Detecting Hidden Encrypted Volume Files via Statistical Analysis

Mario Piccinelli and Paolo Gubian
University of Brescia
Via Branze 38, 25123 Brescia (Italy)
email: mario.piccinelli@gmail.com

Abstract — **Nowadays various software tools have been developed for the purpose of creating encrypted volume files. Many of those tools are open source and freely available on the internet. Because of that, the probability of finding encrypted files which could contain forensically useful information has dramatically increased. While decoding these files without the key is still a major challenge, the simple fact of being able to recognize their existence is now a top priority for every digital forensics investigation. In this paper we will present a statistical approach to find elements of a seized filesystem which have a reasonable chance of containing encrypted data.**

Keywords — **Forensics, Anti-anti-forensics, Encryption, Detection.**

I. Introduction

DATA ENCRYPTION is here understood as the process of transforming information using an algorithm (called cypher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key. In computer forensics the encrypted data is usually a part of a filesystem (one or more files) cyphered into a single binary blob, which in turn can be saved as a single file in the main filesystem of the machine it is hosted on or as a whole partition (or a part of it). When a single encrypted file hosts a filesystem it is called volume file, because it mimics a logical volume of a disk.

The data produced by modern encryption software like TrueCrypt or PGP Virtual Disk is usually indistinguishable from uniform random data, and has no recognizable header. This means that it is impossible to link an encrypted volume file to the methodology used to encrypt it, nor it is possible to even prove that it is in fact an encrypted file. This goes under the principle of “Plausible Deniability”, by which it is not possible to prove under a court the existence of hidden data. Nonetheless, experimental evidence proved that there is very low probability for a normal file to have a random distribution of data; under this assumption it can be said that proving a file to be random means proving it has a big chance of being an encrypted volume.

A major problem in recognizing encrypted data by its randomness is that random data can be produced by other means, the most important of those being data wiping algorithms. Data wiping tools in fact destroy data by overwriting the interested area with a random sequence [1], which is as stated before indistinguishable from encrypted data. This is a still open challenge in the analysis of entire disks, entire volumes of a disk, or apparently unused disk space [2]; in these cases it is impossible to prove that these areas contain encrypted data (according to the principle of “Plausible Deniability” described before). For encrypted files, instead, it is impossible to hide their existence as single entities on the disk, and this can lead to the conclusion that they could contain useful data.

II. NIST Statistical Test Suite

The NIST Statistical Test Suite (identified by the description “a statistical test suite for random and pseudo-random number generators for cryptographic applications”) is a software tool written in ANSI C developed by the National Institute for Standards and Technology (U.S. Department of Commerce). It includes 10 pseudo-random number generation algorithms and 15 algorithms for testing randomness of a given data stream. It has been made available in the public domain under an open source license, and can be downloaded from the NIST website with exhaustive documentation. For the experiments mentioned in this paper we used the last release available at the time of writing, version 2.1.1 dated April, 2010. For the purposes of this research only a subset of the testing algorithms will be used. The subset will be chosen according to the size of the file to analyze, because each test has a recommended minimum length (n) in bytes for each run (each sequence is split into a chosen number of runs to be analyzed individually, and each run is n bits long).

A. How the suite is used

Once the package has been compiled a single executable named assess is created. It accepts one integer parameter, the bit stream length n.

```
$ ./assess 32000
```

Once launched, the software asks the user to choose the sequence generator among all the available pseudo random number generators. For our tests we select the option:

```
[0] Input File
```

Then, the software asks to enter the name of the data file to analyze.

```
User Prescribed Input File: _
```

In the next screen the user is asked about which statistical test to run on the selected file. The user can choose between running each test (option 1) or be brought to a next screen in which he'll be able to select a specific subset (option 0).

Whether the user selects all the tests or just a subset, another menu is shown to present the default test-specific parameters, and let the user modify them. The parameters depend mainly on the size n of the streams, and are well described in the NIST manual. They won't be described here because they are outside the scope of this paper. To go on the user has to choose option 0.

At last the user is asked to insert the number of runs of n bit to extract from the selected data source, then how the input file is built.

```
Input File Format:
[0] ASCII - A sequence of ASCII 0's and 1's
[1] Binary - Each byte in data file contains 8 bits of data
```

For our tests the second option will be chosen. Then the test begins.

B. How the suite is used

To be able to validate the detection algorithm described in this paper many test runs had to be performed. To make the process faster, the NIST suite was modified to be able to perform a test without user interaction by passing all the needed parameters from command line. The needed parameters are the size in bits of a run (which is already provided by command line), the number of runs to perform and the input file. With the modified NIST suite an entire test is performed by calling:

```
./assess 32000 250 testfile.ext
```

The above mentioned line tests the file testfile.ext with 250 runs of 32000 bits each.

C. How the tests are performed

This section has been extracted from the NIST Statistical Test Suite manual [3]. Each test is formulated to test a specific null hypothesis (H0), which states that the analyzed sequence is random. Associated with the null hypothesis is the alternative hypothesis (Ha), which is that the sequence is not random. For each test a decision is derived that accepts or rejects the null hypothesis.

For each test, a relevant randomness statistic must be chosen and used to determine the acceptance or rejection of the null hypothesis. Under an assumption of randomness, such a statistic has a distribution of possible values. A theoretical reference distribution of this statistic under the null hypothesis is determined by mathematical methods.

From this reference distribution, a critical value is determined (typically, this value is "far out" in the tails of the distribution, say out at the 99% point). During a test, a test statistic value is computed on the data (the sequence being tested). This test statistic value is compared to the critical value. If the test statistic value exceeds the critical value, the null hypothesis for randomness is rejected. Otherwise, the null hypothesis (the randomness hypothesis) is not rejected (i.e., the hypothesis is accepted).

Each test is based on a calculated test statistic value, which is a function of the data. The test statistic is used to calculate a P-value that summarizes the strength of the evidence against the null hypothesis. For these tests, each P-value is the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested, given the kind of non- randomness assessed by the test. If a P-value for a test is determined to be equal to 1, then the sequence appears to have perfect randomness. A P-value of zero indicates that the sequence appears to be completely non-random. A significance level (α) can be chosen for the tests. If P-value $\geq \alpha$, then the null hypothesis is accepted; i.e., the sequence appears to be random. If P-value $< \alpha$, then the null hypothesis is rejected; i.e., the sequence appears to be non-random. The parameter α denotes the probability of the Type I error (i.e. the probability of rejecting a random sequence), and its default value (which will be used during the following tests) is 0.01, which means that one would expect one sequence in 100 sequences to be rejected by the test if the sequence was random.

D. How the tests are interpreted

The output data from the tests is made up of ASCII text files saved in the directory experiments/AlgorithmTesting/. This directory contains several subdirectories (one for each test) and two general files. Each test-specific subdirectory contains two test-specific files.

Test-specific files:

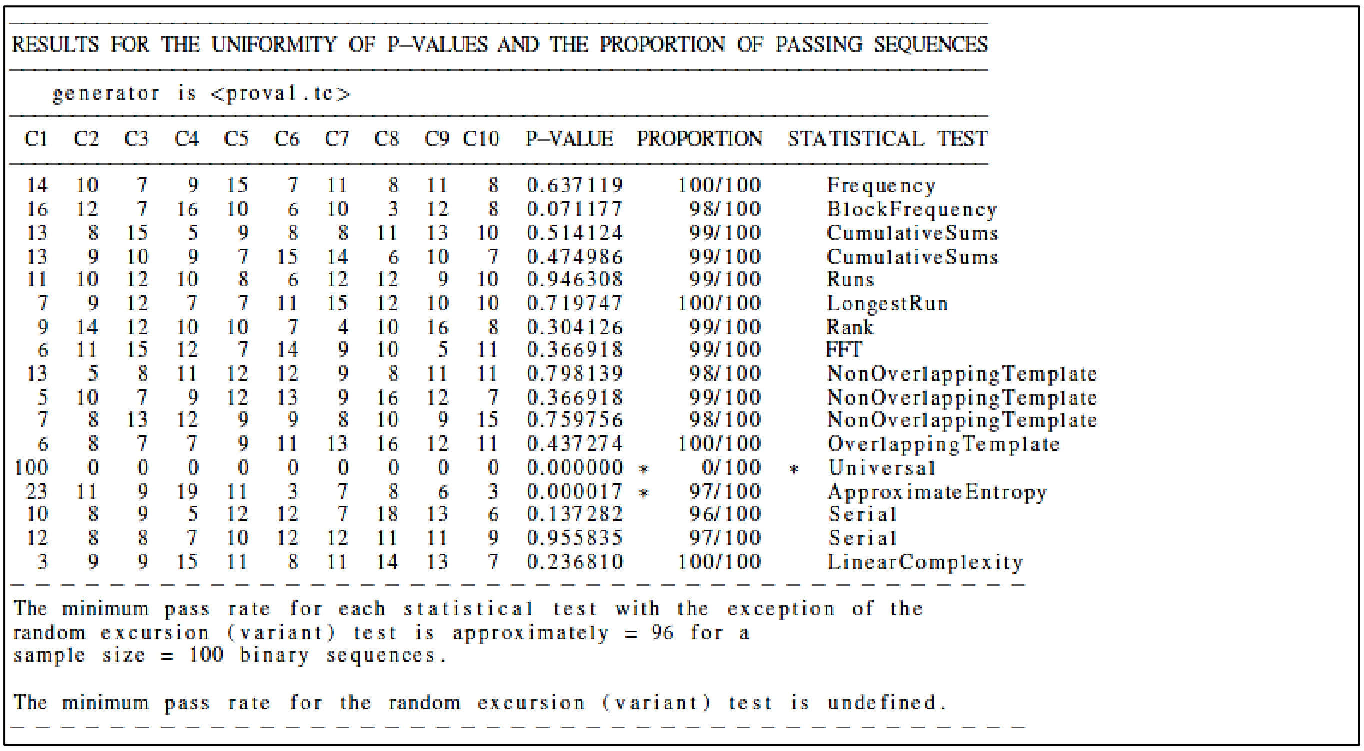
- *results.txt* contains the p-values of the single runs.
- *stats.txt* contains test-specific computational information for each run.

General files:

- *freq.txt* contains the count of 0s and 1s in each run.
- *finalAnalysisReport.txt* is the main result file.

For further analysis the main result file *finalAnalysisReport.txt* will be used. The file has a structure as shown in Listing 1. This file contains a row for each test performed, and shows the results as:

- Columns C1-C10 show the distribution of the p-values. The p-value range (0-1) is split into 10 sub-ranges, and the software counts the number of runs with a p-value included in each (i.e., the column C1 contains the number of tests with a p-value between 0.0 and 0.1).
- Column P-value contains the P-value that arises via the application of a chi-square test, used to assess the uniformity of P-values for each test performed.
- Column Proportion shows the proportion of single runs which passed the test.



```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <proval.tc>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
 14  10   7   9  15   7  11   8  11   8  0.637119    100/100     Frequency
 16  12   7  16  10   6  10   3  12   8  0.071177     98/100     BlockFrequency
 13   8  15   5   9   8   8  11  13  10  0.514124     99/100     CumulativeSums
 13   9  10   9   7  15  14   6  10   7  0.474986     99/100     CumulativeSums
 11  10  12  10   8   6  12  12   9  10  0.946308     99/100     Runs
  7   9  12   7   7  11  15  12  10  10  0.719747    100/100     LongestRun
  9  14  12  10  10   7   4  10  16   8  0.304126     99/100     Rank
  6  11  15  12   7  14   9  10   5  11  0.366918     99/100     FFT
 13   5   8  11  12  12   9   8  11  11  0.798139     98/100     NonOverlappingTemplate
  5  10   7   9  12  13   9  16  12   7  0.366918     99/100     NonOverlappingTemplate
  7   8  13  12   9   9   8  10   9  15  0.759756     98/100     NonOverlappingTemplate
  6   8   7   7   9  11  13  16  12  11  0.437274    100/100     OverlappingTemplate
100   0   0   0   0   0   0   0   0   0  0.000000 *    0/100  *  Universal
 23  11   9  19  11   3   7   8   6   3  0.000017 *   97/100     ApproximateEntropy
 10   8   9   5  12  12   7  18  13   6  0.137282     96/100     Serial
 12   8   8   7  10  12  12  11  11   9  0.955835     97/100     Serial
  3   9   9  15  11   8  11  14  13   7  0.236810    100/100     LinearComplexity
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 96 for a
sample size = 100 binary sequences.

The minimum pass rate for the random excursion (variant) test is undefined.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Listing 1 Example of NIST final analysis report

At the end of the output file the minimum pass rate for each statistical test is shown, determined using the confidence interval defined as:

$$CI = \hat{p} \pm 3\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

where $\hat{p}$ = 1-α and n is the sample size. For example, if α = 0.01 and n = 1000, the confidence interval is:

$$CI = 0.99 \pm 3\sqrt{\frac{0.99 \cdot 0.01}{1000}} = 0.99 \pm 0.009439$$

which means the proportion should lie above 0.98056. When a test fails this condition (or the condition provided by the uniformity condition) a star (*) symbol is inserted near the failing value.

III. Detecting Encrypted Files

As stated before, there is no simple mean of detecting files encrypted using modern cyphering tools such as Truecrypt. The files have no useful header section and present no useful extension. Moreover, the extension of encrypted files can be easily modified to make the file appear like a normal system file. It is not uncommon for Truecrypt files to have their name changed to something system-like, such as system.dll (on Windows systems), and be placed in unusual positions on the filesystem (as in the windows/system directory among many other .dll files). In such cases it is impossible to detect these files at first glance or with a superficial analysis of the disk, and deeper methods must be employed.

A. Detecting suspicious files

A statistical analysis on each file in a computer could take a huge amount of time because of the high number of files in a normal system. For a first analysis it could be useful to be able to detect suspicious files, i.e. files which appear to be something different of what they are supposed to be. The next sections outline some simple methods for the identification of the most interesting target files for a first analysis.

File size: If a file is used to hide an encrypted volume it has to be big enough to contain the data. Under this assumption, it is unlikely to find a small file used to hide an encrypted volume. The first files to be checked on an acquired file system should be the bigger ones.

File extension: In normal conditions the extension of a file identifies the file type and so the software which should be able to handle it. For example, a .jpg file is supposed to be an image, and so it should be read by any software able to handle that kind of images. A big file with a known extension but which can't be opened by the right software is suspicious. This means the first thing to do on a copy of a seized filesystem is try to open all the files with known extensions to assess whether they are what they look like or not.

File type via header: In normal conditions all files are identified by their header, the first part of a file which contains information about the file itself. Known file types are identified by their header data, which should match with their extension (a .jpg file should present the .jpg header data) [4] [5]. A mismatch between file header and file extension (or having a known extension on a header less file) is suspicious.

It should be noted that this methodology could be disrupted using software tools able to change header information of a file or add a known header to a header less file, such as the Transmogrify tool [6].

File content via know hash: Known operating system and program installation files can be deemed not important (and thus not need to be further analyzed) if it can be proved that their content is not different from what is expected in a non-manipulated case. To verify that it is possible to check them (for example by their MD5 hash) versus the same files in a reference system. Another way is to check their MD5 hashes against a database of known hashes, such as the one provided by the HashKeeper tool [7] provided free of charge by the National Drug Intelligence Center, a component of the Department of Justice of the United States.

B. Analysing suspicious files

Once a file has been deemed for further analysis (or, at least, each file which is not clearly recognized as forensically useless), the statistical analysis with the NIST Statistical Test Suite can be performed to assess whether the file data is random and thus it can be recognized with a certain probability as an encrypted volume file.

The tests to be performed on the data must be chosen according to the size of the data itself, according to the minimum n values recommended for each test. Some tests require a big value for n, such as the Random Excursion tests (see appendix A-N and A-O) requiring at least one million bits. Testing sequences with a high value of n can require a large amount of time, so it's preferable to select lower values and choose the tests list accordingly.

Each file is split into 1 MB blocks, and for each block the NIST test is performed for 250 runs of 4 KB each (n = 320,000). These values have been chosen after extensive testing as a good tradeoff between run size and test duration.

Once a block has been tested a report like the one in listing 1 is produced for further interpretation.

For the results shown in the following chapters all tests have been considered except for the Binary Matrix Rank Test, the Random Excursion Test and the Random Excursion Variant Test, which require a value of n greater than 32000 to be considered reliable.

C. Interpretation of analysis results

After the analysis is completed by the NIST tool, results in the output file have to be interpreted to discriminate whether the sequence can be considered random or not.

To choose whether the file is random or not we used the following algorithm:

- for each block (1MB):
  - for each kind of test performed on the selected block:
    - if the test is run only one time: the test is passed if both p-value and proportion are deemed passed by the NIST suite.
    - if the test is run many times: the test is passed if at least 90% of the times it is deemed passed by the NIST suite.
  - The block is deemed passed if at least 70% of the test kinds performed are passed.
- The whole file is deemed passed (and thus random) if at least 70% of the blocks are passed.

To achieve this result we defined three thresholds. These thresholds have been determined by experiments with various samples of "normal" data and data produced by cyphering algorithms.

The algorithm has been implemented as a Python script which receives as input the *finalAnalysisReport.txt*'s of all the blocks of a single file concatenated.

D. Example of interpretation

As an example to illustrate the algorithm we decide to test a 50 MB file. The file is split into 50 blocks of 1 MB each, and on each block the NIST test is performed with 250 runs of 4 KB each. We assume the first test gives an output like this:

```
- Test: 49, lines: 188, passed: 160
- test: Frequency, passed: 1/1 (PASS)
- test: BlockFrequency, passed: 1/1 (PASS)
- test: CumulativeSums, passed: 2/2 (PASS)
- test: Runs, passed: 1/1 (PASS)
- test: LongestRun, passed: 1/1 (PASS)
- test: Rank, passed: 1/1 (PASS)
- test: FFT, passed: 1/1 (PASS)
- test: NonOverlappingTemplate, passed: 148/148 (PASS)
- test: OverlappingTemplate, passed: 1/1(PASS)
- test: Universal, passed: 0/1 (FAIL)
- test: ApproximateEntropy, passed: 0/1(FAIL)
- test: Serial, passed: 2/2 (PASS)
- test: LinearComplexity, passed: 1/1(PASS)
Tests passed: 11/13 (84%)
PASSED
```

In the previous listing it is shown that 188 tests of 13 different kinds have been performed on the block. The reason for this incongruity is that some tests are performed many times (such as the non-overlapping template test, which is performed 148 times, each time with a different test template). To keep all the tests on the same level of importance it has been decided not to count every occurrence of them, but to deem the kind "passed" if at least 90% of the occurrences are passed. This way the non-overlapping template test, run 148 times, still counts as one "passed".

After all the test results are analyzed the percentage of passed tests against the number of tests is calculated. In the example this percentage is 84%, over the 70% threshold, so the whole block is deemed passed.

After testing all the 1 Mb blocks in the file, a final statistic is calculated:

Final results: 49/50 blocks passed
PASSED

If at least 70% of the blocks is deemed passed, then the whole file can be considered random.

IV. Test cases

To test the proposed detection algorithm we chose the most widely used open source encryption software, TrueCrypt, to create some test cases. The results are validated by testing TrueCrypt volume files and volumes versus standard files from a reference repository.

A. TrueCrypt files

Using TrueCrypt version 7.0a on Mac Os X we created three encrypted volume files of 50 MB each, one with no data, one half full and one full. Results are shown in Table I. TrueCrypt files have no header and are recognized as random data for all the file size (both encrypted data and empty space are equally random).

TABLE I
Test results for TrueCrypt volumes

File	Blocks	Random blocks	Nonrandom blocks	Perc.
Empty volume	50	49	1	98%
Half full volume	50	49	1	98%
Full volume	50	50	0	100%

B. TrueCrypt partitions

A TrueCrypt encrypted partition was created on a USB mass storage of 256 MB using TrueCrypt version 7.0a on Mac Os X. This partition was then dumped to a file using EnCase v. 4.20 for Windows. The file was then split into chunks of 50 MB each, and the above described analysis was performed on each part. The results are shown in Table II. It is shown that all the parts are correctly recognized as random data.

TABLE II
Test results for TrueCrypt volume files

File	Blocks	Random blocks	Nonrandom blocks	Perc.
Chunk 1	50	48	2	96%
Chunk 2	50	49	1	98%
Chunk 2	50	47	3	94%
Chunk4	50	50	0	100%
Chunk 5	50	47	3	94%

C. Standard files

In order to test the classifier we looked on the Internet for a publicly available repository of standard files, here intended as files more likely to be found on a computer (which should be classified nonrandom). We found a repository named Digital Forensics Corpora[i] which was set in order to have a database that can be used for research purposes. From that repository we downloaded directory 000 and chose some of the files larger than 5 MB to test. Results are shown in Table III. It is shown that all files but one are clearly recognized as nonrandom.

TABLE III
Test results for standard files

File	Blocks	Random blocks	Nonrandom blocks	Perc.
000033.xls	6	0	6	0%
000030.xls	8	0	8	0%
000113.doc	14	0	14	0%
000134.ppt	9	0	9	0%
000143.pdf	5	0	5	0%
000187.pdf	9	6	3	66%
000208.pdf	6	0	6	0%
000559.ppt	17	0	17	0%
000564.csv	8	0	8	0%
000736.gz	6	0	6	0%
000766.ps	5	0	5	0%
000801.doc	6	0	6	0%
000938.txt	29	0	29	0%

V. Similar tools found in literature

In literature we found a little number of tools which claim to be able to detect cyphered files. The most interesting tools are:

- FI Tools from Forensics Innovations
- TCHunt from 16 Systems

Interestingly all their developers agree on the fact that it is impossible to accurately detect encrypted files from random files, because they appear identical on every analysis. It is remarked that the only method to provide some sort of detection is to identify files containing random data [8] [9].

TCHunt uses another three file attributes to try to detect TrueCrypt files:

- The suspect file size modulo 512 must equal zero, because TrueCrypt files are built from 512 bytes blocks.
- The suspect file size is at least 19 KB in size, because this is the minimum dimension for a TrueCrypt volume file.
- The suspect file must not contain a common file header.

After performing some tests it appears that these tools have almost the same success rate of the methodology explained, because they work on the same hypothesis.

VI. Conclusions

This paper was motivated by the lack of open source forensically sound tools to provide some sort of detection of encrypted volume files. While it was known from the beginning that a true detection of this sort of archives is not possible, a methodology was developed to identify filesystem elements which with high probability contain encrypted data.

The methodology was tested against a number of test cases which proved it to be reliable for identifying data encrypted with a popular encryption tool. The next step in this research work will be to provide an integrated software tool which could be used by both researchers and practitioners in digital forensics to easily scan a filesystem and identify realistic candidates for further cryptographic examination.

APPENDIX A
TESTS IN THE NIST SUITE

This appendix reports a brief description of the statistical tests used in the NIST Suite. The descriptions are taken from the NIST Statistical Test Suite manual [3].

A. Frequency (Monobit) Test

The purpose of this test is to determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence. The test assesses the closeness of the fraction of ones to 0.5, that is, the number of ones and zeroes in a sequence should be about the same.

It is recommended that each sequence to be tested consist of a minimum of 100 bits (i.e., $n \geq 100$).

B. Frequency Test within a block

The purpose of this test is to determine whether the frequency of ones in an M-bit block is approximately M/2, as would be expected under an assumption of randomness. For block size M=1, this test degenerates to test 1, the Frequency (Monobit) test.

It is recommended that each sequence to be tested consist of a minimum of 100 bits (i.e., $n \geq 100$). Note that $n \geq MN$. The block size M should be selected such that $M \geq 20$, $M > .01*n$ and $N < 100$.

C. Run test

The purpose of the runs test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. A run of length k consists of exactly k identical bits and is bounded before and after with a bit of the opposite value. In particular, this test determines whether the oscillation between such zeros and ones is too fast or too slow.

It is recommended that each sequence to be tested consist of a minimum of 100 bits (i.e., $n \geq 100$).

D. Test for the Longest Run of Ones in a Block

The purpose of this test is to determine whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones that would be expected in a random sequence.

The recommended length of the sequence to be tested is $n \geq 128$. According to this dimension, the length M of the blocks is chosen as follows:

Minimum n	**M**
128	8
6272	128
750000	10^4

E. Binary Matrix Rank Test

The purpose of this test is to check for linear dependence among fixed length substrings of the original sequence, by calculating the rank of disjoint sub-matrices of the entire sequence.

The probabilities for $M = Q = 32$ (where M is the number of rows in each matrix, and Q the number of columns) have been calculated and inserted into the code. Other choices of M and Q may be selected, but the probabilities would need to be calculated. The minimum number of bits to be tested must be such that $n \geq 38MQ$ (i.e., at least 38 matrices are created). For $M = Q = 32$, each sequence to be tested should consist of a minimum of 38,912 bits.

F. Discrete Fourier Transform (Spectral) Test

The purpose of this test is to detect periodic features (i.e., repetitive patterns that are close to each other) in the tested sequence that would indicate a deviation from the assumption of randomness. The intention is to detect whether the number of peaks exceeding the 95% threshold is significantly different than 5%.

It is recommended that each sequence to be tested consist of a minimum of 1000 bits (i.e., $n \geq 1000$).

G. Non-overlapping Template Matching Test

The purpose of this test is to detect sequences with too many occurrences of a given non-periodic (aperiodic) pattern. An m-bit window is used to search for a specific m-bit pattern. If the pattern is not found, the window slides one bit position. If the pattern is found, the window is reset to the bit after the found pattern, and the search resumes.

The test code has been written to provide templates for $m = 2, 3,...,10$. It is recommended that $m = 9$ or $m = 10$ be specified to obtain meaningful results. Although $N = 8$ has been specified in the test code, the code may be altered to other sizes. However, N should be chosen such that $N \geq 100$ to be assured that the P-values are valid. Additionally, be sure that $M > 0.01 \geq n$ and $N = \text{floor}(n/M)$.

H. Overlapping Template Matching Test

Both this test and the Non-overlapping Template Matching (section A-G) test use an m-bit window to search for a specific m-bit pattern. As with the test in A-G, if the pattern is not found, the window slides one bit position. The difference between this test and the test in section A-G is that when the pattern is found, the window slides only one bit before resuming the search.

The values of K, M and N have been chosen such that each sequence to be tested consists of a minimum of 106 bits (i.e., $n \geq 106$). Various values of m may be selected, but for the time being, NIST recommends $m = 9$ or $m = 10$.

I. Maurers Universal Statistical Test

The purpose of the test is to detect whether or not the sequence can be significantly compressed without loss of information by evaluating the number of bits between matching patterns. A significantly compressible sequence is considered to be non-random.

This test requires a long sequence of bits ($n \geq (Q+K)L$) which are divided into two segments of L-bit blocks, where L should be chosen so that $6 \leq L \leq 16$. The first segment consists of Q initialization blocks, where Q should be chosen so that $Q = 10 \cdot 2^L$. The second segment consists of K test blocks, where$K = \lceil n/L \rceil - Q \approx 1000 \cdot 2^L$. The values of L, Q and n should be chosen as follows:

n	L	$Q = 10 \cdot 2^L$
$\geq$ 387840	6	640
$\geq$ 904960	7	1280
$\geq$ 2068480	8	2560
$\geq$ 4654080	9	5120
…	…	…

J. *Linear Complexity Test*

The purpose of this test is to determine whether or not the sequence is complex enough to be considered random by evaluating the length of a linear feedback shift register (LFSR). Random sequences are characterized by longer LFSRs. An LFSR that is too short implies non-randomness.

It is recommended that $n \geq 106$. The value of M must be in the range $500 \leq M \leq 5000$, and $N \geq 200$.

K. *Serial Test*

The purpose of this test is to determine whether the number of occurrences of the 2m m-bit overlapping patterns is approximately the same as would be expected for a random sequence. Random sequences have uniformity; that is, every m-bit pattern has the same chance of appearing as every other m-bit pattern. Note that for m = 1, the Serial test is equivalent to the Frequency test.

It is recommended to choose m and n such that$m < \lfloor \log_2 n \rfloor - 2$.

L. *Approximate Entropy Test*

The purpose of the test is to compare the frequency of overlapping blocks of two consecutive/adjacent lengths (m and m+1) against the expected result for a random sequence.

It is recommended to choose m and n such that$m < \lfloor \log_2 n \rfloor - 5$.

M. *Cumulative Sums (Cusum) Test*

The focus of this test is the maximal excursion (from zero) of the random walk defined by the cumulative sum of adjusted (-1, +1) digits in the sequence. The purpose of the test is to determine 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. This cumulative sum may be considered as a random walk. For a random sequence, the excursions of the random walk should be near zero.

It is recommended that each sequence to be tested consist of a minimum of 100 bits (i.e., $n \geq 100$).

N. *Random Excursions Test*

The focus of this test is the number of cycles having exactly K visits in a cumulative sum random walk. The cumulative sum random walk is derived from partial sums after the (0, 1) sequence is transferred to the appropriate (-1, +1) sequence. A cycle of a random walk consists of a sequence of steps of unit length taken at random that begin at and return to the origin. The purpose of this test is to determine if the number of visits to a particular state within a cycle deviates from what one would expect for a random sequence. This test is actually a series of eight tests (and conclusions), one test and conclusion for each of the states: -4, -3, -2, -1 and +1, +2, +3, +4.

It is recommended that each sequence to be tested consist of a minimum of 1,000,000 bits (i.e., $n \geq 10^6$).

O. *Random Excursions Variant Test*

The focus of this test is the total number of times that a particular state is visited (i.e., occurs) in a cumulative sum random walk. The purpose of this test is to detect deviations from the expected number of visits to various states in the random walk. This test is actually a series of eighteen tests (and conclusions), one test and conclusion for each of the states: -9, -8, ..., -1 and +1, +2, ..., +9.

It is recommended that each sequence to be tested consist of a minimum of 1,000,000 bits (i.e., $n \geq 10^6$).

REFERENCES

[1] A. Savoldi, M. Piccinelli, and P. Gubian. *A statistical method for detecting on-disk wiped areas*. Digital Investigation, Elsevier, Volume 8, 2012.

[2] A. Czeskis, D.J.St. Hilaire, T. Kohno, K. Koscher, S.D. Gribble, and B. Schneier. *Defeating Encrypted and Deniable File Systems: TrueCrypt v5.1a and the Case of the Tattling OS and Applications*. Retrieved January, 2011, from http://pdos.csail.mit.edu/6.858/2010/readings/truecrypt.pdf.

[3] D. Banks, E. Barker, J. Dray, A. Heckert, S. Leigh, M. Levenson, J. Nechvatal, A. Rukhin, M. Smid, J. Soto, M. Vangel, and S. Vo. *NIST Statistical Test Suite*, 2008. Retrieved January, 2011, from http://csrc.nist.gov/groups/ST/toolkit/rng/documents/sts-2.1.zip.

[4] D.J. Hickok, D.R. Lesniak, and M.C. Rowe. *File Type Detection Technology, 2005*. Retrieved January, 2011, from http://www.uwplatt.edu/csse/courses/prev/csse411-materials/StudentConferencePublications/MICS2005 File Type Detection Technology.pdf.

[5] C. Sadowski and G. Levin. *SimHash: Hash-based Similarity Detection*, 2007. Retrieved January, 2011, from http://simhash.googlecode.com/svn/trunk/paper/SimHashWithBib.pdf.

[6] B. Blunden. *Anti-Forensics: The Rootkit Connection*. In Black Hat USA 2009 Conference proceedings, 2009. Retrieved January, 2011, from http://www.blackhat.com/presentations/bh-usa-09/BLUNDEN/BHUSA09-Blunden-AntiForensics-PAPER.pdf.

[7] *Hashkeeper web site*. Retrieved on February, 2011 from http://www.justice.gov/ndic/domex/hashkeeper.htm.

[8] *Comments from president of Forensic Innovations, Inc. Rob Zirnstein on FI blog post TrueCrypt is now detectable*. Retrieved January, 2011, from http://www.forensicinnovations.com/blog/?p=7.

[9] *TCHunt FAQs from 16 System website*. Retrieved on February 14th, 2011 from http://16s.us/TCHunt/faq/.

[i] http://domex.nps.edu/corp/files/govdocs1/