Book Stack Operation Manual

S. Doroshenko A. Fionov A. Lubkin V. Monarev
A. Pestunov B. Ryabko

1 Designation

The Book Stack is a statistical test which enables one to tell (with some degree of confidence) whether a given sequence of letters (a sample) was generated randomly or not. A randomly generated sequence (as it is assumed throughout this document) is one in which all letters are independent and appear with equal probabilities. The program implementation that comes along with this manual focuses on binary sequences, i.e. the letters are 0 and 1. So it allows to distinguish between random and non-random bit sequences, which is of particular interest for cryptology (e.g., in the development of block and stream ciphers).

2 History

A statistical test called "Book Stack" (in a more general form than the one used here) was first suggested in [1], see also its description in [2]. In a number of works of the authors, cf. [3], it was shown that this test is more powerful than all 16 tests recommended by the US National Institute of Standards and Technologies (NIST) [4]. Then the test was successfully used to detect deviations from randomness in RC-4 key-stream generator with 8-bit words [5]. We have also employed this test in distinguishing attacks against ZK-Crypt [6] and other candidates to eSTREAM — the ECRYPT Stream Cipher Project (ZK-Crypt didn't pass the test).

The first program implementations of the Book Stack test, based mainly on binary trees, were made independently by A. Pestunov, V. Monarev, S. Doroshenko, and A. Lubkin. In numerous discussions with B. Ryabko, A. Fionov and other specialists, new implementation ideas arose toward the use of hashing to speed up computations. The present program implementation based on hashing is due to Alexey Lubkin.

The last notice here concerns the discussion provoked by the name of the test. As a universal source coding method, Book Stack was first suggested by Ryabko [7]. Several years later the method was re-discovered in [8] where it was called as a "Move-To-Front" (MTF) scheme. The latter name was accepted by many researchers since they were unfamiliar with the Russian source (though published in English). In view of this state of affairs the authors of the test feel it right for themselves to use the original name given by one of them.

3 Theory

3.1 Hypothesis testing

To distinguish between random and non-random sequences the null hypothesis H_0 that the sequence is random must be tested against the alternative hypothesis H_1 that the sequence is not random. A statistical test must decide on whether to accept or reject the null hypothesis. More specifically in our test, the null hypothesis H_0 is that the sequence was generated by the source whose outputs are independent and equiprobable (in the binary case, all zeroes and ones are independent of each other and appear with probability 1/2). The alternative hypothesis $H_1 = \neg H_0$ is that the sequence was generated by a stationary and ergodic source different from the source under H_0. A generated sequence subject to statistical testing is usually called a sample. Often the whole generated sequence is divided into a number of independent samples for testing.

Hypothesis testing is probabilistic in its nature. This is just because if H_0 is true then *any* sample of a given length is equally likely. So when we look at a specific sample we cannot say for sure whether it is random or not. A so called Type I error occurs if H_0 is true but, nevertheless, is rejected by the test. The probability of Type I error is often called the level of significance of the test and denoted by α. The values of α stretching form 0.001 to 0.05 are employed in practical cryptography. The opposite situation, when H_1 is true but the test accepts H_0, is a Type II error. The probability of Type II error, denoted by β, is usually difficult to determine. For example, the sequence generated by a pseudo-random bit generator (PRGB) is definitely non-random since any PRGB is a deterministic algorithm. Yet, for a good (cryptographically secure) PRBG any applicable test should accept H_0. We can say that within some model of non-randomness, less values of β

correspond to more powerful tests.

3.2 Statistical criterion used

To derive a decision upon the null hypothesis a statistic on a sample is first computed. In our test, we use a well-known x^2 statistic which is described as follows. Let n denote the sample size, n_0 the number of zeroes and n_1 the number of ones in the sample, $n_0 + n_1 = n$. Let p and q denote *a priori* probabilities of zero and one, respectively. Then pn and qn are the expected numbers of zeroes and ones in a sample of size n. The x^2 statistic is defined by the equation

$$x^2 = \frac{(n_0 - pn)^2}{pn} + \frac{(n_1 - qn)^2}{qn}. \tag{1}$$

For testing H_0 directly, we have $p = q = 1/2$ and (1) is reduced to

$$x^2 = \frac{(n_0 - n_1)^2}{n}.$$

However, direct testing is usually inefficient and, as a rule, some processing of the sample is performed after which an equivalent to H_0 hypothesis (denoted H_0^*) is tested for very skew distribution.

The scheme of statistical test is the following. We set some critical (threshold) value $t_\alpha > 0$. Then we compute the x^2 statistic on a sample and compare it to the critical value. The null hypothesis H_0 (or H_0^*) is accepted if $x^2 < t_\alpha$. Otherwise H_0 is rejected. So the probability of Type I error (the level of significance) is the probability of the event $x^2 \geq t_\alpha$ when H_0 is true, i.e. $\alpha = P(x^2 \geq t_\alpha)$.

It is known that the x^2 statistic obeys asymptotically the χ^2 (chi-square) distribution (with one degree of freedom in our case). It is generally accepted that it is quite correct to employ the χ^2 distribution for x^2 if both qn and pn are greater than 5. There are percentile tables for χ^2 distribution suggested in the literature that show the values of t_α for various α. For our experiments we used:

$$\begin{array}{ll} \alpha = 0.05 & t_\alpha = 3.8415, \\ \alpha = 0.001 & t_\alpha = 10.8376. \end{array}$$

For example, if in a series of tests (on many samples of the generated sequence), in 95% of cases, we observed the values of statistic greater than 3.8415, we may conclude that the sequence is not random (i.e. reject H_0).

3.3 The Book Stack test

In the Book Stack test, as it is implemented in the supplied computer program, the input sample $x_1, x_2, \dots, x_s$ of size s, where each $x_i \in \{0, 1\}$, is considered to consist of w-bit words, $1 \le w \le 32$, extracted from the sample one after another with an optional omission of several bits (a "blank" between words). The words do not overlap. For example, if $w = 3$ and blank is 1, the bit sample 0111010100010100... converts to the word sample 011 010 000 010 ... or, in decimal notation, 3 2 0 2 ....

So we have now a sequence of words $y_1, y_2, \dots, y_n$ obtained from the input sample, where all $y_i \in \{0, 1, \dots, 2^w - 1\}$.

In the Book Stack test, all the words are ordered from 0 to $2^w - 1$. We denote the ordinal number of a word a by $\nu(a)$. The order is changed after observing each word y_i according to the rule

$$\nu^{i+1}(a) = \begin{cases} 0\,, & \text{if } y_i = a, \\ \nu^i(a) + 1\,, & \text{if } \nu^i(a) < \nu^i(y_i), \\ \nu^i(a)\,, & \text{if } \nu^i(a) > \nu^i(y_i) \end{cases} \tag{2}$$

where ν^i is the order after observing $y_1, y_2, \dots, y_i$, $i = 1, \dots, n$, the initial order ν^1 being defined arbitrarily. (For example, we can set $\nu^1 = (0, 1, \dots, 2^w - 1)$.)

Let us explain informally (2). Suppose that the words are arranged in a stack, like a stack of books, and $\nu^1(a)$ is a position of word a in the stack. Let the first word y_1 of the sample $y_1, y_2, \dots, y_n$ be a. If it occupies the k-th position in the stack ($\nu^1(a) = k$), then extract a out of the stack and push it to the top. (It means that the order is changed according to (2).) Repeat the procedure with the second letter y_2 and the stack obtained, *etc*.

It can help to understand the main idea of the suggested method if we take into account that, if hypothesis H_1 is true, the frequent words (as frequently used books) will have relatively small ordinals (will spend more time near the top of the stack). On the other hand, if H_0 is true, the probability to find each word at each position is equal to $1/2^w$.

Let's continue the description. The set of all indexes $\{0, \dots, 2^w - 1\}$ is divided into two subsets $A_0 = \{0, 1, \dots, u - 1\}$ and $A_1 = \{u, \dots, 2^w - 1\}$. The subset A_0 is said to be the upper part of the book stack and its cardinality $|A_0| = u$ is specified as a parameter of the test. Then, observing $y_1, y_2, \dots, y_n$, we calculate how many $\nu^i(y_i)$, $i = 1, \dots, n$, belong to subsets A_0 and A_1. We denote these numbers by n_0 and n_1, respectively. More formally,

$$n_k = |\{i : \nu^i(y_i) \in A_k, i = 1, \dots, n\}|, \quad k = 0, 1\,.$$

Obviously, if the null hypothesis H_0 is true then all the words have the same probability $1/2^w$ and the probability of the event $\nu^i(y_i) \in A_k$ is equal to $|A_k|/2^w$. Using the notation of Sect. 3.2 and $|A_0| = u$ we may write $p = u/2^w$, $q = 1 - p$. Now testing H_0 is replaced by testing the equivalent hypothesis H_0^* that the binary random variable Y obeys the distribution $P(Y = 0) = p$, $P(Y = 1) = q$, given the sample $y_1, y_2, \ldots, y_n$ with n_0 zeroes and n_1 ones. This can be done using the χ^2 distribution as was explained in Sect. 3.2.

We do not describe the exact rule for selecting the parameters of the test, namely, the word length w, the blank, and the size of the upper part u, but recommend to carry out some experiments for finding the parameters which make the sample size minimal (or, at least, acceptable). The point is that there are many cryptographic applications where it is possible to implement some experiments for optimizing the parameter values and, then, to test the hypothesis based on independent data. For example, in case of testing a PRBG it is possible to seek suitable parameters using a part of generated sequence and then to test the generator using a new part of the sequence.

Let us consider an example. Let

$$\begin{aligned} w &= 3\,, \quad y_1 \ldots y_8 = 3\,6\,3\,3\,6\,1\,6\,1\,, \\ u &= 3\,, \quad \nu^1 = (0,1,2,3,4,5,6,7)\,. \end{aligned}$$

Then

$$\begin{array}{lll}
\nu^1 = (0,1,2,3,4,5,6,7), & n_0 = 0\,, & n_1 = 0\,; \\
\nu^2 = (3,0,1,2,4,5,6,7), & n_0 = 1\,; & \\
\nu^3 = (6,3,0,1,2,4,5,7), & & n_1 = 1\,; \\
\nu^4 = (3,6,0,1,2,4,5,7), & n_0 = 2\,; & \\
\nu^5 = (3,6,0,1,2,4,5,7), & n_0 = 3\,; & \\
\nu^6 = (6,3,0,1,2,4,5,7), & n_0 = 4\,; & \\
\nu^7 = (1,6,3,0,2,4,5,7), & & n_1 = 2\,; \\
\nu^8 = (6,1,3,0,2,4,5,7), & n_0 = 5\,; & \\
\nu^9 = (1,6,3,0,2,4,5,7), & n_0 = 6\,. &
\end{array}$$

We can see that the words 3 and 6 are quite frequent and the book stack test indicates this non-uniformity quite well. Indeed, the average values of n_0 and n_1 are equal to 3 and 5, whereas the real values are 6 and 2, respectively.

Let us make a remark on complexity of the test. The "naive" method of transformation according to (2) would take the number of operations proportional to 2^w. But the simple observation is that only the upper part of the stack has to stored, which effectively reduces complexity to $O(u)$ operations. More complicated algorithms based on AVL or other balanced trees can perform all operations in (2) in $O(\log u)$ time. In the supplied program implementation, we use an even faster algorithm based on hashing whose expected running time is $O(1)$.

4 User's Guide

The Book Stack test implemented in C++ language is supplied as `bookstack.zip`. The contents of the archive is the following:

Directory	File	Description
`source`	`bs.cpp`	the Book Stack test
`source`	`rc4.cpp`	RC4 key-stream generator
`source`	`zk.cpp`	ZK-Crypt key-stream generator
`win32`	`*.exe`	executable files for Win32 computing environment
`win32`	`*.cmd`	scripts to run sample tests for RC4 and ZK-Crypt generators by double-clicking with the mouse

The executables for Win32 are supplied since, as a matter of fact, not all the users of Windows operating systems have C++ compilers available. The programs can be readily run on UNIX-type systems, such as Linux, FreeBSD, and other (we hope), in which case we assume that GCC C++ compiler should be used. To compile the programs use the following command lines (exemplified by compiling `bs.cpp`):

`bcc32 -O2 bs.cpp`	for Borland C
`cl /O2 /EHsc bs.cpp`	for MS Visual C
`icl /O3 bs.cpp`	for Intel C
`cc -O2 -o bs bs.cpp -l stdc++`	for GCC

The executable file `bs.exe` or `bs` (in case of GCC) must then be placed somewhere to be able to run from. For example, it may be the current working directory in Windows or `/bin` in UNIX-type systems. Make all similarly with the other programs.

The command line to run the Book Stack test has the following format:

```
bs [-f filename] [-n num] [-w num] [-b num] [-u num] [-q]
```

The brackets should not be typed, they are only used to show that the parameter within is optional and may be omitted. The parameter meanings are explained below.

`-f filename` Specifies the name of a file containing the sample to test. The sample is considered as a stream of bits without any internal structure. The sample size is determined by the size of file (unless

-n is specified). If the parameter is omitted, stdin is assumed. This allows to bind the generator's output with the test's input via a pipe.

-n num Sets the maximum number of bits to read from a file or stdin. If not specified, the sample is read till the end of file.

-w num Specifies the word length w for the test. The values from 1 to 32 are supported. The default is $w = 32$.

-b num Specifies the blank between words. The values from 0 to 32 are currently supported. The default value is 0, i.e. no blanks. If the parameter is given *after* -w then the first w bits of the input stream are used to form a word, then the blank is applied (i.e. the specified number of bits are discarded), then the next word is formed an so on. If the parameter -b num is given *before* -w then the blank is applied before each word formation.

-u num Specifies the size of the upper part of book stack. The default value is $u = 2^{w/2}$. It makes no sense to set $u \geq 2^w$.

-q Suppresses the explanatory information for the test results. The test writes to the standard output only the value of statistic computed. This mode is useful when embedding the Book Stack test into user-defined shells for performing, e.g., a series of tests and computing the net result.

Together with the test we also supply our implementation of RC4 (with 8-bit words) and ZK-Crypt generators in files rc4.cpp and zk.cpp, respectively. These may be compiled similarly as bs.cpp. The programs write generated sequences to stdout. Their parameters are as follows.

```
rc4 | zk [-k key] -n num [-q]
```

-k key The secret key (seed) for generator. A 128-bit key is specified as a hexadecimal number (with optional 0x prefix). If the parameter -k is omitted then an internally generated random (not cryptographically secure) key is used. The key actually used is written (in hexadecimal) to stderr.

-n num Specifies the number of bits to produce. This parameter is mandatory.

-q Suppresses the output of the key.

Let us show some examples of usage:

```
rc4 -k 0x5123b5678d01234f678ec123b56a8972 -n 1000 > rc4.bin
```

```
zk -n 10000000 -q > zk.bin
```

```
bs -f rc4.bin -w 16 -u 5
```

```
bs -f zk.bin -w 16 -b 16 -u 20000 -q
```

And a more efficient connection between programs via a pipe:

```
rc4 -n 10000000 | bs -w 16 -u 5
```

The following parameters were used in experiments described in [5, 6]:

```
rc4 -n 4294967296 | bs -w 16 -u 16
zk -n 16777216 | bs -w 32 -u 65536
```

Note that program bs does not attempt to interpret the result of a test, it simply outputs the value of x^2 statistic computed. The interpretation is left to the user since it depends on a number of things outside of the program (the desired level of significance, the number of tests carried out, *etc*). The user may refer to the following table that shows selected percentiles of the χ^2 distribution with 1 degree of freedom.

α	0.99	0.95	0.75	0.50	0.25	0.10
x^2	0.00016	0.00393	0.1015	0.4549	1.323	2.7055
α	0.05	0.025	0.010	0.005	0.001	
x^2	3.8415	5.0239	6.6349	7.8794	10.8276	

The entries in the table have the following meaning: if X is a random variable that obeys the χ^2 distribution with one degree of freedom, then $P(X > x^2) = \alpha$.

For example, suppose that, on a given sample, bs outputs $x^2 = 7$. If the sample were random, this value of x^2 would occur with probability less than 0.01. So we may suspect the sample to be non-random. Similarly, if bs outputs $x^2 = 0.00015$, we may also suspect the sample as non-random since in 99% of cases truly random samples give the values of x^2 greater than 0.00016.

But usually we need to carry out many experiments on independent data in order to judge more definitely on the source of these data (the generator). For example, if in 100 experiments bs outputs $x^2 \geq 6.6349$, we may conclude, with the level of significance 0.01, that the generator is not random.

References

[1] Ryabko, B. and Pestunov, A. (2004). "Book stack" as a new statistical test for random numbers, *Probl. Inform. Transmission*, **40**, 1, pp. 66–71.

[2] Ryabko, B. and Fionov, A. (2006). *Basics of Contemporary Cryptography for IT Practitioners*, World Scientific Publishing Co.

[3] Ryabko, B. Ya., Fionov, A. N., Monarev, V. A. and Shokin, Yu. I. (2005). Using information theory approach to randomness testing, *Computational Science and High Performance Computing II*, Springer, pp. 261–272.

[4] Rukhin, A. *et al.* (2001). *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, NIST Special Publication 800-22 (rev. May 15, 2001).

[5] Doroshenko, S. and Ryabko, B. (2006). The experimental distinguishing attack on RC4, *Cryptology ePrint Archive*, Report 2006/070 (http://eprint.iacr.org/).

[6] Lubkin, A. and Ryabko, B. (2005). The distinguishing attack on ZK-Crypt cipher, *eSTREAM, ECRYPT Stream Cipher Project*, Report 2005/076 (http://www.ecrypt.eu.org/stream).

[7] Ryabko, B. Ya. (1980). Information compression by a book stack, *Probl. Inform. Transmission*, **16**, 4, pp. 16–21.

[8] Bently, J. L., Sleator, D. D., Tarjan, R. E. and Wei, V. K. (1986). A locally adaptive data compression scheme, *Comm. ACM*, **29**, pp. 320–330.