On the revision of NIST 800-22 Test Suites

We found that the revised code fails again, and we fixed it!

Katarzyna Anna Kowalska, Davide Fogliano, Jose Garcia Coello,
Crypta Labs, London, United Kingdom.
email: katarzyna@cryptalabs.com

***Abstract*—At Crypta Labs we are developing Quantum Random Number Generator technology and are using different random number test suites to assess the quality of our products. Among these is the NIST 800-22 suite. When testing our datasets, we found that we were consistently failing one particular test: the Overlapping Template Matching test. This was surprising to us, so we fed data from a known PRNG source into the same test and discovered that NIST approved PRNG was also failing in a similar fashion. At this point we decided to debug NIST's code. We did indeed find an error within the probability calculations and, once corrected, ran the tests again and passed. The code for this test had previously been revised by NIST due to an incorrect calculation of the probabilities, however, later in the revised source code the corrected calculations were calculated again using the originally incorrect formulas, and these overwrote the revised fix. Furthermore, the NIST 800-22 Test suite is currently under revision and our paper is a contribution towards it.**

***Index Terms*—NIST 800-22, RNG, probabilities, tests**

I. Introduction

A NEW generation of random number generators are in development. Quantum Random Number Generators are recently gaining popularity for their quality, robustness and potential speeds. Aside from being a better complement to traditional Cryptographic applications, they are suited to new protocols such as QKD BB84 where (due to the symmetric nature) a high bit rate of generation is needed. The bandwidth of these new post-quantum protocols can be limited by the bit-rate generation of the RNGs used, which is why QRNGs have a huge advantage over TRNGs. To ensure QRNG technology is sufficiently tested for this application, large amounts of data needs to be extracted from the QRNG to be scrutinized. Crypta Labs decided to extract and perform the tests on 8 Gigabits of Random Numbers, splitting them into one hundred streams of 80 Mbits. We found that there was a test in the NIST 800-22 suite for which only 72 out of 100 streams had a 'pass' rating. The expected minimum pass-rate for the sample size in the case of 100 streams is 95. After fixing issues with this test we get a 'pass' in all the streams (100).

The test in question is the 'Overlapping template matching test'. Full details of this test can can be found in NISTs [1] published documents. A breakdown of this test is as follows: A random number sequence is partitioned into independent substrings of length M. The number of occurrences of a template B (an m-bit run of ones) in each of the substrings is calculated. The chi-square statistic is then calculated on the basis of the number of occurrences of B and occurrence probabilities π_i of number of occurrence of B in a string of M bits. If the P-value of the chi-square statistic is less than the significance level, the test concludes that the tested sequence appears to be non-random. Otherwise, the test concludes that the tested sequence appears to be random. Pass rate is defined by the proportion of sequences that pass the test. In August 2021, NIST's Cryptographic Publication Review Board initiated a review process for NIST Special Publication (SP) 800-22 Rev. 1a, A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Specifically, point 4 reads: resolve inconsistencies with the SP 800-22 and SP 800-90 series.[2] We have written this paper in the hope it supports this point.

II. Materials and Methods.

In our study to determine whether the test code itself was to blame for our data failing, we decided to extract and submit data sets from 4 different known, high quality sources of randomness to the same test and analyse the results. We used our own QRNG data after conditioning with SHA256 (denoted CL QRNG) and generated data with: NIST 800-90A approved AES-256 CTR mode, our own modification of the AES-256 CTR mode with a higher rate of reseeding (denoted CL AES-256 CTR) and Cha-Cha20 from the Linux kernel.

We ran the test from the software provided by NIST at https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final

RNG	# of streams passing the test
CL QRNG	72
Cha-Cha20	75
AES-256 CTR	75
CL AES-256 CTR	77

TABLE I
Results based on the number of 80 Mbit streams that passed the Overlapping test. Note: the minimum pass-rate on 100 streams is 95, therefore all datasets failed the test.

As we can see, the results are consistent and all datasets fail the test by similar proportions.

We decided to start looking back into the NIST literature and found that the original Overlapping test is not accurate enough due to the probabilities being estimated in a too simplistic manner. NIST briefly mentions this in a section describing the technicalities of the overlapping test [1] and it transpires that the original code for this test was revised following the correction of probabilities proposed in a paper by K. Hamano and T. Kaneko [3]. In this revision, a new set of probabilities are calculated for the initial set of parameters (line 22 of the overlappingTemplateMatchings.c). However, even using this revised version of the test, all data still fails

in the same way. We decided to debug the code and found that in lines 40-44 of the overlappingTemplateMatchings.c file, the code re-calculates and overwrites the previously corrected probabilities with the imprecise probabilities again. We therefore commented out lines 40-44 so the probabilities set in the initialization of probability values are retained.

III. Results

We carried out tests on our four datasets using our modified test code. We present the results in the table below.

RNG	# of streams passing the test
CL QRNG	100
Cha-cha20	99
AES-256 CTR	99
CL AES-256 CTR	100

TABLE II
Results based on the number of 80 Mbit streams that passed the Overlapping test. Clearly, after the correction of the probabilities, all the datasets pass the test.

IV. Revision of the Statistical Test Suite Software

With the revised test code (Rev1A), the test seems to function correctly when bitstreams of ∼1Mbit are tested. However, if the bitstream tested is ∼100Mbit, this will lead to failing data.

The hard-coded Hamano and Kaneko correction found in Rev1A is only suitable for use in the specific case of datasets containing 1032 bit substrings and 5 degrees of freedom, which limits flexibility when testing larger bitstreams.

We have further extended the code (Rev1A-CL) to improve the capabilities of the test to allow for any bitstream length, length of 'String M' as well as length of 'Runs B'. This code is provided in Appendix A with a belief it should be incorporated in a future revision of the Statistical Test Suite.

We have tested our implementation and it gives the correct estimates of the probabilities in double precision variables. This is very important as, when the bitsream size increases, as does the need for precision on the decimal places of the probability calculation. We have increased the decimal places from 6 in Rev1A to 16. We have tested the performance for the initial case with M=1032, k=6 and m=9 and we point out the Hamano Kaneko method implementation is slower than the original NIST estimation (8.136 ms vs. 0.079 ms respectively, 100 times difference in this particular case). This however is not concerning as, for a given dataset, we calculate the probabilities only once; holding them in memory for using with each substring under testing.

We performed the additional tests using NIST approved AES-256 CTR data. The total size of the dataset was 8 Gbit, however the number of bitstreams within the datasets differ for each test run. We split the dataset into shorter bitstreams of 8 Mbit 1000 bistreams) and later 1.01 Mbit (7000 bitstreams). Below We present the percentage of bitstreams passing the test for Rev1A vs Rev1A-CL.

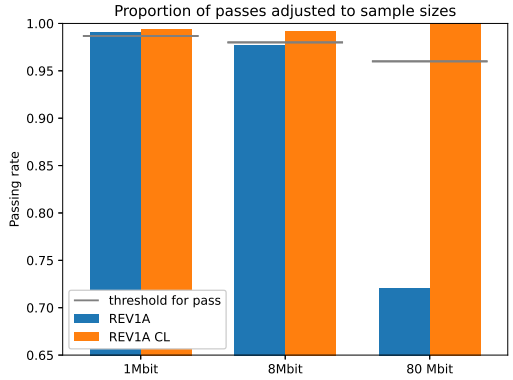

Fig. 1. Proportion of bitstreams passing the test compared to the necessary pass-rate for the whole dataset. It is important to note at this point that, within the test documentation, it states to use bitstreams of 1Mbit or more when testing. Notice that we use a dataset with a total size of 8 Gbit. Each dataset contains a different number of bitstreams and the longer the bitstream length, the smaller the pass-rate threshold proportion.

Our results show that the Rev1A is not suited for large sized bitstreams, whereas Rev1A-CL does not have the same issue. Rev1A will only pass data with a bitstream of ∼1Mbit and fails larger bitsreams which should otherwise be passing the test.

Our study adds to the results presented in [3] where it was shown that the incorrect failing of tests usually becomes visible for datasets containing 22275 bitstreams or more. In our case, the largest number of bitstreams tested was 7000, smaller than the sample size causing problems in [3] and (as expected[3] this passes. Therefore we can conclude that it was certainly the size of the bitstreams that caused the erroneous fails with Rev1A. We would also want to point out another study not focused on the same issue but mentioning a fail in the Overlapping Template Matching test. In [4] a study on the impact of keys used in different PRGBs on the performance under NIST tests is carried out. 10 keys are tested with 1000 bitstreams of 1Mbit length. Results for AES 128 bit key, OFB mode are presented from which we learn that among 10 keys one gave a fail in the Overlapping test. This suggests that even with small bitstreams in the quantity smaller than 22275 we could observe some issues. Therefore a careful reconsideration of the tests is necessary.

V. Discussion

We have found that the NIST revised code (Rev1A) contains an error, and overwrites the correctly initialised probabilities later in the code. We point out that this inaccuracy of the calculated probabilities is crucial in the failures observed in bitstreams of a large sizes. Following this line of thought, we consider it useful to not only correct the probability estimates but also study the effects of probabilities that have more than 6 decimal places. We believe for a very large bitstream (e.g. 1Gbit) the decimal place accuracy can have an impact on the calculations of p-values, such that the currently hard-coded 6 places in Rev1A might not be enough. We notice that the implementation of the accurate calculation of probabilities in

[3] has reported processing time complexity $O(M^2)$, where M is the lengths of substrings to which we partition our bistreams (in the standard case M=1032). We did not work on the optimisation of this code but implemented it with minor changes following the original version. A further research into optimisation of the algorithm or its implementation could be done.

VI. Acknowledgements

This research was carried out under AQuRand project. We thank Julio Hernandez-Castro of University of Kent for his helpful assistance and Oliver Maynard suggestions on editing.

Appendix A
Rev1A-CL Revision for overlappingTemplateMatchings.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

void CalculatePiHamanoKaneko(double pi[], int k, int N,
     int m);

int main(int argc, char **argv)
{
  double *pis;

  printf("Calculate Pi using the method by Hamano &
       Kaneko\n");

  int K,N,M;

  K = atoi(argv[1]);
  N = atoi(argv[2]);
  M = atoi(argv[3]);
  printf("k = %d, n = %d, m = %d\n", K, N, M);
  pis = calloc(K, sizeof (double));

  CalculatePiHamanoKaneko(pis, K, N, M);

  for(int i = 0; i < K; i++)
  {
    printf("Pi[%d] = %1.60f\n", i, pis[i]);
  }

}

/*
  Calculate k Pi values [0:k-1] for Overlapping Template
       Matching Test
  with given N and m using the method from the paper:
  "Correction of Overlapping Template Matching Test
       Included in NIST Randomness Test Suite"
  by Kenji Hamano and Toshinobu Kaneko.

  Paramters:
  long double *pi: pointer to array of long double to
       store the generated pi values
  int k : number of pi values to generate, ( 6 in the
       paper)
  int N : n value for overlapping templates matching test
       ,NIST recommends n = 1032
  int m : m value for overlapping templates matching test
       ,NIST recommends m = 9

  The #define _i(n) converts the index in range -1:N to
       array range 0:N+1
*/
#define _i(n) (n+1)
void CalculatePiHamanoKaneko(double pi[], int k, int N,
     int m)
{
  // alloc memory for tables
  long double **T;

  T = calloc(k, sizeof(long double *));
  if(T == NULL)
  {
    printf("\t\tTABLES:  Insufficient memory, Overlapping
         Template Matchings test aborted!\n");
    exit(1);
  }

  for(int i = 0; i < k ; i++)
  {
    T[i] = calloc((N + 2), sizeof(long double));
    if(T[i] == NULL)
    {
      printf("\t\tTABLES:  Insufficient memory,
           Overlapping Template Matchings test aborted!\n
           ");
      exit(1);
    }
  }

  // Compute T0(j) (j     n) using Eq. (2).
  for(int n = -1; n <= N; n++)
  {
    if(n == -1 || n == 0)
    {
      T[0][_i(n)] = 1;
    }
    else if( n <= (m - 1))
    {
      T[0][_i(n)] = 2 * T[0][_i(n-1)];
    }
    else
    {
      T[0][_i(n)] = 2 * T[0][_i(n-1)] - T[0][_i(n-m-1)];
    }
  }

  // Compute T1(j) (j     n) using Eq. (4) and the values
       for T0.
  for(int n = -1; n <= N; n++)
  {
    if(n <= (m-1))
    {
      T[1][_i(n)] = 0;
    }
    else if( n == m)
    {
      T[1][_i(n)] = 1;
    }
    else if( n == m + 1)
    {
      T[1][_i(n)] = 2;
    }
    else
    {
      long double sum = 0;
      for(int j = -1; j<= (n-m-1); j++)
      {
        sum += T[0][_i(j)] * T[0][_i(n-m-2-j)];
      }
      T[1][_i(n)] = sum;
    }
  }

  //Compute T  (j) (j     n) using Eq. (5) and the values
       for T0,  T   1 (2          4).
  for(int a = 2; a <= (k-2); a++)
  {
    for(int n = -1; n <= N; n++)
    {
      long double sum = 0;
      for(int j = -1; j<= (n-(2*m)-a); j++)
      {
        sum += T[0][_i(j)] * T[a-1][_i(n-m-2-j)];
      }

      T[a][_i(n)] = T[a-1][_i(n-1)] + sum;
    }
  }

  // Compute T5(n) using Eq. (1) and the values for Ti (0
       i       4)
  double pi_sum = 0;
  //Compute the first pi K values:
  for(int i = 0; i <= (k-2); i++)
  {
    pi[i] = (double)(T[i][_i(N)] / powl(2, N));
```

```
    pi_sum += pi[i];
  }

  pi[k-1] = 1 - pi_sum;

  for(int i = 0; i < k ; i++)
  {
    free(T[i]);
  }

  free(T);
}
```

REFERENCES

[1] L. E. Bassham, A. L. Rukhin, J. Soto, *et al.*, "Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications," Gaithersburg, MD, USA, Tech. Rep., 2010. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf.

[2] *Proposal to revise sp 800-22 rev. 1a — csrc*$_2$022, 2022. [Online]. Available: https://csrc.nist.gov/news/2022/proposal-to-revise-sp-800-22-rev-1a.

[3] K. Hamano and T. Kaneko, "Correction of overlapping template matching test included in nist randomness test suite," *IEICE Transactions*, vol. 90-A, pp. 1788–1792, Sep. 2007. DOI: 10.1093/ietfec/e90-a.9.1788.

[4] S.-J. Kim, K. Umeno, and A. Hasegawa, *Corrections of the nist statistical test suite for randomness*, Cryptology ePrint Archive, Report 2004/018, https://ia.cr/2004/018, 2004.