htw saar
Studiengang Kommunikationsinformatik (Master)
Studiengang Praktische Informatik (Master)
Prof. Dr.–Ing. Damian Weber

Security and Cryptography

Random Numbers

1 Motivation

A cryptographic key is a valuable secret.

It protects messages, sometimes other keys, either changes frequently such as in session keys per web browser session or almost never – a public key certificate validity duration may easily span years.

Secrets should be unguessable, i.e. not predictable.

If it's computable, it's predictable.

So if it's not predictable, it cannot be computable.

Since real world computers are inherently deterministic, an ideal random source must be located externally. For theoretical computer scientists, the model is a probabilistic Turing machine, which is defined with a coin tossing unit to determine which state the next one given a character of input on the tape. For practical computer scientists, some external devices are in use, see turbid for an example which uses audio I/O

`http://www.av8n.com/turbid/`

BTW not all complexity questions are solved for the theoretical model, for example it is not known if coin tossing allows for problems to be solved in polynomial time i.e. $P = BPP$, here the term BPP stands for bounded-error probabilistic polynomial time. Small evidence that BPP does not give more capabilities is the proof that prime detection is in P, a problem that until 2002 only a BPP-solution.

BTW not all uses of random numbers are as valuable as a secret key, but also the following uses may leak some information if the random number can be guessed:

initialization vectors for symmetric cipher modes, random authentication challenges, nonces which are numbers used only once in protocols.

By all practical means, an external random source can not be assumed for every computing device, the accompanying cost and convenience cannot be justified.

So cryptography usually resorts to the construction

a) generate a small random secret externally, the *seed*

b) produce algorithmically a long sequence of *pseudo–random* bits

   i) predictably by the creator

   ii) computationally unpredictable by an attacker

For the remainder of this note, we are therefore concerned with seeded pseudo-random-generators (PRNGs) which are producing a sequence of seemingly random bits.

As an aside, provable security w.r.t. factoring being hard and/or discrete logs are hard, one can take 1 bit from the Blum-Blum-Shub generator and some bits out of the Gennaro generator provided that the moduli used have a bitsize ≥ 5000.

`http://eprint.iacr.org/2006/229.pdf`

2 Unpredictability

Unpredictability means that nobody (but the creator) can output the same random bits with a probability better than 1/2. This must even hold true even after observing an arbitrarily long sequence of pseudo-random bits by that generator. Implicitely this means, an attacker cannot predict the next n bits correctly with probability better than $1/2^n$.

Implicitly this definition covers all statistical attacks. If, for example the PRNG outputs 1 with probability 2/3 and 0 with probability 1/3 then we can guess the next bit with probability $2/3 > 1/2$. Equally, if the subsequence 111 occurs more often than in 1/8 of all cases, this violates the $1/2^3$ requirement for the next 3 bits.

Not surprisingly, the first attempt of solving the security problem with random sequences is to start with a PRNG having a lot of good statistical properties. One of these is the linear congruential generator.

3 Linear Congruential Generator (LCG)

Fix a modulus $m \in \mathbf{N}$, choose $a, b < m$, a seed x_0 with $0 \leq x_0 < m$ and define the sequence

$$x_i = a \cdot x_{i-1} + b \mod m, \qquad i \in \mathbf{N}.$$

This sequence has been invented and first studied by D.H. Lehmer.

It has a number of provable statistical properties which make it usable in many non–security–critical environments for example in computer games. There are use cases where other statistical defects of the LCG prohibit its use (serial correlation makes LCG unsuitable for Monte–Carlo–Simulation). LCGs typically fail on a statistical test called *spectral test.*

When a programmer uses `rand()` or `lrand()` with a C compiler, in many systems the LCG is used.

The length of the period is m if and only if

a) $\gcd(b, m) = 1$

b) $a - 1$ divisible by all prime factors of m

c) if $4|m$ then $4|a - 1$

which is the statement of the Hull-Dobell Theorem.

More details to be found in

`http://en.wikipedia.org/wiki/Linear_congruential_generator`

As for the unpredictability the LCG is simply not useful.

For instance let a, b, x_0 be secret, you can compute a, b from three successive x_i values, say x_{i-1}, x_i, x_{i+1}.

$$\begin{array}{rcccccc} x_i & = & a & \cdot & x_{i-1} & + & b \quad \mod m \\ x_{i+1} & = & a & \cdot & x_i & + & b \quad \mod m \end{array}$$

Two linear equations with two unknowns a and b are readily solved and output x_{i+1} may be predicted from that.

4 Testing

4.1 American and German Standards

A PRNG can be scrutinized by a series of statistical tests.

What NIST thinks of the necessary statistical tests which a cryptographic generator has to pass is described in its SP800-22 (special publication) document

`http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf`

These are reported here in chronological order.

For comments on the NIST tests and a comparison with other standards, see the talk by Th. Risse on the 11. Workshop Mathematik in Ingenieur-wissenschaftlichen Studiengängen, Hochschule Bochum, 30.9.2013

`http://www.weblearn.hs-bremen.de/risse/papers/MathEng11/RNGs.pdf`

For BSI's point of view regarding generation of random numbers, see

```
https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/
Interpretationen/AIS_20_AIS_31_Evaluation_of_random_number_generators_e.pdf
?__blob=publicationFile
```

4.2 General Concept

Statistical testing starts with assuming a (null) hypothesis H_0 which the test should affirm or falsify. For the purpose of a pseudo-random sequence, our null hypothesis is

> H_0: *the sequence being tested is random.*

The opposite may be true which can be stated as the alternative hypothesis H_a

> H_a: *the sequence being tested is not random.*

The conclusion of the test, namely accepting or rejecting a PRNG has two good and two bad cases:

a) good cases

   i) the generator produces random numbers and we accept it

   ii) the generator produces non-random numbers and we reject it

b) bad cases

   i) (type I error, false positive, false alarm)
   the generator produces random numbers and we reject it

   ii) (type II error, false negative)
   the generator produces non-random numbers and we accept it

The probability α of a false alarm (type I error) is the level of significance of the test. For crypto purposes, a common value of α is 0.01 this is the bound chosen by NIST. This means, a true random generator gets rejected by the procedures of this NIST standard with probability 1%.

The case of a false acceptance (type II error) is fatal for crypto applications, its probability is denoted by β. The sequence will look like random, because statistical tests are passed, but it is not random because of some unknown defect. The goal is to minimize β. Though β is difficult to determine, a bound can be computed as a function of α and n, the number of observed bits.

Testing assumptions about the sequence are

- uniformity: at each point of the bit sequence a 0 and a 1 are equally likely with probability 1/2
- scalability: the test can be applied to any subsequence of the sequence
- consistency: the result is independent of the seed value

4.3 Notes from Probability Theory

4.3.1 Error Function

Crucial for acceptance or rejection of a sequence is the complementary error function

$$\operatorname{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-t^2}\, dt$$

This function is available in the math library of C

```
double erfc(double x);
```

This function is new to Python 2.7

```
math.erfc(x)
```

For most tests, some 100 input bits suffice. If this is not the case, this is explicitly noted, the first one in the list with this property is the binary matrix rank test in section 5.5.

4.3.2 Gamma Function

The Gamma function as termed by Legendre is a continuation of the factorial function to the reals. With

$$\Gamma(t) = \int_0^\infty x^{t-1}e^{-x}dx$$

the equality

$$\Gamma(n) = (n-1)!$$

holds for $n \in \mathbf{N}$. It is the only continuation of the factorial to the reals if the solution shall be a superconvex function, i.e. a function $f(x)$ where $\log(f(x))$ is convex, the uniqueness proved by Bohr and Mollerup in 1922.

`http://en.wikipedia.org/wiki/Bohr%E2%80%93Mollerup_theorem`

The Gamma function is said to be *incomplete* if the upper or lower limit of integration is replaced by a variable. One then talks about the upper or the lower incomplete Gamma function if the lower or the upper bound of integration is replaced by a variable. In the statstical tests proposed by NIST, the upper incomplete Gamma function as defined by

$$\Gamma(a,x) = \int_x^\infty t^{a-1}e^{-t}dt$$

is sometimes employed when rejecting a PRNG. In order to avoid confusion, the lower incomplete Gamma function is denoted by a lowercase $\gamma(a,x)$ with integration bounds 0 and x.

$$\gamma(a,x) = \int_0^x t^{a-1}e^{-t}dt$$

See Mathworld entry

`http://mathworld.wolfram.com/IncompleteGammaFunction.html` or Wikipedia at

```
http://en.wikipedia.org/wiki/Gamma_function
```

It is implemented in the GSL (GNU Scientific Library), which is a C/C++ Library.

```
double gsl_sf_gamma_inc_Q (double a, double x)
```

The GSL is available as port `math/gsl` in FreeBSD or as source code download from a GNU mirror, for example

```
http://ftp.halifax.rwth-aachen.de/gnu/gsl/gsl-1.16.tar.gz
```

The gamma functions are also implemented in GP/PARI where `gamma` serves as $\Gamma(t)$, `incgam` as upper incomplete $\Gamma(a,x)$ `incgamc` as lower (read complementary) incomplete $\gamma(a,x)$

```
? gamma(5)
%2 = 24.00000000000000000000000000000000000
? incgam(1,0)
%3 = 1.000000000000000000000000000000000000
? incgamc(1,0)
%4 = 0.E-38
```

The distinction can be made at these special values, since

$$\Gamma(1,x) = e^{-x} \qquad \gamma(1,x) = 1 - e^{-x}.$$

4.4 χ^2 Test

A χ^2 test is useful to determine if an observed number of counts in categories is in accordance with a theoretical expectation. Categories mean that this test only applies to discrete values and not to values measured on a continuous scale (although one might group these into categories).

As an illustrative example we are told that in a tombola winning and losing tickets are equally distributed. We have two categories: *winning* and *losing*.

There are 1,000 tickets, we cannot afford to buy all these and check them. But we can do interviews with some people (which should be randomly chosen) and can count their wins and losses.

After counting 120 tombola tickets, we realize, there were 40 winning and 80 losing tickets. The question that the χ^2 test answers is: can we reject the hypothesis that the reality (true rate of winning tickets is 1/2) is in accordance with the theory?

The χ^2 test performs the following steps.

a) For each category i, compute the formula
$$d_i = \frac{(c_i - e_i)^2}{e_i}$$
where c_i is the counted, e_i is the expected value of the category

b) sum up these d–values
$$\chi^2 = \sum_i d_i$$

c) compute the P–value (probability value): how probable is it that the counted values can occur, if the theoretical expectation t_i should be fulfilled.
$$P = \Gamma(k/2, \chi^2/2).$$
Here k is the degree of freedom, i.e. how many independent categories are considered.

d) if the P–value is below a threshold α (social sciences 5%, random number generators 1%) then reject the hypothesis (the theoretical expectation must be wrong, the alleged random number generator is not random)

In our example
$$\chi^2 = \frac{(80-60)^2}{60} + \frac{(40-60)^2}{60} = 13.34.$$

We have two categories but the number in one category (how many winning tickets) determines the other (how many losing tickets), so we have $k = 1$ degree of freedom.

We therefore compute
$$P = \Gamma(1/2, \chi^2/2) = 0.000000425555$$
which is far below $\alpha = 0.01$ so we reject the hypothesis of equal distribution of wins and losses.

5 Statistical Tests by SP800–22

The i–th subsection of this section refers to section 2.i of SP800–22.

5.1 Frequency (Monobit) Test

The frequency bit test redefines the bits 0 and 1 as -1 and 1 respectively by using the function $f(b) = 2b - 1$. It proceeds by summing up all the $f(b)$ where equal occurences of 0 and 1 cancel out. Call the resulting value S and compute the test statistic

$$s = \frac{|S|}{\sqrt{n}}$$

and its P–value

$$P = \operatorname{erfc}\left(\frac{s}{\sqrt{2}}\right)$$

Reject the PRNG if $P < \alpha$.

5.2 Frequency Test within a Block

For this test to be meaningful, the monobit frequency test has to be passed.

The block test partitions the input sequence in N blocks of size M, where $N = n/M$. Superflous bits are discarded. The recommendation for n, M and N are at least $n \geq 100$ input bits, $M > 0.01 \cdot n \geq 20$ and $N < 100$.

Let π_i the percentage of 1s in block i.

Compute the χ^2 test.

$$\chi^2 = 4M \sum_{i=1}^{N} \left(\pi_i - \frac{1}{2}\right)^2$$

Compute the P–value by using the (upper) incomplete gamma function $P = \Gamma(N/2, \chi^2/2)$.

Reject the PRNG if $P < \alpha$.

5.3 Runs Test

The Runs Test concerns with the number of runs in a sequence. A run is a maximal sequence of identical bits, be it a 0-run or a 1-run. A run is bounded on the left and on the right by the opposite bit. If the length of such a sequence is k, we speak of a run of length k. The test asserts if the oscillation between 0s and 1s is too fast or too slow.

For this test to be meaningful, the monobit frequency test has to be passed.

The test counts how often the consecutive bits are not identical, i.e.

$$V_n = \sum_{i=0}^{n-2} (\epsilon_i \neq \epsilon_{i+1})$$

where $(\epsilon_0, \ldots, \epsilon_{n-1})$ is the input sequence and the condition inside $(\ldots)$ yields *false* as 0 and *true* as 1.

The P–value of this experiment is

$$P = \operatorname{erfc}\left(\frac{|V_n - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}\right)$$

where π is the proportion of 1s in the input sequence.

As usual, reject the PRNG if $P < \alpha$.

5.4 Test for the Longest Run of Ones in a Block

The test checks whether the observed number of longest runs of length k is in accordance with the expected number. The lengths are partitioned in classes v_i and the size M of the blocks is prescribed within NIST's test suite.

$$M \in \{8, 128, 10\,000\}$$

The classes are constructed as follows:

v_i	8	128	10 000
v_0	≤ 1	≤ 4	≤ 10
v_1	2	5	11
v_2	3	6	12
v_3	≥ 4	7	13
v_4		8	14
v_5		≥ 9	15
v_6			≥ 16

The distribution to be expected is the χ^2–distribution.

$$\chi^2 = \sum_{i=0}^{K} \frac{(v_i - N\pi_i)^2}{N\pi_i}$$

where K is the number of v_i classes minus one and the value of N is chosen w.r.t. M as follows: N is 16, 49, 75 when M is 8, 128, 10000 respectively.

The P–value is

$$P = \Gamma\left(\frac{3}{2}, \frac{\chi^2}{2}\right)$$

as usual, reject the sequence if $P < \alpha$.

5.5 Binary Matrix Rank Test

The purpose of this test is to detect linear dependencies among fixed length subsequences of the original sequence. The length of a vector is denoted by M as before, so the vector essentially is a block within the bitstream. Nevertheless the block in this test is defined in another way, see the next paragraph.

The NIST test is configured to check 32×32 matrices, that is, 1024 bits or 128 bytes per round. Divide the matrix into N blocks of 1024 bits. For meaningful results, it is recommended that $N \geq 38$.

For each block construct the matrix A_i consisting of 32 rows with 32 entries each. Determine the rank of of A_i and partition all observed ranks into the following classes (SP800-22 uses row rank which is equal to column rank):

a) class 1: full rank

b) class 2: full rank minus 1

c) class 3: less than full rank minus 1

If the sequence is random, the rank shows a χ^2 distribution, which is checked by computing

$$\chi^2 = \sum_{i=1}^{3} \frac{(F - c_i N)^2}{c_i N}$$

with $c_1 = 0.2888, c_2 = 0.5776, c_3 = 0.1336$ the probabilities that this rank class is appropriate.

The P–value is computed as

$$P = e^{-\chi^2/2}.$$

As usual, the PRNG gets rejected if $P < \alpha$.

5.6 Discrete Fourier Transform (Spectral) Test

The purpose of this test is to detect periodic features in an allegedly random sequence.

Like in the frequency bit test the bits 0 and 1 are redefined as -1 and 1 respectively by using the function $f(b) = 2b - 1$ on the input sequence

$$(\epsilon_0, \ldots, \epsilon_{n-1}) \qquad n \geq 1000.$$

It proceeds by performing a discrete Fourier Transform of the resulting +1/–1–vector. For $0 \leq j < n/2$ compute

$$a_j = \sum_{k=0}^{n-1} f(\epsilon_k) \cdot \exp(2\pi i \cdot k \cdot j/n)$$

where $i = \sqrt{-1}$ is the imaginary unit.

As usual this can be carried out in the reals via

$$\exp(2\pi i \cdot k \cdot j/n) = \cos(2\pi k \cdot j/n) + i \sin(2\pi k \cdot j/n)$$

Then the absolute value is computed on the first half of this vector. For $a \in \mathbf{C}$ with $a = c + di$ we have

$$|a| = \sqrt{(c^2 + d^2)}.$$

Define

$$T = \sqrt{\left(\log \frac{1}{0.05}\right) \cdot n}$$

as the 95% peak height threshold value. This means, 95% of all $|a_j|$ should be $\leq T$. Let N_0 be the expected number and N_1 the actual observed number, that is

$$N_0 = 0.95 \cdot n/2.$$

We use these to obtain a value

$$d = \frac{N_1 - N_0}{\sqrt{0.95 \cdot 0.05 \cdot n/4.}}$$

Finally, the P–value is

$$P = \operatorname{erfc}\left(\frac{|d|}{\sqrt{2}}\right).$$

Reject the PRNG, if $P < \alpha$.

5.7 Non-overlapping Template Matching Test

The input sequence is divided in blocks of size M, so that there are $N = n/M$ blocks.

A bit template B of size $m \leq M$ is chosen which slides window-like over each block.

Let W_j be the number of times that B matches a substring in block j, where $0 \leq j \leq N-1$. The window slides one bit except in case of a match when it slides over m bits.

This is an experiment obeying a χ^2 distribution, computed with help of mean and variance.

$$\mu = \frac{M-m+1}{2^m} \qquad \sigma^2 = M\left(\frac{1}{2^m} - \frac{2m-1}{2^{2m}}\right).$$

The χ^2 value and P–value are computed by

$$\chi^2 = \sum_{j=0}^{N-1} \frac{(W_j - \mu)^2}{\sigma^2}$$

and

$$P = \Gamma\left(\frac{N}{2}, \frac{\chi^2}{2}\right)$$

Reject the PRNG, if $P < \alpha$.

5.8 Overlapping Template Matching Test

The only difference w.r.t. the previous test is that the window slides by 1 bit in the matching case, too. So for example, matching ,,000" in the bit sequence 100001 gives two matches here and 1 match in the non–overlapping test.

We skip this test, because

- the exposition in the NIST document is not accurate (degree of freedom is defined but not used, computation of the π_i is not described in detail),
- it aims for similar defects than the previous test, and
- apparently not even true random number generators are able to pass these two tests.

All these points are also mentioned within the 2005 report *Random Number Generators: An Evaluation and Comparison of Random.org and Some Commonly Used Generators*

`http://www.random.org/analysis/Analysis2005.pdf`

5.9 Maurer's ,,Universal Statistical" Test

The goal of this test is to estimate the per-bit entropy of the input stream. Based on a compression type idea of Ziv several statistic measures should be checked by this method according to the author. Ueli Maurer is a renowned cryptographer for at least two decades contributing many research papers within this time span.

For $6 \leq L \leq 16$ the test checks the distance of occurences of each L–bit pattern. Since there are 2^L bit patterns, this quickly becomes quite time consuming for larger L.

The input of length $n = 10 \cdot 2^L + 1000 \cdot 2^L$ gets partitioned into into a initialization segment of Q blocks of L bit and a test segment of K blocks of L bit, where L and Q are parameter of the test.

Set

$$K = [\frac{n}{L}] + 1 - Q.$$

Superflous bits that do not form an L bit block are discarded.

Initialize a table with all L–bit patterns and record the last instance (block number b) where a pattern p occurs within the initialization segment as $T_p = b$. If the pattern p does not occur within the initialization segment then record $T_p = 0$.

Now run L–bit–blockwise through the test segment and do the following for each pattern p detected at block number b:

a) sum up logarithmic entropy value $s = s + \log_2(bT_p)$.

b) update T_p according to $T_p = b$

5.10 Linear Complexity Test

This is the LFSR test. How big is the state of a linear feedback shift register in order to produce such a sequence. Each finite sequence can be produced by an LFSR, there is a constructive proof in a Theorem of Berlekamp-Massey that gives rise to a corresponding algorithm. Possibly, the LFSR is as big as the sequence so there is no gain in building that one. Incidentally, then the sequence can be considered random. With a probabilistic argument, if most of the sequence can be simulated using an LFSR, this is bad for randomness.

Assuming the input sequence consists of N blocks of fixed length M one attempts the Berlekamp-Massey algorithm for each block. Let L_i be the length of the LFSR of block i. For the probability estimations to work, $500 \leq M \leq 5000$.

The theoretical mean for the L_i is

$$\mu = \frac{M}{2} + \frac{9 + (-1)^{M+1}}{36} - \frac{\frac{M}{3} + \frac{2}{9}}{2^M}$$

For each block i compute

$$T_i = (-1)^M \cdot (L_i - \mu) + \frac{2}{9}.$$

The T_i are counted in $K = 7$ classes as follows

class k	condition	π_k
0	$T_i \leq -2.5$	0.010417
1	$-2.5 < T_i \leq -1.5$	0.031250
2	$-1.5 < T_i \leq -0.5$	0.125000
3	$-0.5 < T_i \leq +0.5$	0.500000
4	$+0.5 < T_i \leq +1.5$	0.250000
5	$+1.5 < T_i \leq +2.5$	0.062500
6	$+2.5 < T_i$	0.020833

Whenever T_i is in class k count this occurence in v_k.

The v_k obey a χ^2 distribution which is checked by

$$\chi^2 = \sum_{k=0}^{K-1} \frac{(v_k - N\pi_k)^2}{N\pi_k}$$

As usual with χ^2–distributions, for the P–value, we note

$$P = \Gamma\left(\frac{K}{2}, \frac{\chi^2}{2}\right).$$

5.11 Serial Test

For a fixed pattern bit length m with $m < \log_(n) - 2$ compute the frequency of all b-bit patterns with

$$b \in \{m-2, m-1, m\}$$

in v_I where I runs over all bitstrings of length b.

Compute the following ψ values, the differences of which form a χ^2 distribution. For the three considered values of b evaluate

$$\psi_b^2 = \frac{2^b}{n} \sum_I \left(v_I - \frac{n}{2^b}\right)$$

Compute the two differences

$$\nabla\psi_m^2 = \psi_m^2 - \psi_{m-1}^2 \qquad \nabla^2\psi_m^2 = \psi_m^2 - 2\psi_{m-1} + \psi_{m-2}^2$$

For each of these differences, a P–value is computed

$$P_1 = \Gamma\left(2^{m-2}, \nabla\psi_m^2\right) \qquad P_2 = \Gamma\left(2^{m-3}, \nabla^2\psi_m^2\right)$$

which are checked as usual.

5.12 Approximate Entropy Test

This test checks the frequency of overlapping m–bit patterns for two adjacent m's, say m and $m+1$, where for the length of the pattern

$$m < \log_2(n) - 5.$$

Do a frequency count v_I of each pattern I of length m while proceeding bit-by-bit. At the end of the sequence append necessarily $m-1$ bits from the beginning.

Set

$$C_I^{(m)} = \frac{v_I}{n}$$

compute

$$\Phi^{(m)} = \sum_I C_I^{(m)} \log C_I^{(m)}$$

and the corresponding $\Phi^{(m+1)}$ value by using patterns of length $(m+1)$.

Finally, $ApEn(m) = \Phi^{(m)} - \Phi^{(m+1)}$ and

$$\chi^2 = 2n(\log 2ApEn(m))$$

with P–value

$$P = \Gamma\left(2^{m-1}, \frac{\chi^2}{2}\right).$$

5.13 Cumulative Sums (Cusum) Test

Recall from the monobit frequency test, that we evaluated the sum S of redefined bits $f(b)$ with

$$f(b) = 2b - 1.$$

Unlike as in the monobit test, we are interested in the largest excursion from the origin while summing up.

For each partial sum S_k which sums k (redefined) bits let

$$z = \max_k |S_k|$$

and

$$P = \sum_{k=(\frac{-n}{z}+1)/4}^{(\frac{n}{z}-1)/4} \left[\Phi\left(\frac{(4k+1)z}{\sqrt{n}}\right) - \Phi\left(\frac{(4k-1)z}{\sqrt{n}}\right)\right] + \sum_{k=(\frac{-n}{z}-3)/4}^{(\frac{n}{z}-1)/4} \left[\Phi\left(\frac{(4k+3)z}{\sqrt{n}}\right) - \Phi\left(\frac{(4k+1)z}{\sqrt{n}}\right)\right]$$

with $\Phi()$ as the Standard Normal Cumulative Probability Distribution Function.

This function is defined as the following well known integral

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-\frac{u^2}{2}}\, du$$

and computed with a math-library via the error function `erfc()` with the appropriate scaling:

```
double cumulativeNormal(double x)
{
       return 0.5 * erfc(-x * M_SQRT1_2);
}
```

This is found in

```
http://stackoverflow.com/questions/2785944/
cumulative-normal-distribution-function-in-objective-c
```

and has to be checked for correctness.

5.14 Random Excursions Test

This test needs at least $n = 1\,000\,000$ bits of input.

Consider the Cumulative Sums (Cusum) Test, where transformed bits are added. We are interested in the zero crossings of these partial sums. A partial sum is called *state* here and is denoted by variable s.

Subsequences starting with zero, ending with a zero and containing no other zero are called a *cycle*. Let J be the number of cycles and numerate all cycles from $i = 1, 2 \ldots, J$.

Within each cycle we count the occurence of a state s in

$$\{-4, -3, -2, -1, 1, 2, 3, 4\}$$

in count variables

$$c_{i,-4}, c_{i,-3}, \ldots, c_{i,4}.$$

After setting all count variables, we are interested in the number of cycles in which state s occurs exactly k times, for $k = 0, 1, \ldots, 5$. Denote this number by $v_k(s)$. For $k > 5$ this occurence is accumulated in v_5.

For a certain state s, we have

$$\sum_k v_k(s) = J,$$

since in each cycle some number of occurence of s exists.

For each s compute a χ^2 value.

$$\chi_s^2 = \sum_{k=0}^{5} \frac{(v_k(s) - K\pi_k(s))^2}{J\pi_k(s)}$$

where $\pi_k(s)$ is taken from the following table (by symmetry obviously $\pi_k(s) = \pi_k(-s)$):

s	$\pi_0(s)$	$\pi_1(s)$	$\pi_2(s)$	$\pi_3(s)$	$\pi_4(s)$	$\pi_5(s)$
1	0.5000	0.2500	0.1250	0.0625	0.0312	0.0312
2	0.7500	0.0625	0.0469	0.0352	0.0264	0.0791
3	0.8333	0.0278	0.0231	0.0193	0.0161	0.0804
4	0.8750	0.0156	0.0137	0.0120	0.0105	0.0733
5	0.9000	0.0100	0.0090	0.0081	0.0073	0.0656
6	0.9167	0.0069	0.0064	0.0058	0.0053	0.0588
7	0.9286	0.0051	0.0047	0.0044	0.0041	0.0531

for each state s, compute the corresponding P–value

$$P = \Gamma\left(\frac{5}{2}, \frac{\chi_s^2}{2}\right).$$

5.15 Random Excursions Variant Test

This variant of the test before is that cycles don't matter here anymore. For all integers s with $-9 \le s \le 9$, $s \ne 0$, it is counted, how often that state occurs within the random walk. As before, J is the number of cycles, i.e. the number of state $s = 0$ within the excursion.

Let $\xi(s)$ be the number of occurences of state s.

For each s compute a P–value

$$P_s = \operatorname{erfc}\left(\frac{|\xi(s) - J|}{\sqrt{2J(4|s| - 2)}}\right)$$