AN4230
Application note

STM32 microcontroller random number generation validation using the NIST statistical test suite

Introduction

Many standards created requirements and references for the construction, the validation and the use of random number generators (RNGs), in order to verify that the output they produce is indeed random.

This application note provides some guidelines to verify the randomness of the numbers generated by the RNG peripheral embedded in a selection of STM32 microcontrollers (MCUs) listed in the table below. This verification is based either on the statistical test suite (STS) SP 800-22rev1a (April 2010) or SP 800-90b (January 2018) of the NIST (National Institute of Standards and Technology).

This document is structured as follows:

- a general introduction to STM32 microcontroller random number generator (see *Section 1*)
- the NIST SP800-22b test suite (see *Section 2*)
- the steps needed to run NIST SP800-22b test and analysis (see *Section 3*)
- the NIST SP800-90b test suite (see *Section 4*)
- the steps needed to run NIST SP800-90b test and analysis (see *Section 5*)

Table 1. Applicable products

Type	Products	
	Checked with SP800-22rev1a	**Checked with SP800-90b**
Microcontrollers	STM32F2 Series, STM32F4 Series, STM32F7 Series STM32H7 Series STM32L0 Series, STM32L4 Series, STM32L4+ Series	STM32L5 Series

Contents

1 STM32 MCU RNG 5

1.1 Introduction 5

1.2 STM32 MCU implementation description 5

2 NIST SP800-22b test suite 7

2.1 Introduction 7

2.2 NIST SP800-22b test suite description 7

3 NIST SP800-22b test suite running and analyzing 9

3.1 Firmware description 9

3.1.1 STM32 MCU side 9

3.1.2 On the NIST SP800-22b test suite side 9

3.2 NIST SP800-22b test suite steps 10

3.2.1 Step 1: random number generator 10

3.2.2 Step 2: NIST statistical test 10

3.2.3 Step 3: test report 15

4 NIST SP800-90b test suite 16

4.1 Introduction 16

4.2 NIST SP800-90b test suite description 16

4.2.1 Non-IID track: entropy estimation for non-IID data 16

5 NIST SP800-90b test suite running and analyzing 18

5.1 Firmware description 18

5.1.1 STM32 MCU side 18

5.1.2 NIST SP800-90b test suite side 18

5.2 NIST SP800-90B test suite steps 18

5.2.1 Step 1: random number generator 18

5.2.2 Step 2: NIST statistical tests 19

5.2.3 Step 3: test report 19

6 Conclusion 20

Appendix A NIST SP800-22b statistical test suite 21

Appendix B NIST SP800-90b statistical test suite 24

Revision history 26

List of figures

Figure 1. STM32 true RNG block diagram . . . 6
Figure 2. Block diagram of deviation testing of a binary sequence from randomness based on NIST test suite . . . 10
Figure 3. Main sts-2.1.1 screen . . . 11
Figure 4. File input screen . . . 11
Figure 5. Statistical test screen . . . 12
Figure 6. Parameter adjustment screen . . . 12
Figure 7. Bitstreams input . . . 13
Figure 8. Input file format . . . 13
Figure 9. Statistical testing in progress . . . 14
Figure 10. Statistical testing complete . . . 14

1 STM32 MCU RNG

1.1 Introduction

Random number generators (RNGs) used for cryptographic applications typically produce sequences made of random 0's and 1's bits.

There are two basic classes of random number generators:

- Deterministic RNG or pseudo RNG (**PRNG**)

  A deterministic RNG consists of an algorithm that produces a sequence of bits from an initial value called a seed. To ensure forward unpredictability, care must be taken in obtaining seeds. The values produced by a PRNG are completely predictable if the seed and generation algorithm are known. Since in many cases the generation algorithm is publicly available, the seed must be kept secret and generated from a TRNG.

- Non-deterministic RNG or True RNG (**TRNG**)

  A non-deterministic RNG produces randomness that depends on some unpredictable physical source (the entropy source) outside of any human control.

The RNG hardware peripheral implemented in some STM32 MCUs is a true random number generator.

1.2 STM32 MCU implementation description

The table below lists the STM32 Arm®(a) core-based MCUs that embed the RNG peripheral.

Table 2. STM32 lines embedding the RNG hardware peripheral

Series	STM32 lines
STM32F2 Series	STM32F2x5, STM32F2x7
STM32F4 Series	STM32F405/415, STM32F407/417, STM32F410, STM32F427/437, STM32F429/439, STM32F469/479
STM32F7 Series	STM32F7x5, STM32F7x6
STM32L0 Series	STM32L05x, STM32L06x, STM32L072/073
STM32L4 Series	STM32L4x6
STM32L4+ Series	All lines
STM32H7 Series	STM32H742, STM32H743/753, STM32H745/755, STM32H747/757, STM32H750 Value line
STM32L5 Series	STM32L5x2

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

The true RNG implemented in the STM32 MCUs is based on an analog circuit. This circuit generates a continuous analog noise that is used in the RNG processing to produce a 32-bit random number.

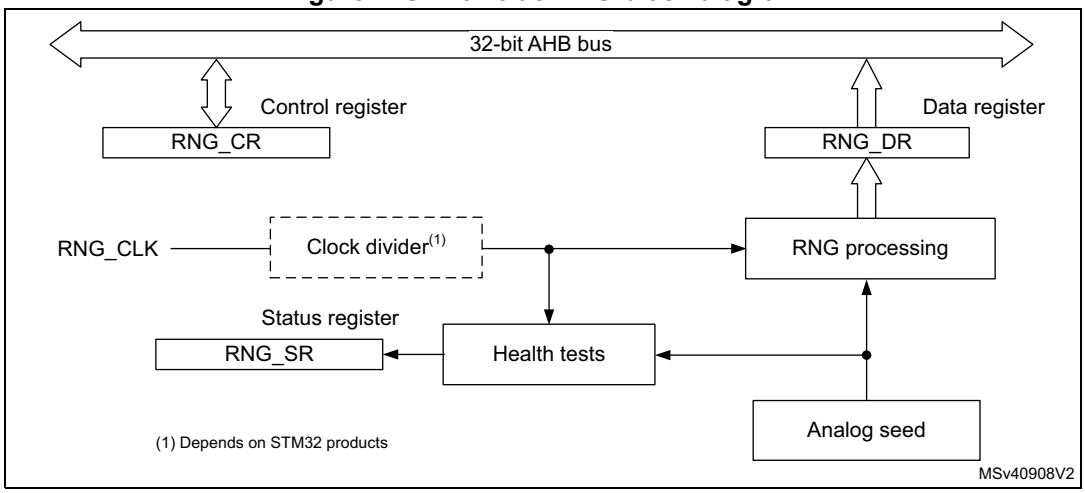
The analog circuit is made of several ring oscillators whose outputs are XORed.

The RNG processing is clocked by a dedicated clock at a constant frequency and, for a subset of microcontrollers, the RNG dedicated clock can be reduced using the divider inside the RNG peripheral.

For more details about the RNG peripherals, refer to the STM32 reference manuals.

The figure below shows a simplified view of a true RNG in STM32 microcontrollers.

Figure 1. STM32 true RNG block diagram



2 NIST SP800-22b test suite

2.1 Introduction

The NIST SP800-22b statistical test suite is used to probe the quality of RNGs for cryptographic applications. A comprehensive description of the suite is presented in the NIST document entitled *A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications*.

2.2 NIST SP800-22b test suite description

The NIST SP800-22b statistical test suite “sts-2.1.1” is a software package developed by NSIT that can be downloaded from the NIST web site (search for *download the NIST Statistical Test Suite* at csrc.nist.gov).

The source code has been written in ANSI C. The NIST statistical test suite consists of 15 tests that verify the randomness of a binary sequence. These tests focus on various types of non-randomness that can exist in a sequence.

These test can be classified as follows:

- **Frequency tests**
  - Frequency (Monobit) test

    To measure the distribution of 0’s and 1’s in a sequence and to check if the result is similar to the one expected for a truly random sequence.
  - Frequency test within a block

    To check whether the frequency of 1’s in a M-bit block is approximately M/2, as expected from the theory of randomness.
  - Run tests

    To assess if the expected total number of runs of 1’s and 0’s of various lengths is as expected for a random sequence.
  - Test of the longest run of 1’s in a block

    To examine the long runs of 1’s in a sequence.
- **Test of linearity**
  - Binary matrix rank test

    To assess the distribution of the rank for 32x32 binary matrices.
  - Linear complexity test

    To determine the linear complexity of a finite sequence.
- **Test of correlation** (by means of Fourier transform)
  - Discrete Fourier transform (spectral) test

    To assess the spectral frequency of a bit string via the spectral test based on the discrete Fourier transform. It is sensitive to the periodicity in the sequence.
- **Test of finding some special strings**
  - Non-overlapping template matching test

    To assess the frequency of m-bit non-periodic patterns.
  - Overlapping template matching test

    To assess the frequency of m-bit periodic templates

- **Entropy tests**
  - Maurer's "Universal Statistical" test

    To assess the compressibility of a binary sequence of L-bit blocks.
  - Serial test

    To assess the distribution of all 2^m m-bit blocks.

Note: For m = 1, the serial test is equivalent to the frequency test of Section 2.2.

  - Approximate entropy test

    To assess the entropy of a bit string, comparing the frequency of all m-bit patterns against all (m+1)-bit patterns.

- **Random walk tests**
  - Cumulative sums (Cusums) test

    To assess that the sum of partial sequences is not too large or too small; it is indicative of too many 0's or 1's.
  - Random excursion test

    To assess the distribution of states within a cycle of random walk.
  - Random excursion variant test

    To detect deviations from the expected number of visits to different states of the random walk.

Each of the above tests is based on a calculated test statistic value, that is a function of the testing sequence.

The test statistic is used to calculate a **Pvalue,** which is the probability that a perfect random number generator generated a sequence less random than the sequence that was tested.

For more details about the NIST statistical test suite, refer to the following NIST document available on the NIST web site: *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications" Special Publication 800-22 Revision 1a*.

3 NIST SP800-22b test suite running and analyzing

3.1 Firmware description

To run the NIST statistical test suite as described in the previous section, two firmware are needed, one on the STM32 microcontroller side and one on the NIST SP800-22b test suite side.

3.1.1 STM32 MCU side

The firmware package is provided upon request. For more details, contact the local ST sales representative.

This program allows random numbers generation, using the STM32 RNG peripheral. It also retrieves these numbers on a workstation for testing with the NIST statistical test suite.

Each firmware program is used to generate 10 64-Kbyte blocks of random numbers. The output file contains 5,120,000 random bits to be tested with the NIST statistical test.

As recommended by the NIST statistical test suite, the output file format can be one of the followings:

- a sequence of ASCII 0's and 1's if the *FILE_ASCII_FORMAT* Private define is uncommented in the *main.c* file
- A binary file of random bytes if the *FILE_BINARY_FORMAT* Private define is uncommented in the *main.c* file.

Form more details about the program description and settings, refer to the *readme* file inside the firmware package.

Note: The USART configuration can be changed via the `SendToWorkstation()` *function in the main.c file.*

The output values can be changed by modifying the Private define in the main.c file as follows:

```
#define NUMBER_OF_RANDOM_BITS_TO_GENERATE   512000
#define BLOCK_NUMBER 10
```

3.1.2 On the NIST SP800-22b test suite side

Downloaded on a workstation, the NIST statistical test suite package sts-2.1.1 verifies the randomness of the output file of the STM32 RNG peripheral.

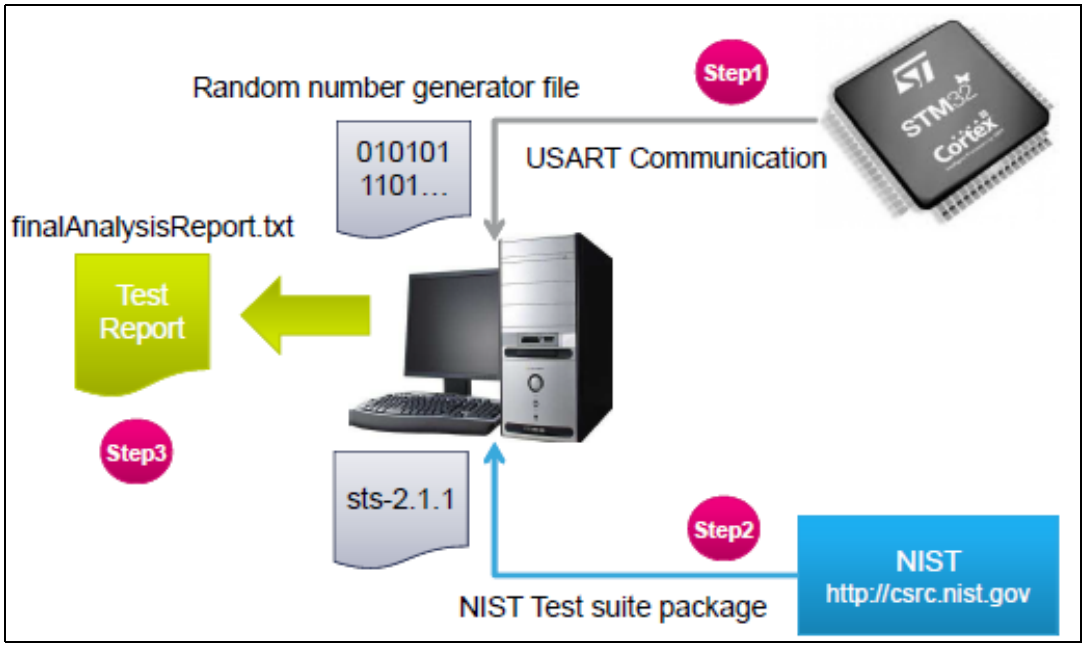
The generator file to be analyzed must be stored under the *data* folder (sts-2.1.1\data).

For more details about how the NIST statistical tests work, refer to section ‘How to get started in the NIST document *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*.

3.2 NIST SP800-22b test suite steps

The figure below describes the steps needed to verify the randomness of an output number generated by STM32 MCUs using the NIST statistical test suite package sts-2.1.1.

Figure 2. Block diagram of deviation testing of a binary sequence from randomness based on NIST test suite



3.2.1 Step 1: random number generator

Connect the STM32 board to the workstation. Depending on the type of board, the connection is made as follows:

- via a null-modem female/female RS232 cable
- via a USB Type-A to Mini-B cable

The STM32 RNG is run via the UART firmware in order to generate a random number as described in *Section 3.1.1: STM32 MCU side*. Data are stored on the workstation using a terminal emulation application such as a PuTTY (free and open-source terminal emulator, serial console and network file transfer application).

3.2.2 Step 2: NIST statistical test

The sts-2.1.1 package is compiled as described in the NIST statistical test suite documentation in order to create an executable program using visual C++ compiler.

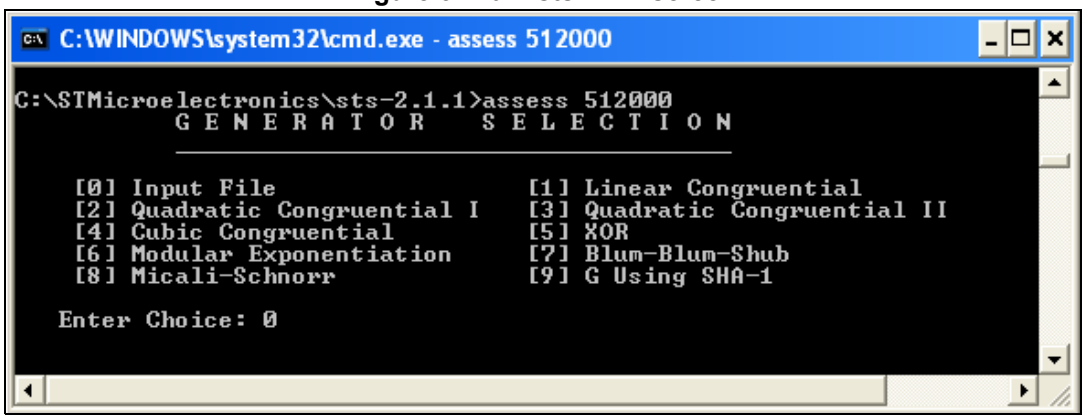
After running the NIST statistical test suite program, a series of menu prompts are displayed in order to select the data to be analyzed and the statistical tests to be applied.

In this application note, the NIST statistical test suite is compiled under the name *assess.exe* and saved under the *NIST_Test_Suite_OutputExample* folder. As described previously, the random number is defined as *512,000 bits per block*.

The various steps are detailed as follows:

1. The first screen that appears is shown below.

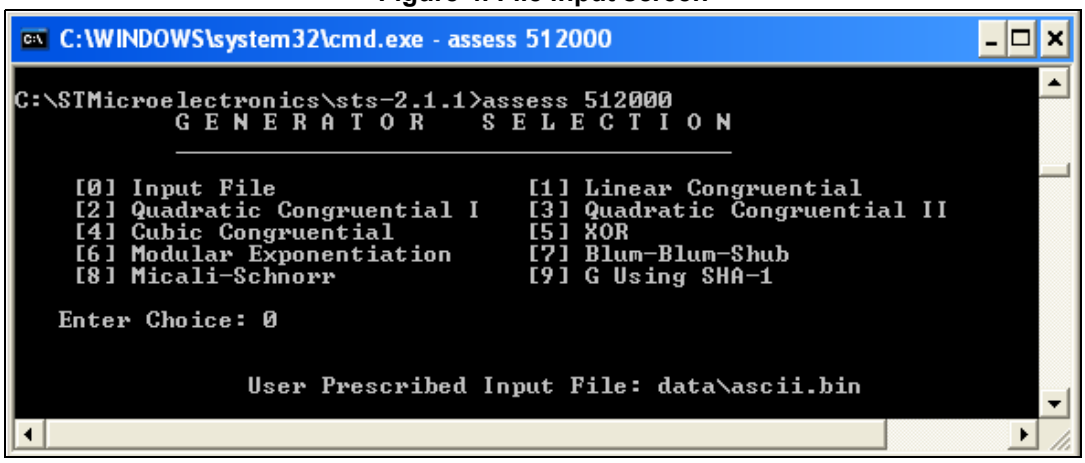
Figure 3. Main sts-2.1.1 screen



When value 0 is entered, the program requires to enter the file name and path of the random number to be tested.

2. The second screen is shown below.

Figure 4. File input screen

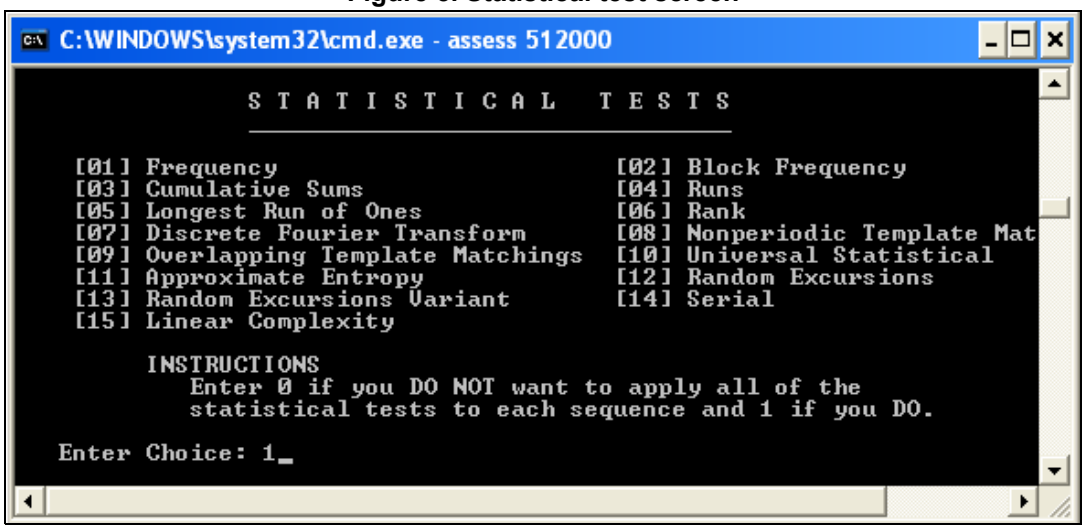


This application note details an example with two files per series, generated with the STM32 RNG, with the following file formats as recommended by NIST:

– *ascii.bin:* sequence of ASCII 0's and 1's
– *binary.bin*: each byte in the data file contains 8 bits of data

3. The NIST statistical test suite displays 15 tests that can be run via the screen shown below.

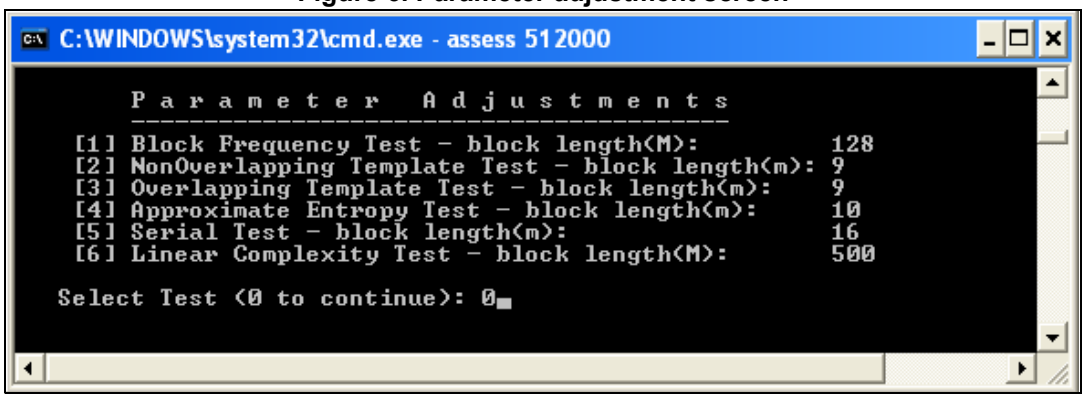
Figure 5. Statistical test screen



In this case, 1 has been selected to apply all of the statistical tests.

4. The parameter adjustments ca be done in the screen shown below.

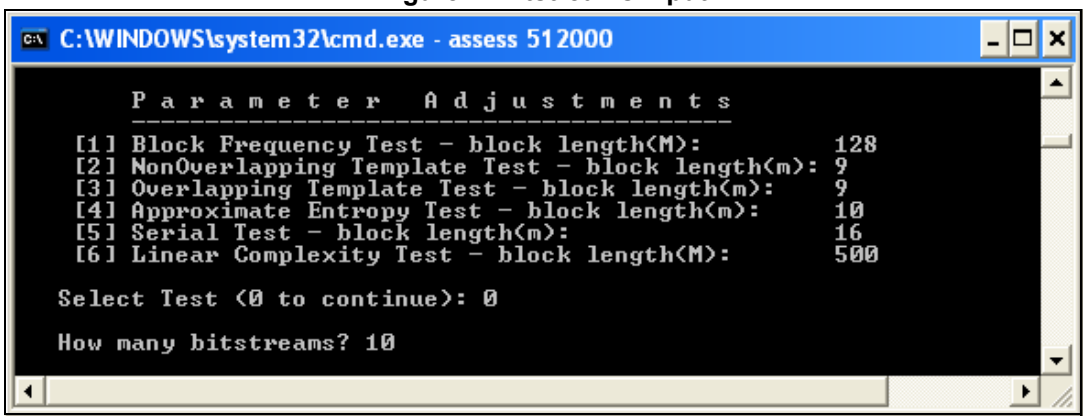
Figure 6. Parameter adjustment screen



In this example, the default settings are kept and value 0 is selected to go to the next step.

5. The user needs to provide the number of bitstreams.

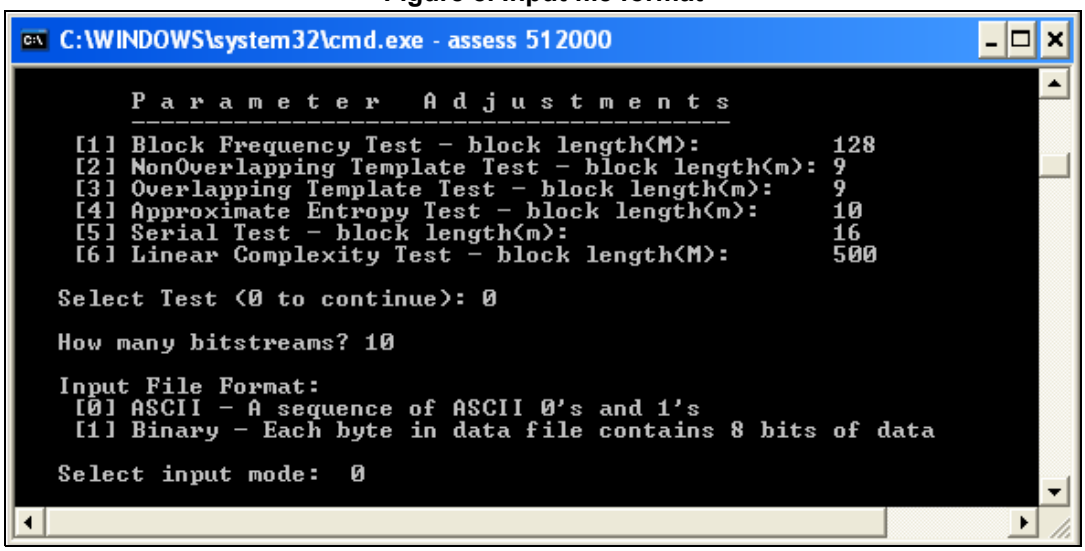
Figure 7. Bitstreams input



The NIST statistical test suite requires to put the number of bitstreams: 10 is entered for this example, meaning 10 blocks of 512 Kbits are selected (5,12 Mbits).

6. The user must then specify whether the file consists of bits stored in ASCII format or hexadecimal strings stored in a binary format using the following screen.

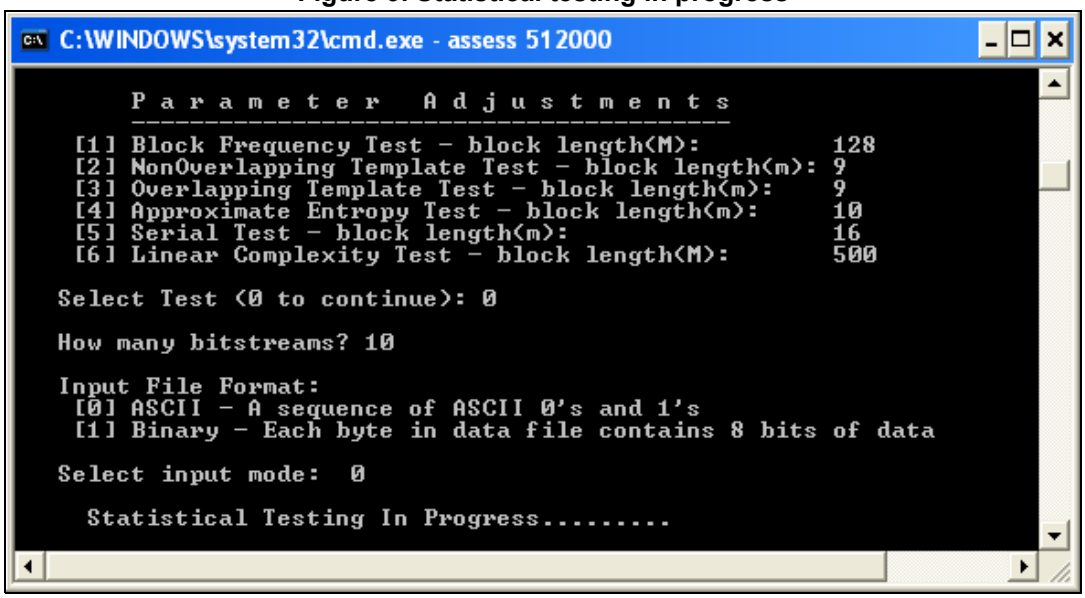
Figure 8. Input file format



Value 0 is selected because the file is in ASCII format.

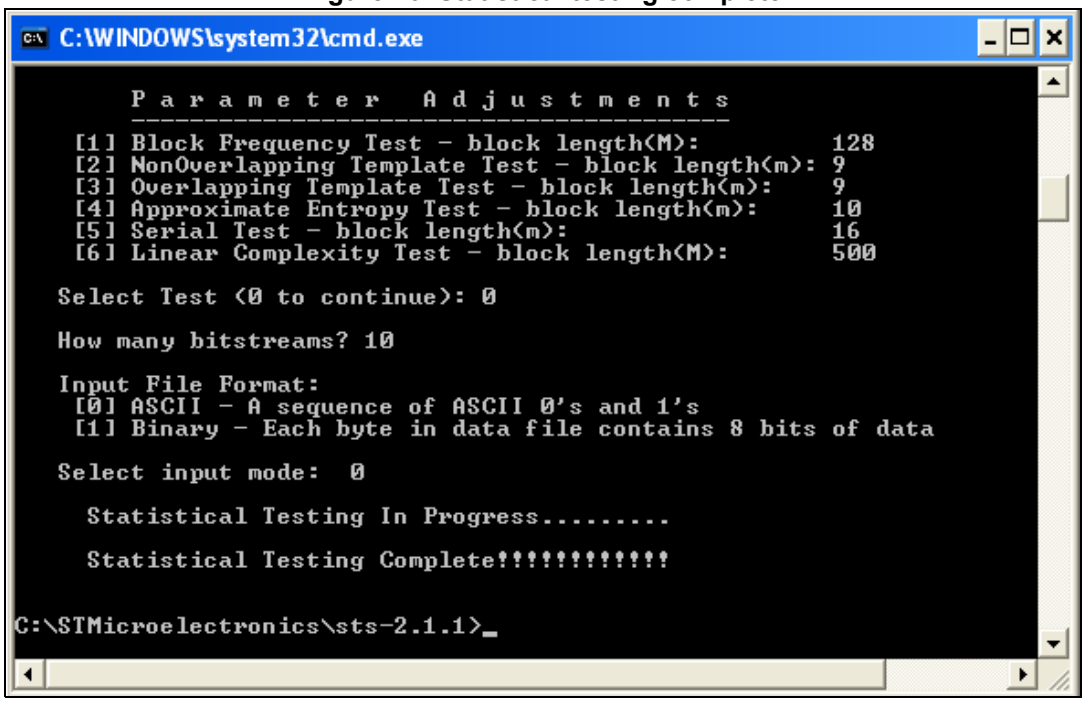
7. After entering all necessary inputs, the NIST statistical test suite starts analyzing the input file.

Figure 9. Statistical testing in progress



8. When the testing process is complete, the screen below appears.

Figure 10. Statistical testing complete



The statistical test results can be found in sts-2.1.1\experiments\AlgorithmTesting.

3.2.3 Step 3: test report

The NIST statistical tests provide an analytical routine to facilitate the interpretation of the results. A file named *finalAnalysisReport* is generated when the statistical testing is complete and saved under *sts2.1.1\experiments\AlgorithmTesting*.

The report contains a summary of experimental results of 15 tests (see *Appendix A*).

The NST statistical tests also provide a detailed report for each test, saved under *sts-2.1.1\experiments\AlgorithmTesting*\<*Test suite name*>.

The two following examples of complete NIST statistical test suite output reports are available under NIST_Test_Suite_OutputExample:

- example of an *Ascii_File_Format*, with two folders:
  - *Input_File*: contains the random number generator saved with the ascii format.
  - *Final_Analysis_Report:* contains the complete NIST statistical test suite output report based on this input file, the summary of experimental results and the report of each test.
- *example of a Binary_File_Format, with* two folders:
  - *Input_File:* contains the random number generator saved with the binary format.
  - *Final_Analysis_Report:* contains the complete NIST statistical test suite output report based on this input file, the summary of experimental results and the report of each test.

4 NIST SP800-90b test suite

4.1 Introduction

The cryptographic random bit generators (RBGs), also known as random number generators (RNGs), require a noise source that produces outputs with some level of unpredictability, expressed as min-entropy.

The specificity of the NIST SP800-90b statistical test suite is to probe the quality of random generators for cryptographic applications by its standardized means of estimating the quality of a source of entropy.

A comprehensive description of the suite is presented in the NIST document entitled *Recommendation for the Entropy Sources Used for Random Bit Generation*.

4.2 NIST SP800-90b test suite description

The NIST SP800-90b statistical test suite can be downloaded from the GitHub web site (https://github.com/usnistgov/SP800-90B_EntropyAssessment).

The SP800-90B_EntropyAssessment C++ package implements the min-entropy assessment methods included in Special Publication 800-90B.

The project is composed of two separate sections:

- IID tests, confirming that a dataset is IID (independent and identically distributed)
- non-IID tests, providing an estimate for min-entropy for any data provided

The STM32 certifiable TRNG noise source is tested using non-IID tests.

4.2.1 Non-IID track: entropy estimation for non-IID data

Not all the noise sources are able to produce IID outputs. Sequences with dependent values result in overestimates of entropy. But the diverse number of estimates reduces the probability of the overestimation of the source's entropy.

For non-IID data, various estimators (detailed below) must be calculated on the outputs of the noise source and outputs of any conditioning component, except for:

- three keyed algorithms that have been vetted for a keyed conditioning component: HMAC, CMAC and CBC-MAC
- three unkeyed functions that have been vetted for an unkeyed conditioning component: all the approved hash functions specified in FIPS 180 or FIPS 202, Hash_df and Block_Cipher_df

Note: STM32 certifiable TRNGs use the approved CMAC conditioning component.

The minimum of all the estimates is taken as the entropy assessment of the entropy source for this recommendation.

The estimators are the following:

- **Most common value estimate**: A confidence interval is constructed for the proportion p of the most common value in the input dataset and the min-entropy per sample is estimated from the upper bound of this confidence interval.
- **Collision estimate**: measures the mean number of samples to the first repeated value in a dataset. Based on the collision times, this method estimates the probability of the most-likely output value.
- **Markov estimate**: The Markov model can be used as a template for testing sources with dependencies, as the sample value always depends on the value of the previous ones. After measuring the dependencies between consecutive values from the input dataset, this model provides a min-entropy estimate that is based on the entropy present in any subsequence of outputs, instead of an estimate of the min-entropy per output.
- **Compression estimate**: Based on the amount of compression of the dataset, the entropy rate is computed by the compression estimate. The estimate is computed by generating a dictionary of values, and then computing the average number of samples required to produce an output, based on the dictionary. An entropy rate is still obtained even if the compression rate has been effected when using this statistic for testing sequences with dependencies.
- **T-tuple estimate**: The frequency of the t-tuples (such as pairs or triples) that appear in the input dataset is examined and an estimate of the entropy per sample is produced based on this frequency.
- **Longest repeated substring (LRS) estimate**: Based on the number of repeated substrings (tuples) within the input dataset, this method estimates the collision entropy of the source. This is a complementary estimate, as it handles tuple sizes that are too large for the t-tuple estimate
- **Multi-most common in window (multiMCW) prediction estimate**: Based on the last n outputs, every subpredictor of the multi-most common in window predictor aims to guess the next output. The multiMCW predictor records the number of times that each subpredictor predicted correctly the value that occurs most often in the window of the n previous outputs. And to predict the next value we use the subpredictor with the most correct predictions.
- **Lag prediction estimate**: Based on a specified lag, every subpredictor of the lag predictor predicts the next output. A scoreboard is kept by the lag predictor to record the number of times that each subpredictor was correct and the one with the most correct predictions are used for the prediction of the next value.
- **Multi-Markov model with counting (MultiMMC) prediction estimate**: composed of multiple MMC subpredictors. The frequencies for transitions from one output to a subsequent output are recorded by each MMC predictor and a prediction is made based on the most frequently observed transition from the current output. Each 1 to n depth of the n MMC subpredictors run in parallel and the one with the most correct number of predictions is used to predict the next value.
- **LZ78Y prediction estimate:** the strings that have been added to the dictionary so far are kept in the predictor dictionary. This dictionary keeps adding new strings until it reaches its maximum capacity. Every substring in the most recent n samples updates or gets added to the dictionary each time a sample is processed.

5 NIST SP800-90b test suite running and analyzing

5.1 Firmware description

To run the NIST statistical test suite as described in the previous section, two firmware are needed, one on the STM32 microcontroller side and one on the NIST SP800-90b test suite side.

5.1.1 STM32 MCU side

The firmware package is provided upon request. For more details, contact the local ST sales representative.

This program allows random numbers generation, using the STM32 RNG peripheral. It also retrieves these numbers on a workstation for testing with the NIST statistical test suite.

Each firmware program is used to generate two 64-Kbyte blocks of random numbers. The output file contains 1,024,000 random bits to be tested with the NIST statistical test.

Note: The USART configuration can be changed via the `SendToWorkstation()` function in the main.c file.

The output values can be changed by modifying the Private define in the main.c file as follows:

```
#define NUMBER_OF_RANDOM_BITS_TO_GENERATE   512000
#define BLOCK_NUMBER 2
```

5.1.2 NIST SP800-90b test suite side

Downloaded on a workstation, the NIST statistical test suite package verifies the randomness of the output file of the STM32 RNG peripheral.

The generator file to be analyzed must be stored under the *bin* folder (...\bin\data).

For more details about how the NIST statistical tests work, refer to the *readme* file available on the GitHub web site (https://github.com/usnistgov/SP800-90B_EntropyAssessment).

5.2 NIST SP800-90B test suite steps

5.2.1 Step 1: random number generator

Connect the STM32 board to the workstation, via a USB Type-A to Micro-B cable

The STM32 RNG is run via the UART firmware in order to generate a random number as described in *Section 5.1.1: STM32 MCU side*. Data are stored on the workstation using a terminal emulation application such as an PuTTY (free and open-source terminal emulator, serial console and network file transfer application).

5.2.2 Step 2: NIST statistical tests

The Makefile is used to compile the program as described in the *readme* file mentioned in *Section 5.1.2*).

For non-IID tests, the user must follow the steps detailed below:

1. Use the Makefile to compile the program: `make non_iid`
2. Run the program with

```
./ea_non_iid [-i|-c] [-a|-t] [-v] [-l <index>,<samples> ] <file_name>
[bits_per_symbol]
```

where

- **`-i`** indicates that data is unconditioned and returns an initial entropy estimate (default).
- **`-c`** indicates that data is conditioned.
- **`-a`** estimates the entropy for all data in the binary file (default).
- **`-t`** truncates the created bitstring representation of data to the first 1 Mbits.
- **`-l`** reads (at most) data samples after indexing into the file by * bytes.
- **`-v`** : verbosity flag for more output (optional, can be used multiple times)
- **`bits_per_symbol`**: number of bits per symbol. Each symbol is expected to fit within a single byte.

Example: `./ea_non_iid ../bin/l5.bin 1 -i -t -v`

5.2.3 Step 3: test report

The NIST statistical tests provide an analytical routine to facilitate the interpretation of the results:

- For the non IID tests, the result for each IID test is provided and, at the end, the final min entropy.

6 Conclusion

This application note describes the main guidelines and steps to verify the randomness of numbers generated by the STM32 microcontrollers RNG peripheral, using either NIST statistical test suite SP800-22rev1a, April 2010 or SP800-90B, January 2018.

Appendix A NIST SP800-22b statistical test suite

The results are represented as a table with p rows and q columns, where:

- p, the number of rows, corresponds to the number of statistical tests applied
- q, the number of columns (q = 13) is distributed as follows:
  - columns 1-10 correspond to the frequency of 10 Pvalues
  - column 11 is the Pvalue that arises via the application of a chi-square test11
  - column 12 is the proportion of binary sequences that passed
  - column 13 is the corresponding statistical test

The example below shows the first and last part of the test results. For more details, refer to the *finalAnalysisReport* file under *sts-2.1.1\experiments\AlgorithmTesting*.

Part 1

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

---

generator is <data/ascii.bin>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
0	1	2	1	2	1	1	1	0	1	0.911413	10/10	Frequency
1	1	0	1	3	0	2	1	1	0	0.534146	10/10	BlockFrequency
0	1	3	3	0	1	0	2	0	0	0.122325	10/10	CumulativeSums
1	1	3	1	0	1	1	1	0	1	0.739918	10/10	CumulativeSums
2	0	2	2	1	1	0	1	0	1	0.739918	10/10	Runs
1	0	1	1	0	3	1	1	0	2	0.534146	9/10	LongestRun
1	2	1	0	2	1	1	0	0	2	0.739918	10/10	Rank
3	0	1	2	1	1	0	1	0	1	0.534146	9/10	FFT
1	1	1	0	0	2	1	2	0	2	0.739918	10/10	NonOverlappingTemplate
1	1	0	0	1	1	1	3	0	2	0.534146	10/10	NonOverlappingTemplate
0	2	1	0	4	0	2	0	0	1	0.066882	10/10	NonOverlappingTemplate
0	0	0	1	1	3	0	2	1	2	0.350485	10/10	NonOverlappingTemplate
0	1	2	2	1	1	1	2	0	0	0.739918	10/10	NonOverlappingTemplate
2	2	1	0	2	0	1	1	1	0	0.739918	10/10	NonOverlappingTemplate
1	0	2	2	1	1	1	0	1	1	0.911413	10/10	NonOverlappingTemplate
0	0	1	1	0	0	2	3	1	2	0.350485	10/10	NonOverlappingTemplate

Part 2

2 0 1 0 1 2 1 0 2 1 0.739918 10/10 OverlappingTemplate
1 0 2 1 0 2 2 1 1 0 0.739918 10/10 Universal
1 1 0 0 2 0 2 3 1 0 0.350485 10/10 ApproximateEntropy
0 1 1 1 1 0 0 0 1 0 ---- 5/5 RandomExcursions
1 1 0 0 2 0 0 0 0 1 ---- 5/5 RandomExcursions
0 1 1 1 0 0 0 0 1 1 ---- 5/5 RandomExcursions
0 0 0 0 0 1 1 0 2 1 ---- 5/5 RandomExcursions
1 0 0 0 3 0 0 0 1 0 ---- 5/5 RandomExcursions
0 0 0 1 1 0 0 1 1 1 ---- 5/5 RandomExcursions
1 0 1 1 0 2 0 0 0 0 ---- 5/5 RandomExcursions
1 0 0 0 1 1 1 0 1 0 ---- 5/5 RandomExcursions
2 1 0 1 1 0 0 0 0 0 ---- 5/5 RandomExcursionsVariant
2 1 0 0 1 1 0 0 0 0 ---- 5/5 RandomExcursionsVariant
1 1 0 2 1 0 0 0 0 0 ---- 5/5 RandomExcursionsVariant
1 2 0 1 1 0 0 0 0 0 ---- 5/5 RandomExcursionsVariant
1 1 1 1 0 0 0 1 0 0 ---- 5/5 RandomExcursionsVariant
1 1 0 1 1 0 0 0 0 1 ---- 5/5 RandomExcursionsVariant
0 1 0 2 1 0 0 0 0 1 ---- 5/5 RandomExcursionsVariant
0 0 0 1 0 1 0 3 0 0 ---- 5/5 RandomExcursionsVariant
0 0 0 0 0 0 2 1 1 1 ---- 5/5 RandomExcursionsVariant
0 0 1 0 0 0 1 1 1 1 ---- 5/5 RandomExcursionsVariant
0 0 0 1 0 0 2 0 2 0 ---- 5/5 RandomExcursionsVariant
0 1 0 0 1 1 1 1 0 0 ---- 5/5 RandomExcursionsVariant
1 0 0 2 0 1 1 0 0 0 ---- 5/5 RandomExcursionsVariant
1 0 0 0 2 1 0 0 0 1 ---- 5/5 RandomExcursionsVariant
0 0 0 1 1 0 1 1 1 0 ---- 5/5 RandomExcursionsVariant
0 0 0 0 2 0 2 0 0 1 ---- 5/5 RandomExcursionsVariant
0 0 1 0 1 2 1 0 0 0 ---- 5/5 RandomExcursionsVariant
0 0 1 0 0 2 2 0 0 0 ---- 5/5 RandomExcursionsVariant
1 1 0 0 0 2 3 0 2 1 0.350485 10/10 Serial
0 2 1 0 3 1 0 1 1 1 0.534146 10/10 Serial
2 1 1 1 0 1 1 3 0 0 0.534146 10/10 LinearComplexity

- -

The minimum pass rate for each statistical test, with the exception of the random excursion (variant) test, is approximately 8 for a sample size of 10 binary sequences.

The minimum pass rate for the random excursion (variant) test is approximately 4 for a sample size of 5 binary sequences.

For further guidelines, construct a probability table using the MAPLE program provided in the addendum section of the documentation.

Appendix B NIST SP800-90b statistical test suite

This is an example of the execution of the non-IID tests.

```
$ ./ea_non_iid ../bin/l5.bin 1 -i -t -v
Opening file: '../bin/l5.bin'
Number of Binary Symbols: 1024000

Symbol alphabet consists of 2 unique symbols

Running non-IID tests...

Running Most Common Value Estimate...
MCV Estimate: mode = 530185, p-hat = 0.51775878906249995, p_u =
0.51903071907139675
   Most Common Value Estimate (bit string) = 0.946108 / 1 bit(s)

Running Entropic Statistic Estimates (bit strings only)...
Collision Estimate: X-bar = 2.4954732991667945, sigma-hat =
0.49998011778222412, p = 0.55717151750062044
   Collision Test Estimate (bit string) = 0.843807 / 1 bit(s)
Markov Estimate: P_0 = 0.4822412109375, P_1 = 0.51775878906249995, P_0,0 =
0.48618305677846313, P_0,1 = 0.51381694322153693, P_1,0 =
0.47857068759018079, P_1,1 = 0.52142931240981927, p_max =
6.2798397734367098e-37
   Markov Test Estimate (bit string) = 0.939536 / 1 bit(s)
Compression Estimate: X-bar = 5.2113069751685934, sigma-hat =
1.0187463366852749, p = 0.035431813237513987
   Compression Test Estimate (bit string) = 0.803135 / 1 bit(s)

Running Tuple Estimates...
t-Tuple Estimate: t = 16, p-hat_max = 0.53187518804173173, p_u =
0.53314533218239224
LRS Estimate: u = 17, v = 38, P_{max,W} = 0.50423097749594759, p_u =
0.50550366496585808
   T-Tuple Test Estimate (bit string) = 0.907399 / 1 bit(s)
   LRS Test Estimate (bit string) = 0.984207 / 1 bit(s)

Running Predictor Estimates...
MultiMCW Prediction Estimate: N = 1023937, Pglobal' = 0.51634782805743162
(C = 527405) Plocal = 0.42674646958653317 (r = 21)
   Multi Most Common in Window (MultiMCW) Prediction Test Estimate (bit
string) = 0.953585 / 1 bit(s)
Lag Prediction Estimate: N = 1023999, Pglobal' = 0.50526439621655594 (C =
516087) Plocal = 0.40830971576974662 (r = 20)
   Lag Prediction Test Estimate (bit string) = 0.984890 / 1 bit(s)
MultiMMC Prediction Estimate: N = 1023998, Pglobal' = 0.51899462537798435
(C = 530147) Plocal = 0.42674521449187308 (r = 21)
```

```
   Multi Markov Model with Counting (MultiMMC) Prediction Test Estimate (bit
string) = 0.946208 / 1 bit(s)
LZ78Y Prediction Estimate: N = 1023983, Pglobal' = 0.51899440603936897 (C =
530139) Plocal = 0.42674552311446512 (r = 21)
   LZ78Y Prediction Test Estimate (bit string) = 0.946209 / 1 bit(s)

H_original: 1.000000
H_bitstring: 0.803135

min(H_original, 1 X H_bitstring): 0.803135
```

Revision history

Table 3. Document revision history

Date	Revision	Changes
13-May-2013	1	Initial release.
22-Jun-2016	2	Updated: – *Introduction*. – *Section 1: STM32 microcontrollers random number generator*. – *Figure 1: Block diagram*. – *Section 3.1: Firmware description*. – *Section 3.2.1: First step: random number generator*. – *Section 3.2.2: Second step: NIST statistical test*. – *Section 4: Conclusion*. Added *Figure 2: STM32 lines embedding the RNG hardware peripheral*.
1-Oct-2019	3	Updated: – *Introduction* and *Table 1: Applicable products* – *Table 2: STM32 lines embedding the RNG hardware peripheral* – *Figure 1: STM32 true RNG block diagram* – *Section 6: Conclusion* – *Appendix A: NIST SP800-22b statistical test suite* Added: – *Section 4: NIST SP800-90b test suite* – *Section 5: NIST SP800-90b test suite running and analyzing* – *Appendix B: NIST SP800-90b statistical test suite*
10-Oct-2019	4	Updated *Table 2: STM32 lines embedding the RNG hardware peripheral*.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to *www.st.com/trademarks*. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2019 STMicroelectronics – All rights reserved