Statistical Analysis of Advanced Encryption Standard

David Josephs
Southern Methodist University
Dallas, Texas
Email: josephsd@smu.edu

Hannah Kosinovsky
Southern Methodist University
Dallas, Texas
Email: hkosinovsky@mail.smu.edu

Carson Drake
Southern Methodist University
Dallas, Texas
Email: drakec@smu.edu

Volodymyr Orlov
Southern Methodist University
Dallas, Texas
Email: vorlov@smu.edu

***Abstract*—Advanced Encryption Standard (AES) is one of the most common and widely used specification for the encryption of electronic data. AES is a block cipher with 128-bit internal state and 128/192/256-bit key (AES-128, AES-192, AES-256, respectively). No efficient attacks against AES are known at this time and the standard is considered practically secure. In this paper we perform an extensive statistical analysis of AES-128 output using NIST Statistical Test Suite and additional randomness tests with a goal to identify any bias in either the entirety of the encrypted output or in sequences of encryption blocks generated from input values created using a counter or a linear feedback shift register (LFSR).**

I. Introduction

Advanced Encryption Standard is a symmetric key block cipher method established by the U.S. National Institute of Standards and Technology (NIST) in 2001. In AES the same key is used for both encrypting and decrypting the data. Since its introduction, AES has been adopted by the U.S. government and is now used for a variety of applications worldwide. There are three variants of AES: AES-128, AES-192 and AES-256, where the number after AES indicates the key length used for encryption and decryption process. Since its adoption, the world has seen little progress in the cryptanalysis of this cipher.

One of the basic properties of AES is indistinguishability of its output from a random sequence of bits. An evaluation of the cipher's output using randomness tests is an important tool in cryptanalysis that helps to ensure the algorithm produces no distinguishable patterns which can be used to deduce an encryption key or a plain text input. For this reason, the evaluation of the output of the AES by means of statistical randomness tests is of great importance. This study analyzes randomness of the output produced by the AES-128 block cipher using NIST statistical test suite and the Diehard test battery.

II. Statistical Randomness Tests

Statistical tests for randomness take arbitrary length input sequence and analyze its distribution to determine if it is random, containing no recognizable patterns or regularities. Usually these tests produce a real number between 0 and 1, the p-value, which shows the probability of finding the observed, or more extreme, results with respect to certain randomness properties of the given input. There exists some Notable software implementations of these statistical tests for randomness, like the NIST Statistical Test Suite or Diehard tests. Tests such as these can be used to evaluate the randomness of AES.

The NIST Test Suite consists of 15 tests specially designed to analyze binary sequences:

- Two frequency tests, for monobit and a block, that test the randomness of a sequence of zeroes and ones
- Two runs test: a simple test and a test for the longest run of ones in a block, which looks for uninterrupted sequence of identical bits in the tested message.
- Binary matrix rank test, which checks for linear dependencies among fixed-length substrings of the original sequence.
- Discrete Fourier transform test, which detects periodic patterns that repeat and are near each other in the tested sequence.
- Overlapping and non-overlapping template matching tests, which detect generators that produce too many occurrences of a given non-periodic pattern.
- Maurer's "Universal Statistical" test, which uses compression without loss to detect whether or not the compressed sequence has less information than the original message.
- Linear complexity test, which determines whether or not the sequence is complex enough to be considered random.
- Serial test, which searches for number of occurrences of the 2^m m-bit overlapping patterns and makes sure that its frequency is approximately the same as would be expected for a uniformly distributed sequence.
- Approximate entropy test, which compares the frequency of overlapping blocks of two consecutive lengths against the expected result for a random sequence.
- Cumulative sums test, that determines whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences.
- Random excursions and random excursions variant tests, which determines if the number of visits to a particular state within a cycle deviates from what one would expect for a random sequence and detect deviations from the expected number of visits to various states in the random walk

All NIST tests examine randomness for the whole binary sequence. In addition to that several tests are also able to detect local regularities.

Aside from the NIST test suites, there are a few other test suites for testing the randomness of cryptographic pseudorandom numbers, such as the Dieharder test suit, SPRNG, and the test suite mentioned in (Statistical Testing of Cryptographic Randomness, Demirhan et al., 2016), which combines the Knuth, Helsinki, Diehard, and SPRNG test batteries.

There is overlap between the NIST Statistical Test Suite and Diehard tests since both of these tests have binary matrix rank tests. However the NIST tests focus on bitlevel randomness and the suitability of a random number generator for cryptography. The tests that are overlapping with STS in the dieharder package – which our group used to run Diehard tests- were rewritten in C by Robert G. Brown to make sure there were no copyright issues and all tests used are attributed to the various authors of that suite. The Dieharder library is an open source project. There are original tests and timing operations inserted by Robert G. Brown in addition to the test available in STS or Diehard. These tests are a check of random numbers using a derived statistic. The Diehard Suite itself was developed by George Marsaglia and was published in 1995. The purely Diehard tests were the tests with "diehard" before the name of the test: ie the first 19 tests from the output found in Table 6. A summary of the tests that are performed:

- The "Birthday spacings" Test is named after the birthday paradox – the paradox that there are seemingly more matching birthdays between n people in a room than expected. The test choses random points on a large interval and the spacings between the points are expected to be asymptoticallyexponentially distributed.
- The "Overlapping permutations" Test analyzes sequences of five consecutive random numbers. It is expected that the 120 possible orderings should occur with statistically equal probability.
- The "Ranks of matrices" Test selects a set number of left-most bits from a randomly chosen number to form a matrix over 0,1, and then determines the rank of said matrix. The rank is then determined. Ranks of less than 28 are considered rare.
- The "Monkey" Tests are named after the monkey theorem – a theorem than states that a monkey hitting random keys on a typewriter for an infinite amount oftime will produce, for example, the entirety of Shakespeare's work. The tests treatsequences of some number of bits as "words", counts the overlapping words in a stream, and then determines whether the number of "words" that don't appear follow a normal distribution (the expected outcome).
- The "Parking lot" Test tests whether a "crash" occurs when you randomly park a car in a 100x100 square. If we plot n attempts with k number of successful parked cars, the behavior if it matches to a random number generator should show that k should average 3523 with sigma 21.9.
- The "Minimum distance" Test randomly places 8000 points in a 10000x10000 square, then finds the minimum distance between the pairs. The square of the minimum distance should beexponentially distributed.
- The "Random spheres" Test randomly choses 4000 points in a cube of edge 1000, centers a sphere on each point, whose radius is the minimum distance to another point, then determines the distribution of the smallest sphere's volume. We expect that this volume should be exponentially distributed with a certain mean.
- The "squeeze" Test multiplies 2^{31} by random floats on (0,1) until the value 1 is reached. This is repeated 100000 times and the distribution of the number of floats is observed and compared to the expected distribution.
- The "Overlapping sums" Test generates a long sequence of random floats on (0,1), adds sequences of 100 consecutive floats, and then observes the distributionof the sums. The sums are expected to be normally distributed.
- The "Runs" Test generates a long sequence of random floats on (0,1) and then counts ascending and descending runs. The observed counts are expected to follow a certain distribution.
- The "craps" Test "plays" 200000 games ofcraps, counts the wins and the number of throws per game, and then observed the counts of each game. The counts are expected to follow a certain distribution.

All the above tests return a pvalue between 0 and 1. A bit stream will fail the test at a .05 level if $p0.025$ or $p > 0.975$

The reason for including these various tests is to cover a wider array of statistical methods in order to detect a lack of randomness in AES-128. The NIST test suite tests for various metrics such as entropy, frequency within a block, random excursions, etcetera. In contrast, the 26-test Dieharder battery tests for distributions, bit distances, overlapping permutations and sums, while the SPRNG battery (13 tests) covers more stochastic processes such as random walks and the Ising model (a mathematical model of ferromagnetism), and the Helsinki test looks for correlations and blocks within the pseudorandom data. The Knuth battery contains a extends the testing scope by including additional unique tests.

III. Experiments

The system of experiments employed in this study consist of 4 blocks:

1) A plaintext generator is first used to generate the necessary datasets consisting of nine different categories of data
2) AES-128 Cipher is then used to encrypt plaintext messages
3) NIST tests suite are used to run basic tests against encrypted messages.
4) Diehard/Dieharder tests suite then serves as a final, more stringent, level of tests.

All four components are schematically represented in Figure 1. All modules run on the SMU ManeFrame II cluster, maximizing block creation capacity. The extensive datasets of cypherblocks give an accurate indication whether any

non-random patterns in encrypted messages are due to AES encryption or statistical outliers.

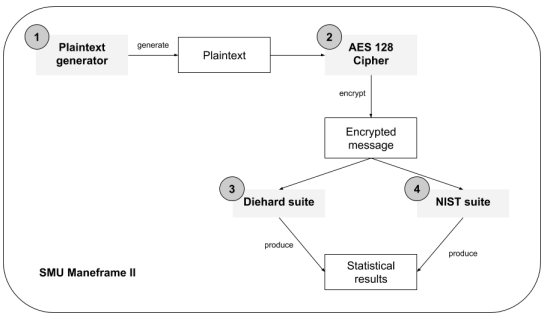

Fig. 1: Schematic layout of major components of the system

A. AES Datasets

The following six categories of encrypted datasets have been selected because of their usefulness in evaluating the randomness of an algorithm's output.

1) Low Density 128-Bit Key: The Low Density 128-Bit Key dataset consists of 300 sequences. Each sequence is made up of 8,257 ciphertext blocks. The ciphertext block is computed using a random plaintext block (unique to the individual sequence) and a low density 128-bit key. The first block utilizes a 128-key of all unset bits. Blocks 2-129 use a key consisting of 127 unset bits and one set bit, rotating the set bit across all 128-bit positions. For blocks 130-8,257, the key consists of 126 unset bits and two set bits, rotating the set bits in all unique combination across the 128-bit positions. All blocks are computed in ECB mode.

2) Low Density Plaintext: The Low Density Plaintext data set consists of 300 sequences. Each sequence is made up of 8,257 ciphertext blocks. The ciphertext block is computed using a random 128-bit key (unique to the individual sequence) and a low density plaintext block. The first plaintext block consists of 128 unset bits. Blocks 2-129 plaintext consisting of 127 unset bits and one set bit, rotating the set bit across all 128-bit positions. For blocks 130-8,257, the plaintext block consists of two set bits and 126 unset bits, rotating the set bits in all unique combination across the 128-bit positions. All blocks are computed in ECB mode.

3) High Density 128-Bit Key: The High Density 128-Bit Key dataset consists of 300 sequences. Each sequence is made up of 8,257 ciphertext blocks. The ciphertext block is computed using a random plaintext block (unique to the individual sequence) and a high density 128-bit key. The first block utilizes a 128-key of all set bits. Blocks 2-129 use a key consisting of 127 set bits and one unset bit, rotating the unset bit across all 128-bit positions. For blocks 130-8,257, the key consists of 126 set bits and two unset bits, rotating the set bits in all unique combination across the 128-bit positions. All blocks are computed in ECB mode.

4) High Density Plaintext: The High Density Plaintext dataset consists of 300 sequences. Each sequence is made up of 8,257 ciphertext blocks. The ciphertext block is computed using a random 128-bit key (unique to the individual sequence) and a high density plaintext block. The first plaintext block consists of 128 set bits. Blocks 2-129 plaintext consisting of all set bits and one unset bit, rotating the unset bit across all 128-bit positions. For blocks 130-8,257, the plaintext block consists of two unset bits and 126 set bits, rotating the unset bits in all unique combination across the 128-bit positions. All blocks are computed in ECB mode.

5) Random Plaintext/Random 128-Bit Key: The Random Plaintext dataset consists of 300 sequences. Each sequence is made of 8,257 ciphertext blocks. To make each sequence, a random 128-bit key is first produced. Then that key is concatenated with 8,257 random plaintext blocks in ECB mode to generate the ciphertext sequence.

6) Plaintext/Ciphertext Correlation: The Plaintext/Ciphertext Correlation is generated in order to study how plaintext-ciphertext pairs relate. 300 sequences were again generated, with a block length of 8,257, following the methods seen in subsubsection III-A5. The original plaintext is then XOR'd with the produced ciphertext, producing the dataset for analysis.

B. Analysis

All tests are implemented Python and C programming languages. For the NIST test suite, a version of the software recommended in paper by Lawrence Bassham and others [1] and modified it for batch execution. For the Diehard test suite, we use Robert G. Brown's Dieharder implementation of this suite. [1]. Both implementations require some additional work from our side to run on SMU ManeFrame II cluster.

C. Results

The results of applying NIST tests are depicted in Table 1, Table 2 and Table 3. Each row corresponds to the statistical tests applied. The second column in a P-value that was calculated using a chi-square test, that is used to test the homogeneity of P-values of a given statistical test. The third column is the proportion of tests sequences that passed.

We have truncated number NonOverlappligTemplate rows from the Table 3 to 20 but this does not change the overall results. As you can see, NIST test suite hasn't found any abnormalities or patterns in bytes generated by AES-128 cipher with p-values exceeding 0.05 which indicates that generated ciphertext cannot be distinguished from random.

Results of applying Dieharder statistical tests for randomness can be found in Table 4, Table 5 and Table 6

[1]Dieharder implementation https://bit.ly/2Sm325J

p-value	Proportion	Test Name
0.637119	19/20	NonOverlappingTemplate
0.534146	20/20	NonOverlappingTemplate
0.035174	20/20	NonOverlappingTemplate
0.350485	20/20	NonOverlappingTemplate
0.012650	20/20	NonOverlappingTemplate
0.275709	19/20	NonOverlappingTemplate
0.834308	20/20	NonOverlappingTemplate
0.834308	20/20	NonOverlappingTemplate
0.048716	20/20	NonOverlappingTemplate
0.213309	20/20	NonOverlappingTemplate
0.911413	20/20	NonOverlappingTemplate
0.964295	20/20	NonOverlappingTemplate
0.534146	20/20	NonOverlappingTemplate
0.534146	19/20	NonOverlappingTemplate
0.350485	19/20	NonOverlappingTemplate
0.025193	19/20	NonOverlappingTemplate
0.637119	20/20	NonOverlappingTemplate
0.834308	20/20	NonOverlappingTemplate
0.911413	20/20	NonOverlappingTemplate
0.739918	20/20	NonOverlappingTemplate

TABLE 1: First 20 rows of results of applying NonOverlappingTemplate test to AES-128 cipher output.

p-value	proportion	statistical test
0.437274	13/13	RandomExcursions
0.025193	13/13	RandomExcursions
0.275709	13/13	RandomExcursions
0.275709	13/13	RandomExcursions
0.006196	13/13	RandomExcursions
0.637119	13/13	RandomExcursions
0.275709	13/13	RandomExcursions
0.162606	13/13	RandomExcursions
0.162606	13/13	RandomExcursionsVariant
0.437274	13/13	RandomExcursionsVariant
0.090936	13/13	RandomExcursionsVariant
0.834308	13/13	RandomExcursionsVariant
0.162606	13/13	RandomExcursionsVariant
0.162606	13/13	RandomExcursionsVariant
0.437274	13/13	RandomExcursionsVariant
0.964295	13/13	RandomExcursionsVariant
0.637119	13/13	RandomExcursionsVariant
0.637119	13/13	RandomExcursionsVariant
0.637119	12/13	RandomExcursionsVariant
0.437274	13/13	RandomExcursionsVariant
0.090936	13/13	RandomExcursionsVariant
0.275709	13/13	RandomExcursionsVariant
0.275709	13/13	RandomExcursionsVariant
0.162606	13/13	RandomExcursionsVariant
0.048716	13/13	RandomExcursionsVariant
0.637119	13/13	RandomExcursionsVariant

TABLE 2: Results of applying RandomExcursions and RandomExcursionsVariant tests to AES-128 cipher output.

p-value	proportion	statistical test
0.437274	19/20	Frequency
0.035174	20/20	BlockFrequency
0.911413	19/20	CumulativeSums
0.834308	19/20	CumulativeSums
0.162606	20/20	Runs
0.162606	20/20	LongestRun
0.213309	19/20	Rank
0.739918	20/20	FFT
0.437274	20/20	OverlappingTemplate
0.834308	20/20	Universal
0.964295	20/20	ApproximateEntropy
0.066882	20/20	Serial
0.012650	20/20	Serial
0.275709	19/20	LinearComplexity

TABLE 3: Results of applying rest of the NIST statistical test for random to AES-128 cipher output.

Test Name	p-value	Result
sts_serial	0.84113664	PASSED
sts_serial	0.96089158	PASSED
sts_serial	0.35277909	PASSED
sts_serial	0.91734586	PASSED
sts_serial	0.99090967	PASSED
sts_serial	0.49106431	PASSED
sts_serial	0.94731942	PASSED
sts_serial	0.7974447	PASSED
sts_serial	0.19434614	PASSED
sts_serial	0.14804278	PASSED
sts_serial	0.60417195	PASSED
sts_serial	0.94910058	PASSED
sts_serial	0.20950952	PASSED
sts_serial	0.26799912	PASSED
sts_serial	0.86474793	PASSED
sts_serial	0.09013199	PASSED
sts_serial	0.674446	PASSED
sts_serial	0.44399883	PASSED
sts_serial	0.63547041	PASSED
sts_serial	0.80168842	PASSED
sts_serial	0.86659603	PASSED
sts_serial	0.61317151	PASSED
sts_serial	0.37714509	PASSED
sts_serial	0.41636181	PASSED
sts_serial	0.26001306	PASSED
sts_serial	0.86892099	PASSED
sts_serial	0.97947313	PASSED
sts_serial	0.94870208	PASSED
sts_serial	0.80682811	PASSED
sts_serial	0.23906065	PASSED

TABLE 4: Results of applying sts_serial statistical test for randomness to AES-128 cipher output.

Test Name	p-value	Result
rgb_lagged_sum	0.20386945	PASSED
rgb_lagged_sum	0.53547734	PASSED
rgb_lagged_sum	0.4592472	PASSED
rgb_lagged_sum	0.98164688	PASSED
rgb_lagged_sum	0.30178746	PASSED
rgb_lagged_sum	0.15326182	PASSED
rgb_lagged_sum	0.35070032	PASSED
rgb_lagged_sum	0.81952418	PASSED
rgb_lagged_sum	0.07805938	PASSED
rgb_lagged_sum	0.20400501	PASSED
rgb_lagged_sum	0.17304838	PASSED
rgb_lagged_sum	0.77597887	PASSED
rgb_lagged_sum	0.67072876	PASSED
rgb_lagged_sum	0.92418862	PASSED
rgb_lagged_sum	0.37276286	PASSED
rgb_lagged_sum	0.60619919	PASSED
rgb_lagged_sum	0.48663031	PASSED
rgb_lagged_sum	0.59630855	PASSED
rgb_lagged_sum	0.41877884	PASSED
rgb_lagged_sum	0.54354751	PASSED
rgb_lagged_sum	0.97941841	PASSED
rgb_lagged_sum	0.67390227	PASSED
rgb_lagged_sum	0.31168676	PASSED
rgb_lagged_sum	0.19550757	PASSED
rgb_lagged_sum	0.56669375	PASSED
rgb_lagged_sum	0.21088034	PASSED
rgb_lagged_sum	0.70869503	PASSED
rgb_lagged_sum	0.88030594	PASSED
rgb_lagged_sum	0.14739976	PASSED
rgb_lagged_sum	0.38008043	PASSED
rgb_lagged_sum	0.61643283	PASSED
rgb_lagged_sum	0.04652487	PASSED
rgb_lagged_sum	0.99766683	WEAK

TABLE 5: Results of applying rgb_lagged_sum statistical test for randomness to AES-128 cipher output.

D. Code

All the code and presentation materials can be found on GitHub. The project is divided into following folders:

- **src** NIST, Diehard and Plaintext Generator code can be found in this directory.
- **results** here you will find CSV tables with results.
- **reports** all presentation materials can be found here.
- **resources** we have collected some useful literature in this folder.
- **submissions** paper drafts and other submitted presentation materials.
- **notes** notes from internal ideas and discussions within the group.

Test Name	p-value	Result
diehard_birthdays	0.18833821	PASSED
diehard_operm5	0.04533398	PASSED
diehard_rank_32x32	0.83444864	PASSED
diehard_rank_6x8	0.81761741	PASSED
diehard_bitstream	0.73817707	PASSED
diehard_opso	0.09790803	PASSED
diehard_oqso	0.57240383	PASSED
diehard_dna	0.67244329	PASSED
diehard_count_1s_str	0.99455563	PASSED
diehard_count_1s_byt	0.66503821	PASSED
diehard_parking_lot	0.25282458	PASSED
diehard_2dsphere	0.9467929	PASSED
diehard_3dsphere	0.88286367	PASSED
diehard_squeeze	0.86716275	PASSED
diehard_sums	0.29072145	PASSED
diehard_runs	0.05858964	PASSED
diehard_runs	0.23169917	PASSED
diehard_craps	0.93126124	PASSED
diehard_craps	0.47630608	PASSED
marsaglia_tsang_gcd	0.57101448	PASSED
marsaglia_tsang_gcd	0.90439992	PASSED
sts_monobit	0.8247006	PASSED
sts_runs	0.20987187	PASSED
rgb_bitdist	0.35198005	PASSED
rgb_bitdist	0.28282591	PASSED
rgb_bitdist	0.54552001	PASSED
rgb_bitdist	0.99392011	PASSED
rgb_bitdist	0.09801383	PASSED
rgb_bitdist	0.89250352	PASSED
rgb_bitdist	0.19245717	PASSED
rgb_bitdist	0.39913738	PASSED
rgb_bitdist	0.23286272	PASSED
rgb_bitdist	0.54883951	PASSED
rgb_bitdist	0.40365781	PASSED
rgb_bitdist	0.94107559	PASSED
rgb_minimum_distance	0.289022	PASSED
rgb_minimum_distance	0.5274979	PASSED
rgb_minimum_distance	0.71582131	PASSED
rgb_minimum_distance	0.11863226	PASSED
rgb_permutations	0.11631851	PASSED
rgb_permutations	0.94693423	PASSED
rgb_permutations	0.42068399	PASSED
rgb_permutations	0.64702528	PASSED
rgb_kstest_test	0.60168538	PASSED
dab_bytedistrib	0.31696752	PASSED
dab_dct	0.31872778	PASSED
dab_filltree	0.09053199	PASSED
dab_filltree	0.63791435	PASSED
dab_filltree2	0.96009477	PASSED
dab_filltree2	0.50575354	PASSED
dab_monobit2	0.69171441	PASSED

TABLE 6: Results of applying rest of the Diehard statistical test for random to AES-128 cipher output.

IV. Conclusion

A vast number of tests on various sequences of encrypted bytes were performed. Not only did we run the standard suite with 15 tests which are described in famous NIST standard [1] but also ran an extended version of the Diehard suite; dieharder [5]. We have not found any significant deviation from randomness in sequences of bytes produced by AES-128 even after running our tests against a multi-gigabyte output of the cipher for multiple days. Thus we conclude that AES-128 produces sequences of bytes that look random and thus are secured against brute-force attacks.

References

[1] Lawrence Bassham (NIST), Andrew Rukhin (NIST), Juan Soto (NIST), James Nechvatal (NIST), Miles Smid (NIST), Elaine Barker (NIST), Stefan Leigh (NIST), Mark Levenson (NIST), Mark Vangel (NIST), David Banks (NIST), N. Heckert (NIST), James Dray (NIST). *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. April 2010.

[2] George Marsaglia. *The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness*. Florida State University. 1995.

[3] Haydar Demirhan, Nihan Bitirim *Statistical Testing of Cryptographic Randomness*. Journal of Statisticians: Statistics and Actuarial Sciences. 2016.

[4] Juan Soto *Randomness Testing of the AES Candidate Algorithms*. National Institute of Standards and Technology.

[5] Juan Soto, Lawrence Bassham *Randomness Testing of the Advanced Encryption Standard Finalist Candidates*. National Institute of Standards and Technology. 2000.