analyze whether the sequence can be significantly compressed without losing information. A sequence that appears to be compressible would deviate from what would be expected from a randomly generated sequence. The bit string is broken into L-bit blocks where Q blocks are used to initialize L-bit patterns to test. This normally distributed (z) test changes input parameters L and Q depending on the length of the bit string. A reference table is provided by NIST. [6]

Serial Test

The serial test revisits the previously discussed frequency principal. Instead of examining the likelihood of generating any single bit, as with a coin flip, this test is utilized to detect overlapping M-bit patterns within the sequence. As with frequency tests, for a randomly generated sequence, it would be expected that each M-bit pattern has an equally likelihood of appearing in the sequence. NIST provides an equation to determine M depending on bit string length and the test follows a χ^2 distribution. [6]

Approximate Entropy Test

The approximate entropy test is an expansion of the application of the serial test. Similarly, the frequency of overlapping patterns is examined. However, instead of applying the test to equally sized blocks of M-bit length, the test is applied to blocks of two adjacent lengths of size M and M+1. As with the serial test, the frequency of the patterns detected in adjacent blocks is compared to what would be expected in a randomly generated sequence. The test also follows a χ^2 distribution and an equation for M is used dependent on bit string length. [6]

E. Excursions

There are three excursion tests included in the NIST testing suite. To apply these tests, the ones and zeros in the sequence are transformed to {1, -1}. An excursion can be described as a random walk from a state of zero. The results of the excursion tests are compared to what would be expected from a randomly generated sequence.

Cumulative Sums (Cusums) Test

The cumulative sums test examines the maximal excursions from the zero state. For a randomly generated sequence, one would expect the maximal excursions to remain close to zero throughout the sequence. The test can be conducted in two modes, forward or backward, indicating the direction the excursion is measured and follows a normal distribution (z).

Random Excursions Test

The random excursions test applies the same methodology as the cumulative sums test. However, instead of examining the maximal excursion, the excursions to eight specific states from zero are calculated. Although described as a single test, this is actually a group of eight separate tests as the number of excursions to states -4, -3, -2, -1, 1, 2, 3 and 4 are reviewed.

Each test statistic for a state is measured against a χ^2 distribution.

Random Excursions Variant Test

The random excursions variant test is an expansion of the previous random excursions test. Where previously the states examined included -4 through 4, the variant test examines excursions to 18 states including -9 through 9. However, opposed from the random excursions test, the random excursions variant test follows a normal distribution (z).

IV. Related Work

A recent 2015 publication presented an evaluation of several key schedule algorithms, including AES-128. The researchers generated 500 random vectors using a Pseudorandom Number Generator (PRNG), Blum-Blum Shub. The 500 vectors were then used as secret keys to generate sub-keys using each encryption algorithm.

Four statistical tests were used to determine randomness of the sub-keys generated from the algorithm. The tests selected included frequency, runs, poker and autocorrelation. The authors note that the AES-128 cipher performed well in three of the four statistical tests however, they noted deficiencies in the frequency test with an imbalanced distribution of zeros and ones in the generated sequences. [7]

Through research, a professional website was uncovered where an experimental scientist took the NIST randomness test suite and converted it into a python program. [8] The program was written in Python 2.6.6. There is no indication when the program was written but the release of Python 2.6.6 was August of 2010 and was the most recent version of Python 2.6 until June of 2011 setting an assumed creation date range. [9] The program was written as a wrapper to be run through the command line of an operating system.

Since the AES-128 ciphertext blocks were created in Python 3, the functions and programs from this existing application were repurposed for Python 3 to be used internally in a Jupyter Notebook. Because the python application was a 3rd party program not certified by NIST, all functions had to be validated with a NIST publication by deriving all fifteen of the randomness tests and certifying them with examples conducted in the text. [6]

V. Methods Used

A series of methods applied for various input scenarios were used in producing ciphertext blocks that were eventually tested at a bit level of ones and zeros. All blocks produced, techniques to convert strings to bits, and an in depth methodology with source code is noted in a Jupyter Notebook that accompanies this paper. [10] However, a brief explanation describing the scenarios and methods will follow for background during the analysis and interpretation sections.

A. Analysis Scenarios

Three scenarios were tested by varying the input parameters of the AES-128 cipher to test its versatility. Two separate options were toggled between the plaintext block and key inputs, either holding the input constant or refreshing the input for a newly generated independent one. These two options are denoted in the paper as "constant" for using the same input in each iteration to generate ciphertext blocks and "variable" for using a new input with each iteration.

There are four possible combinations for toggling the two options, however, the scenario using a constant key and a constant plaintext will not be tested since the cipher is planned to be used in Electronic Codebook mode (ECB). In this scenario, the same exact ciphertext block would be made repeatedly making a test for randomness useless. This leaves the three scenarios as follows: constant key/variable plaintext, variable key/constant plaintext, and variable key/variable plaintext. This nomenclature will be used throughout the paper to describe one of the three scenarios that were tested.

B. AES-128 Input Generation

Plaintext blocks were made in 128-bit blocks, congruent with the required input length needed for the AES-128 object in PyCrypto. Plaintext letters in each block made use of the random package in python [11] to select random numbers in a range where each alphabetic letter was assigned a frequency based on its usage in the English language. [12] Assigned frequencies are meant to mimic legitimate plaintext blocks transmitted over the internet. Even though legible words and sentence are not being construct, – by chance, words may appear – bias in letter selection deviates from the uniformity pseudo-random number generators (PRNG) produce. Therefore, AES-128 will be tested on its ability to hide this frequency bias in its ciphertext.

Keys are generated in a similar fashion but there are a couple key differences. Again, each key contains 128-bits, as specified for encryption. However, the keys are made using the PRNG within the machines operating system (OS) utilizing the urandom function in the os python package. [13] In key generation to locally encrypt information, it is reasonable to assume that the machine operating system would leverage its PRNG to produce the bits making up a 128-bit key, as the PRNG is readily available.

Both plaintext blocks and keys were generated and stored in arrays for each iteration. These arrays were dubbed "banks" and each bank was iterated or not iterated within a for loop depending on the scenario for which ciphertext blocks were being produced.

C. AES-128 Block Output Generation

The process of quickly generating 128-bit ciphertext blocks with AES-128 and storing blocks in a method where they can swiftly be analyzed required the use of the AES cipher object within the PyCrypto package, more specifically, the Cipher sub-package. [14] As with other applications that use the AES-128 algorithm for encryption, the AES cipher object requires a 128-bit plaintext, a 128-bit key, and the mode the algorithm is to run in. There are 6 different options for a mode, most which are the taught as the modes of operation for cryptography (i.e. Electronic Codebook (ECB), Cipher-Block Chaining (CBC), Counter (CTR), etc.). To analyze AES-128 in its simplest form, ECB was selected which is also the default mode in generating a AES cipher object. [14]

As with the inputs for encryption, the ciphertext output was placed in banks for each iteration. Three separate banks had to be made for each scenario: constant key/variable plaintext, variable key/constant plaintext, and variable key/variable plaintext. Furthermore, to fully test blocks as they are being created for frequency, patterns, and properties required for randomness, blocks must be appended to one another. Forming a ciphertext stream that grows in length with each iteration. The last value in each bank holds the most information, but testing the stream with each iteration as it extends gives the ability to track trends, which can indicate if a test will results in a non-random response even though the last p-value is not identified as significant.

VI. AES-128 Analysis Results

NIST suggests only testing the first two randomness tests prior to executing the rest of the fifteen tests. [5] The recommendation comes from the theory that since these two frequency tests are relatively simplistic, if either finds a bit string to be non-random then most likely the other thirteen more complex tests will also find the bit string non-random.

Tests 1 and 2 are plotted with trend charts as ciphertext blocks are appended and retested to view any significant trends that move closer to a p-value of 0.01. This allows a level of prediction where even if the final p-value is above 0.01, a determination can be made where eventually it will be below 0.01 at a certain bit string length.

NIST randomness test one is the frequency, monobit, test. This test is to see the number of ones and zeros in a sequence is what is expected in a random bit string. The trend of p-values as ciphertext blocks are appended to a bit string stream are displayed in Figure II. There is an upward trend for variable key/constant plaintext, which is expected. Constant key/variable plaintext and variable key/variable plaintext show some downward trends in p-values. However, those trends appear to be countered with upward trends. This observed oscillation gets tighter as the stream grows in bits, flattening out at p-values about 0.01, deeming the sequences to be random.

NIST randomness test two is the frequency test within a block. This test is to see if the number of ones in the M-bit block occurs. The three scenarios are plotted for the frequency test within a block in Figure III. No three of the scenarios show a significant downward trend. This can conclude that no three of the scenarios are non-random.

Simply non-random determinations will likely lead to non-random determinations for more complex tests. Which is why the first two tests are important. However, since there were not any non-random determinations in the first two test, the following thirteen tests were conducted. Results from the fifteen tests can be referenced in Table II. There are two tests of the total fifth-teen that yield a significant p-value; test for longest run on ones in a block (constant key/variable plaintext) and random excursions test (variable key/variable plaintext, state x=3). The p-values were below 0.01 and test results for those bit strings are considered non-random. To confirm, a trend analysis will be done. In addition, at least one scenario in eight of the fifth-teen tests resulted in a p-value below 0.10 which triggers an in depth-trend analysis on those as well.

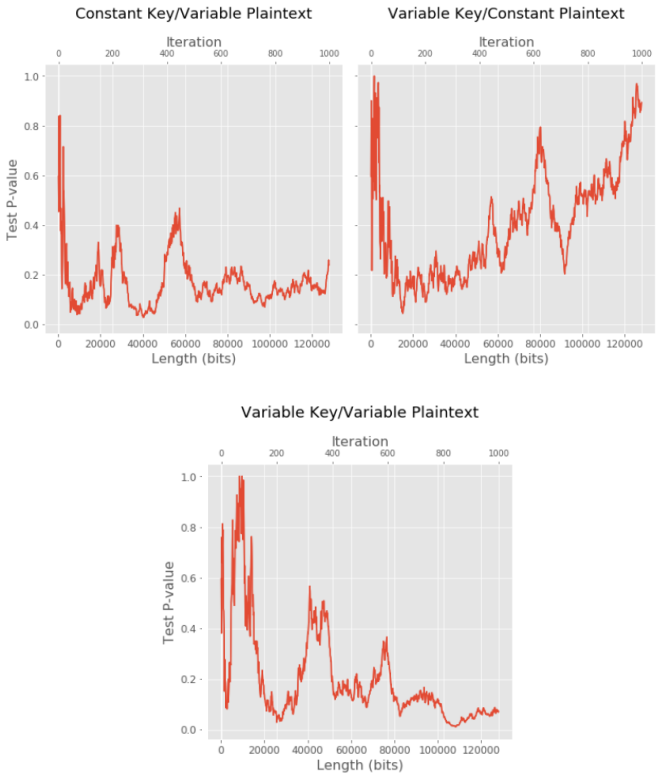

FIGURE II
AES-128 NIST Randomness Test 1: Frequency Test Results. [10]

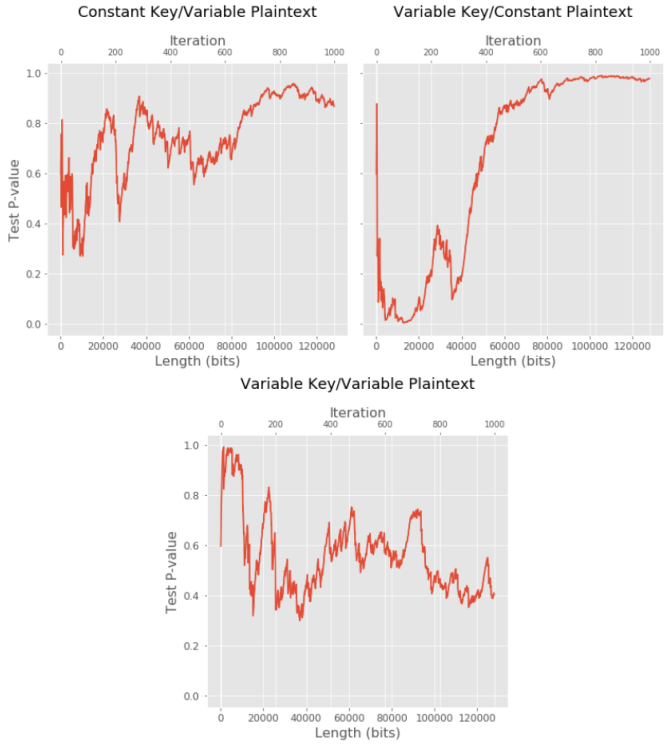

FIGURE III
AES-128 NIST Randomness Test 2:
Frequency Test within a Block Results. [10]

TABLE II
AES-128 ANALYSIS: NIST RANDOMNESS TEST RESULTS

NIST Test	Constant Key/ Variable Plaintext	Variable Key/ Constant Plaintext	Variable Key/ Variable Plaintext
1	0.23819	0.89327	*0.07097*
2	0.86493	0.97750	0.40308
3	0.33434	0.69967	0.21797
4	***0.00966***	*0.01477*	0.58907
5	0.64424	0.12489	0.18237
6	0.64430	0.70043	0.55523
7	0.80283	0.21857	*0.09146*
8	0.73352	0.39481	0.96816
9*	*0.06629*	*0.01539*	0.41093
10	*0.06825*	0.50388	0.90206
11*	0.74716	0.14381	0.85810
12	0.78939	0.13046	0.41908
13*	0.21973	0.55643	*0.04069*
14*	*0.07525*	0.11346	***0.00220***
15*	*0.01444*	0.28884	0.10403

*Test returns a series of p-values from different states/modes of test. Lowest p-value is recorded since all must pass for the test as a whole to pass as random.
underlined italics: p-value below 0.10 triggering a in depth trend analysis.
bold: p-value below 0.01 and test results in bit string as non-random. Trend analysis to be done for confirmation.

VII. AES-128 ANALYSIS INTERPRETATION

For the sake of length, only interesting trends uncovered during the further investigation are covered in this section. All bold and/or italicized and underlined p-values noted in Table II were fully investigated on the Jupyter Notebook previously cited. [10] It can be viewed for a deeper understanding but it will be assumed any test in need of a trend analysis not mentioned in this section is assumed to be insignificant. Meaning there was not a clear downward trend towards zero as more ciphertext blocks were appended to the stream.

Trend charts for both constant key/variable plaintext and variable key/constant plaintext scenarios, shown in Figure IV, show a sharp decline past a bit length of about 70,000 bits in NIST Test 4: Test for Longest-Run-of-Ones within a Block. Constant key/variable plaintext appears to have tapered off in its decline but that is at a point after p-values are below 0.01. While variable key/constant plaintext appears to still be in its decent. The trend would suggest given slightly more iterations of ciphertext blocks would result in a non-random p-value.

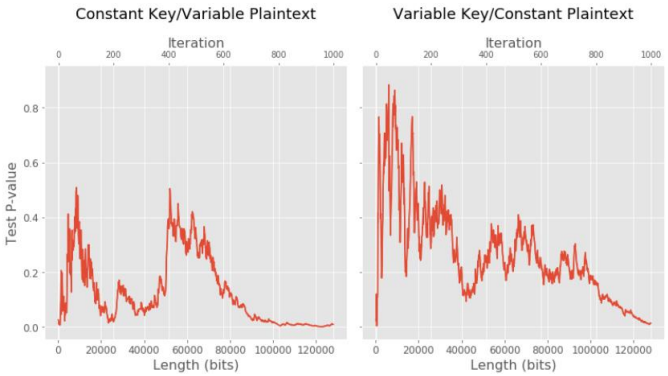

FIGURE IV
AES-128 NIST Randomness Test 4: Test for Longest-Run-of-Ones within a Block Results. [10]

Furthermore, a similar phenomenon is observed for the same two scenarios in Figure V showing the trend charts for NIST Test 9: Maurer's "Universal Statistical" Test. An extremely steep decent is seen in constant key/variable plaintext after the bit string exceeds 90,000 bits. While variable key/constant plaintext experiences a much subtler decline towards zero after 50,000 bits. Even though the trends are slightly different, both scenarios, holding keys constant and the holding plaintext blocks constant, show evidence that the manner in which ciphertext blocks are being generated is not consistent with what is considered to be a random bit string.

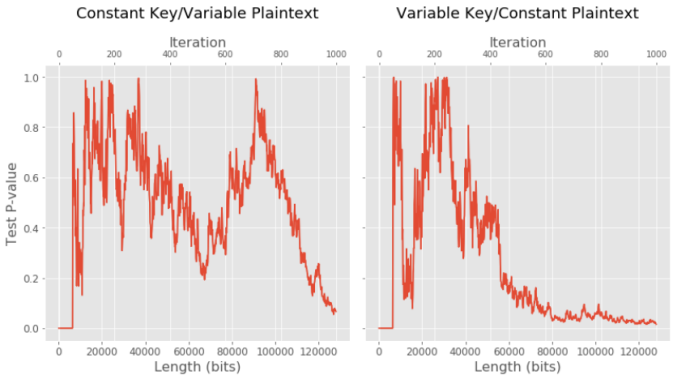

FIGURE V
AES-128 NIST Randomness Test 9: Maurer's "Universal Statistical" Test Results. [10]

VIII. CONCLUSIONS

In summary, it is shown that only a single test in the variable key/variable plaintext scenario produced a non-random p-value. Through investigation, it could be determined that this occurrence may in fact be an anomaly. For the other two scenarios, constant key/variable plaintext and variable key/constant plaintext, however, there is reason to believe the method in which ciphertext blocks were produced is non-random. Evidence backing this claim stems from the results of the Test for the Longest-Run-of-Ones in a Block and Maurer's "Universal Statistical" Test, NIST tests 4 and 9 respectively. In both tests for both scenarios, steep downward trends were found as more ciphertext blocks were appended to each other growing the length of the bit string tested. And in one case, NIST test 4 for constant key/variable plaintext, the decline went beyond the 0.01 threshold producing non-random p-values thereafter.

These results make practical sense based on the notion that ECB mode was used in implanting the AES-128 cipher. ECB, Electronic Codebook, has been noted for containing vulnerabilities when messages are large in length. Highly structured messages also have potential to be exploited through finding patterns which cryptanalysts can begin to determine plaintext-ciphertext pairs. [2] In both scenarios where randomness is in question, there is consistent structure provided. Constant plaintext establishes a vulnerable structure where known plaintext can be found and constant keys in junction with non-uniform plaintext frequency, alphabetic letter frequency consistent with their usage in the English language, establishes a vulnerable structure since the same b-bit block will

produce the same ciphertext. [2]

Delving into the minute details of the AES-128 cipher using ECB mode can provide further reasoning for the non-random trend results that were observed for NIST tests 4 and 9. This leads to a topic of discussion in future work for researchers by building upon the base established in this paper.

REFERENCES

[1] U. Kretzschmar. (2009, July). AES128 – A C Implementation for Encryption and Decryption. [Online].

[2] W. Stallings. (2014). Cryptography and Network Security – Principles and Practice, Sixth Edition.

[3] J. Soto, L. Bassham. (2000, March). *NIST*. Randomness Testing of the Advanced Encryption Standard Finalist Candidates. [Online]. http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=151216

[4] M. Rouse, M. Cobb, (2014, November). *TechTarget*. Advanced Encryption Standard (AES). [Online]. http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard

[5] (2017, September). *NIST*. Random Bit Generation: Guide to the Statistical Tests. [Online]. https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software/guide-to-the-statistical-tests

[6] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, S. Vo. L. Bassham. (2010, April). *NIST*. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Publication: 800-22 ver. 1a. [Online]. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

[7] "STATISTICAL ANALYSIS OF KEY SCHEDULE ALGORITHMS OF DIFFERENT BLOCK CIPHERS." *Science International*, vol. 27, no. 3, 2015

[8] I. Gerhardt. Random Number Testing: The NIST testsuite in Python. [Online]. https://gerhardt.ch/random.php

[9] *Python Software Organization*. Downloads. [Online]. https://www.python.org/downloads/

[10] J. Nelson, T. McWilliams, L. Harris. Generating & Testing AES Cipher Blocks Using NIST Randomness Test Suite. *DataScience@SMU*. (2017, December). [Online]. https://github.com/JackNelson/MSDS7349_TermProject/blob/master/MSDS7349_Project.ipynb

[11] *Python Software Foundation*. 9.6 random – Generate pseudo-random numbers. (2017, December). [Online]. https://docs.python.org/3/library/random.html

[12] A. Sweigart. "Hacking Secret Ciphers with Python". *inventwithpython.com*. (2013, April). Chapter 20. [Online]. https://inventwithpython.com/hacking/chapter20.html

[13] *Python Software Foundation*. 16.1 os – Miscellaneous operating system interfaces. (2017, December). [Online]. https://docs.python.org/3/library/os.html

[14] *PyCrypto.org*. Package Crypto. Version 2.6. (2012, May). [Online]. https://www.dlitz.net/software/pycrypto/api/2.6/