International Journal on Cryptography and Information Security (IJCIS), Vol. 14, No.1, March 2024

A universal whitening algorithm for commercial random number generators

Avval Amil[1,*] and Shashank Gupta[2,†]

[1]*Department of Computer Science and Engineering, I.I.T. Delhi, Hauz Khas, New Delhi - 110016, India*

[2]*QuNu Labs Pvt. Ltd., M. G. Road, Bengaluru, Karnataka 560025, India*

(Dated: February 12, 2024)

Abstract

In this work, we present a universal whitening algorithm using n-qubit permutation matrices to remove the imperfections in commercial random number generators without compression. Specifically, we demonstrate the efficacy of our algorithm in several categories of random number generators and its comparison with cryptographic hash functions and block ciphers.

Keywords: *Quantum Cryptography, Quantum random number generator, entropy*

DOI:10.5121/ijcis.2024.14101

I. INTRODUCTION

Whitening algorithms have demonstrated remarkable improvement in the quality of all kinds of random number generators [1]. It was shown that the raw random numbers fail some of the statistical randomness tests that pass after the application of whitening algorithms. In traditional whitening algorithms, linear combination of the raw random number sequences with the benchmark random sequence resulted in increased variance of the sequence and thus the quality of the randomness [1]. The sole objective of a whitening algorithm is to improve the quality of the random number generators.

Random number generators can be categorized depending upon the notion of randomness [2]: 1. Deterministic or pseudo-random number generators (PRNGs). 2. Epistemic or classical true random number generators (TRNGs). 3. Ontic or quantum random number generators (QRNGs). Recently a general trend towards QRNGs can be seen in the commercial random number market [3] because of their potential to harness random numbers from rather simple and intrinsic random quantum phenomena as compared to the complex chaotic classical phenomenon in classical TRNGs [4] and deterministic PRNGs [5] that are prone to entropic starvation in real-world applications. That said, all three categories of random number generators are reported as imperfect due to manufacturing bias or technological imperfections [6].

Among random number generators, PRNGs are most commonly used (e.g. C++ rand() function, Linux/dev/urandom (random)). Several other PRNGs have been proposed for resource contrained IoT devices. Baldanzi et al. recently presented a cryptographically secure deterministic random bit generators (DRBGs) by analyzing different cryptographic algorithms, such as SHA2, AES-256 CTR mode, and triple DES. In 2020, James et al. reviewed high-quality PRNGs on the basis of Kolmogorov–Anosov theory of mixing in classical mechanical systems. Xorshift is a special PRNG under non-cryptographically secure random number generators. The biggest issue with the PRNGs is the limited entropy injected through seeds resulting in the situation of entropic starvation in real-world applications [7].

The low entropy problem was addressed using hardware-based classical TRNGs that were shortly substituted by QRNGs because of the simplistic phenomenon and ontic nature of the randomness [8]. Remarkable progress has happened over the years in terms of speed, size, cost, self testable in the field of QRNGs. Several commercial QRNGs are available in the

market in several forms factors like PCI cards based on IDQ and Quintessence labs, USB, chip etc. Commercial QRNGs require postprocessing using some whitening algorithms to remove manufacturing biases in the outputs [6].

There are several postprocessing algorithms developed for improving specific random number generators [9]. Our motivation is to devise a universal algorithm that can work with every random number generator. Our algorithm utilizes the huge information space associated with the n-qubit permutation matrices to remove the bias or higher-order correlations among the generated random number sequence. Permutation matrices have been efficient in expanding entropy, maintaining Shannon perfect secrecy, and fundamental in laying the foundation of the quantum-safe cryptography [10]. Permutation matrices can be realized using traditional classical as well as quantum computing systems. Hence, any algorithmic development with permutation matrices offers huge agility in the current as well as future cryptographic infrastructure.

In the present work, we demonstrate the action of an n-qubit permutation matrices-based non-deterministic whitening algorithm in removing bias and other defects in commercial PRNGs, classical TRNGs and QRNGs. In particular, we analyse the efficacy of the whitening algorithm in improving randomness parameters of the popular PRNGs (/dev/urandom, /dev/random, C++ rand() function), ring oscillator-based classical TRNGs and commercial QRNGs, and compare it with cryptographic hash functions (SHA-256), block ciphers (DES-CBC, AES-256-CTR). We demonstrate that our algorithm works in these scenarios without compressing the random data.

The paper is organized in the following way. In Sec.(II), we recapitulate the set of n-qubit permutation matrices and the generation of an n-qubit permutation matrix using QRNG data from the Qosmos (QNu Labs Entropy-as-a-Service). In Sec.(III), we discuss the universal whitening algorithm using the generated permutation matrices. In Sec. (IV), we present our results in the following scenarios, A. PRNGs. B. Comparison with Cryptographic Hash functions C. Comparison with Block cipher DES-CBC and AES-256-CTR. D. TRNG. E. QRNGs. Finally, we conclude in Sec.(V) with some future offshoots of the present analysis.

II. PRELIMINARIES

Some definitions and software we have used to test the data created by our algorithm are described now -

Whitening Algorithms: A whitening algorithm is a technique that is applied on a stream of random numbers (entropy stream) which reduces the bias and correlation for the data (can be in the form of bytes or individual bits or both) and improves the random characteristics for the stream. In many of the commonly used whitening algorithms, multiple input bits are used to create a single output bit, and thus the size of the data reduces. As we will see in the following section, this is in contrast with our whitening algorithm which preserves the size of the data and the algorithm is non-deterministic in nature. Some famous whitening algorithms are-

- XOR: In this technique, two streams of random numbers are used to create a single entropy stream. One bit is taken from each stream and their exclusive OR is taken to give the output bit. Two bits are used up to create one bit here. This is not efficient as the length of the first stream should be the same as the length of the second stream.
- Cryptographic Hash Functions: Various cryptographic hash functions exist which also improve the randomness of the entropy stream. These hash functions (e.g. SHA-256, SHA-512 etc.) reduce the size of the output data. Hence, it decreases the possible throughput. This is currently the well-established technique for achieving information-theoretic security in QRNG.
- Von Neumann's Technique: This is a simple technique in which bits of the input stream are considered two at a time, if the two bits are the same (either '00' or '11') they are discarded, if the bits are '01' or '10, then the output bit is '0' or '1' respectively. It can be seen that this technique uses more than two bits on average to generate a single bit of the output. Hence, the possible throughput decreases.

ENT Randomness Test: This is a program used to test the randomness of a random number sequence [11]. It can be used in two modes - the bit mode which works by considering chunks of single bits, or the byte mode which considers data to be chunks of 8 bits at a time. For all our testing, we have worked with the byte mode. This test gives results for five

parameters related to random data on the basis of which the quality of the random numbers being tested can be gauged. These five parameters are - entropy, chi-square distribution, arithmetic mean, Monte Carlo approximation of π, and serial correlation coefficient. By comparing the values of these parameters with their expected value for truly random data, the quality of the random number generator is judged. Note that these tests are statistical in nature and hence give more accurate results for larger sizes of the input stream. Keeping this in mind, all the data we tested on was large in size (100MB - 1GB of random numbers).

NIST Statistical Test Suite: This is set of 15 tests (and various sub-tests) [12] developed by the National Institute of Standards and Technology, USA. It was developed after DES was cracked to choose today's AES. [13]. It provides a comprehensive set of tests for which a P-value is output. Higher P-values correspond to better-quality random numbers (a P-value of 0 corresponds to perfectly non-random numbers while a P-value of 1 corresponds to perfect randomness) and high-quality random numbers should be able to pass all of these tests.

III. WHITENING ALGORITHM

We have used a variant of the entropy expansion algorithm [14] for the purpose of whitening random data from various sources. This algorithm breaks the input data (which here is the bit-string corresponding to the binary content of the files under consideration) into chunks and then multiplies the chunk with a randomly selected permutation matrix (a matrix in which each row and each column has exactly one non-zero entry equal to one) to create the output chunk. For example, if the input chunk is '0001' which is to be multiplied with the permutation matrix shown, below, the output chunk will be '0010'.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

The output chunks are then concatenated to form another bit-string which is the binary content of the output file. For the generation of permutation matrices, we have used the Fisher-Yates shuffle algorithm, which is now given in the form of pseudocode.

Note that the generation of the matrices has a random step to it that comes from a QRNG ('RandomInt(1,N)' in the pseudocode) which is the reason why this whitening algorithm is not deterministic.

Require: RandomInt(1,N) returns a random integer between 1 and N (inclusive) when called (it is generated by an n-bit quantum random number where $N = 2^n$); Swap(a,b) swaps the values of a and b

Ensure: P is a random permutation matrix

$i \leftarrow 1$
while $i \leq N$ **do**
    $K[i] \leftarrow$ RandomInt(1,N)
    $S[i] \leftarrow i$
    $j \leftarrow 1$
    while $j \leq N$ **do**
        $P[i][j] \leftarrow 0$
        $j \leftarrow (j + 1)$
    end while
    $i \leftarrow (i + 1)$
end while
$i \leftarrow N$
while $i > 0$ **do**
    $p \leftarrow K[i]$
    $Swap(S[p], S[i])$
    $i \leftarrow (i - 1)$
end while
$i \leftarrow 1$
while $i \leq N$ **do**
    $P[i][S[i]] \leftarrow 1$
    $i \leftarrow (i + 1)$
end while

As the size of the permutation matrix (and hence the chunks) increases (N), the permutation space associated with the set of the permutation matrices widens ($N!$), and this

leads to a higher increase in the randomness of the data on applying permutations ($\log_2 N!$). Using a greater number of permutation matrices in our set also increases the entropy. The time complexity of generating all permutation matrices of size N is $\mathcal{O}(N!)$. However, we are generating a truncated set of permutation matrices, therefore the complexity is $\mathcal{O}(N)$. The whitening algorithm involves iterative multiplication of the random chunk with a permutation matrix from the set (time complexity is $\mathcal{O}(N^2)$). Hence, the overall complexity of the whitening algorithm is $\mathcal{O}(N^2)$. The whitening algorithm is a quantum resistant algorithm based on quantum permutation pad. It is robust against popular quantum and classical attacks [10].

Using entropy expansion as a whitening technique has a few differences from the other whitening algorithms described before - 1. *Equivalent size:* If the size of the entropy stream is a multiple of the dimension of the permutation matrix, then the size of the output is exactly the same as the size of the input; 2. *Non-determinism:* The output of the whitening algorithm is different each time as the selection logic of the permutation matrices comes from a QRNG that is truly random (as opposed to most whitening algorithms which are deterministic in nature).

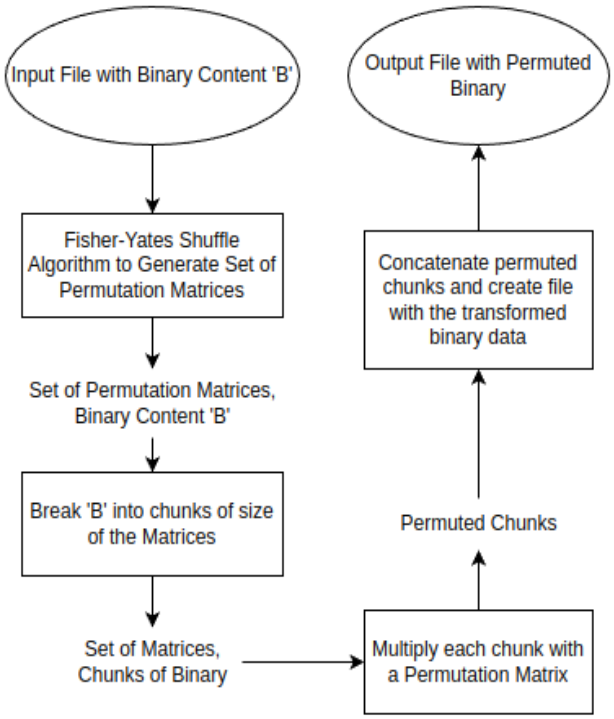

FIG. 1: Flowchart Explaining the Entropy Expansion Whitening Algorithm

IV. TESTING OF THE WHITENING ALGORITHM AND RESULTS

We tested the whitening algorithms on several random data generated by various random number generators, and also compared the results of the whitening algorithm against commonly used cryptographic hash functions like SHA-256 as well as block ciphers like AES-256 and DES. The tests we conducted are now listed -

A. Pseudo-Random Number Generators (PRNGs)

1. C++ rand() Function

We generated 1GB of random numbers using the inbuilt 'rand()' function in C++. The seed taken for the generation of these numbers was decided by using 'time.h' and the current time in seconds was used as the seed. Earlier, the algorithm used by this function was a simple multiply and shift algorithm, but now it has been replaced by a more cryptographically secure algorithm. The random numbers generated passed all the NIST tests (- note that failures in the NIST reports are indicated by an asterisk next to the test parameters). After modification by our whitening algorithm, it was found that the modified numbers still passed all the NIST tests . The ENT test results are tabulated (Table I). It can be seen that except the chi-square value, all parameters show an improvement.

2. Linux /dev/urandom

In Unix-like operating systems, /dev/urandom is a special file that can create pseudo-random numbers using data from environmental noise as a seed. 1GB of random numbers was generated using /dev/urandom and the testing was performed on both the original numbers and the modified numbers. We saw that 2 NIST tests failed for the original random numbers (which was to be expected as /dev/urandom is known for not being cryptographically secure) , while all the tests passed for the modified numbers . The modification was done using 32 matrices of 8192×8192 size. The ENT test results are tabulated (Table I). Except the serial correlation coefficient, all parameters show an improvement.

3. Linux /dev/random

Like /dev/urandom, /dev/random also creates pseudo-random numbers using data from the environment as seed, but if there is less entropy, it blocks the random numbers. This results in better quality random numbers and it was observed that all NIST tests passed for /dev/random . After modification, this was still the case and all NIST tests passed for the modified numbers . The ENT test results are tabulated (Table I). Here too, except the serial correlation coefficient, all parameters show an improvement.

Parameter	Input File 1	Output File 1	Input File 2	Output File 2	Input File 3	Output File 3	Ideal Value
Entropy	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000
Chi-Square Distribution	256.99	271.33	281.74	254.60	298.63	239.96	256.00
Arithmetic Mean	127.5024	127.5002	127.5035	127.4985	127.4964	127.5008	127.5
Monte Carlo value of Pi	3.141405155	3.141684931	3.141483117	3.141554084	3.141550664	3.141517494	3.141592653
Serial Correlation Coefficient	-0.000024	0.000022	-0.000015	0.000019	-0.000022	0.000032	0.0

TABLE I: Results for C++ rand() (File 1), /dev/urandom (File 2) and /dev/random (File 3) 1GB random numbers

Parameter	SHA-256 on C++ rand()	Modified C++ rand()	SHA-256 on /dev/urandom	Modified /dev/urandom	SHA-256 on /dev/random	Modified /dev/random	Ideal Value
Entropy	7.999999	8.000000	7.999999	8.000000	7.999999	8.000000	8.000000
Chi-Square Distribution	336.31	271.33	268.95	254.60	267.53	239.96	256.00
Arithmetic Mean	127.4986	127.5002	127.4912	127.4985	127.5020	127.5008	127.5
Monte Carlo value of Pi	3.141316661	3.141684931	3.141813093	3.141554084	3.141714298	3.141517494	3.141592653
Serial Correlation Coefficient	-0.000036	0.000022	0.000043	0.000019	0.000004	0.000032	0.0

TABLE II: Comparison of SHA-256 with Entropy Expansion for C++ rand(), /dev/urandom and /dev/random 1GB random numbers

B. Comparison with Cryptographic Hash Function SHA-256

For the 1GB random numbers generated above (using C++ rand(), /dev/urandom and /dev/random), we applied the SHA-256 hashing algorithm for every 160 bytes of these files and compared the randomness of the results with the entropy expansion whitening algorithm using the ENT test. Note that while using SHA-256 reduces the file size (in this case by a factor of 5 as 160 bits are used at a time and the output of the SHA-256 algorithm is 32 bytes or 256 bits long), entropy expansion does not reduce the file size. As entropy is determined by statistical testing, this is favourable for entropy of the resulting data as can be seen in the data compiled (Table II). For numbers generated by C++ rand(), all the parameters were observed to be better for the entropy expanded data than for the SHA-256 hashed numbers. Similarly, for numbers generated by /dev/urandom, entropy expansion showed better results than SHA-256 for all parameters. For numbers generated by /dev/random, it was seen that entropy expansion had better results for 3 parameters (entropy, arithmetic mean and Monte

Parameter	DES-CBC on C++ rand()	Modified C++ rand()	DES-CBC on /dev/urandom	Modified /dev/urandom	DES-CBC on /dev/random	Modified /dev/random	Ideal Value
Entropy	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000
Chi-Square Distribution	256.65	271.33	235.62	254.60	227.95	239.96	256.00
Arithmetic Mean	127.5003	127.5002	127.5003	127.4985	127.5029	127.5008	127.5
Monte Carlo value of Pi	3.141422988	3.141684931	3.141670578	3.141554084	3.141474486	3.141517494	3.141592653
Serial Correlation Coefficient	-0.000031	0.000022	0.000024	0.000019	0.000024	0.000032	0.0

TABLE III: Comparison of DES-CBC with Entropy Expansion for C++ rand(), /dev/urandom and /dev/random 1GB random numbers

Carlo value of π) while SHA-256 had better results for chi-square distribution and the serial correlation coefficient.

C. Comparison with Block Cipher DES-CBC

The random numbers generated by the three PRNGs were also encrypted using the block cipher DES-CBC, which was the encryption standard until it was replaced by AES. Unlike SHA-256, block ciphers like DES or AES-256 do not reduce the size of files. Randomness comparison was done using both ENT and NIST. In the ENT tests, it was observed that for C++ rand() data, entropy expansion showed better results than DES for all parameters except the chi-square distribution (Table III). For /dev/urandom data, all parameters except the arithmetic mean showed better results for entropy expansion (Table III). For /dev/random data, all parameters showed better results for entropy expansion except the serial correlation coefficient (Table III).

Parameter	AES-256 on C++ rand()	Modified C++ rand()	AES-256 on /dev/urandom	Modified /dev/urandom	AES-256 on /dev/random	Modified /dev/random	Ideal Value
Entropy	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000
Chi-Square Distribution	234.06	271.33	247.86	254.60	295.12	239.96	256.00
Arithmetic Mean	127.5005	127.5002	127.5041	127.4985	127.5037	127.5008	127.5
Monte Carlo value of Pi	3.141616169	3.141684931	3.141542811	3.141554084	3.141491916	3.141517494	3.141592653
Serial Correlation Coefficient	0.000000	0.000022	-0.000017	0.000019	0.000043	0.000032	0.0

TABLE IV: Comparison of AES-256-CTR with Entropy Expansion for C++ rand(), /dev/urandom and /dev/random 1GB random numbers

For C++ rand() data and /dev/random data, it was seen that after encryption using DES-CBC, the files passed all the NIST tests for C++ rand() and for /dev/random) which was also the case when entropy expansion was applied (for C++ rand() and for /dev/random). For /dev/urandom however, it was seen that even after encryption with DES-CBC, the data failed for one NIST test , while when using entropy expansion it passed all the tests .Note that entropy for all the files is the maximum possible so that cannot be a point of comparison here.

D. Comparison with Block Cipher AES-256-CTR

Next, the random numbers generated by the three PRNGs were also encrypted using the block cipher AES-256-CTR, which is one of the most commonly used encryption schemes. Again, ENT and NIST tests both were performed. For C++ rand() data, entropy expansion

Parameter	QRNG File 1	Output File 1	QRNG File 2	Output File 2	QRNG File 3	Output File 3	Ideal Value
Entropy	7.999990	8.000000	8.000000	8.000000	8.000000	8.000000	8.000000
Chi-Square Distribution	265.55	236.57	282.64	255.05	266.87	269.28	256.00
Arithmetic Mean	127.4810	127.4994	127.5030	127.5014	127.4977	127.5003	127.5
Monte Carlo value of Pi	3.142213394	3.141519305	3.141527487	3.141333663	3.141651890	3.141697103	3.141592653
Serial Correlation Coefficient	-0.000530	0.000017	0.000014	0.000012	-0.000031	0.000008	0.0

TABLE V: Results for Tropos QRNG File-1 (raw data file), File-2 (10% compression using Toeplitz matrix based PA) and File-3 (20% compression). Note that all files contain around 1GB of random numbers.

showed better results for chi-square distribution and arithmetic mean but AES-256-CTR showed better results for Monte Carlo value of π and the serial correlation coefficient (Table IV). For /dev/urandom data, it was seen that entropy expansion showed better results for all parameters except the serial correlation coefficient (Table IV). For /dev/random data, it was observed that entropy expansion showed better results than AES-256-CTR did for all the parameters involved. Again, entropy for all the files is maximum here so it can't be used for comparison.

For the NIST tests, all the AES-256-CTR encrypted files passed all the NIST tests (for C++ rand(), for /dev/urandom and for /dev/random encrypted with AES-256). This is the same as for the entropy expanded versions of these files (for C++ rand(), for /dev/urandom and for /dev/random). The variation of the chi-square distribution and the arithmetic mean for all of the above data was plotted (Fig. 2 and Fig. 3) and it was seen that the modified files after entropy expansion (bold green colour) gave some of the best results with respect to

Parameter	ID Quantique	Output File 1	Crypta Labs	Output File 2	Classical TRNG	Output File 3	Ideal Value
Entropy	7.999999	7.999999	7.999998	7.999998	8.000000	8.000000	8.000000
Chi-Square Distribution	231.00	265.90	247.79	260.48	240.82	285.03	256.00
Arithmetic Mean	127.4973	127.4978	127.5075	127.4961	127.5050	127.5010	127.5
Monte Carlo value of Pi	3.141995186	3.141344618	3.141686046	3.141344522	3.141373392	3.141479930	3.141592653
Serial Correlation Coefficient	-0.000092	0.000036	-0.000092	0.000073	-0.000039	-0.000024	0.0

TABLE VI: Results for ID Quantique's QRNG Data (125 MB) (File-1), Crypta Labs' QRNG Data (100 MB) (File-2) and Classical TRNG Data (1.3 GB) (File-3)

the ideal values of these quantities.

E. Classical TRNG Data

Lastly, we tested the effect of entropy expansion on some data (around 1.3 GB) generated by a non-quantum classical TRNG. The file was modified and the ENT test was performed. It was observed that except the chi-square distribution, all the randomness parameters were showing improvement (Table VI).

F. Quantum Random Number Generators (QRNGs - Tropos, ID Quantique and Crypta Labs)

The whitening effect of the entropy expansion algorithm on quantum random number generators is observed. For this, we took three files with data (around 1GB each) generated using a QRNG and with their privacy amplified to different degrees were also taken and the

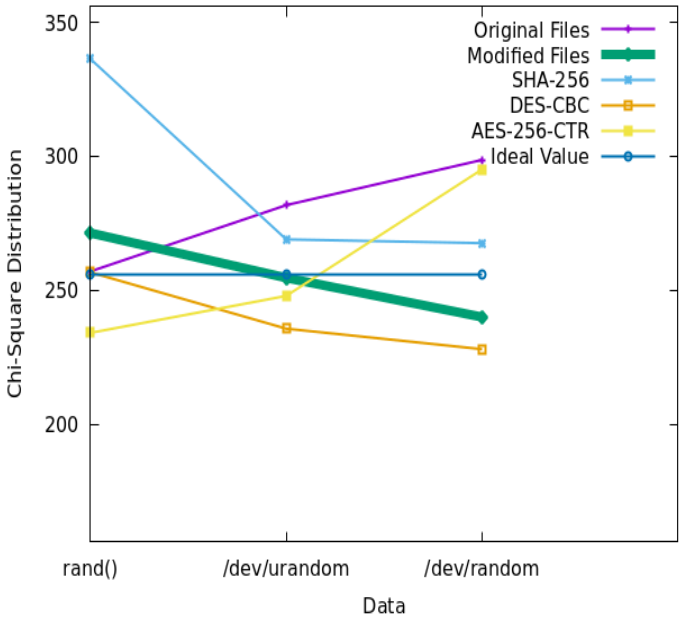

FIG. 2: Variation of Chi-Square Distribution of the Original Data and after Various Modifications

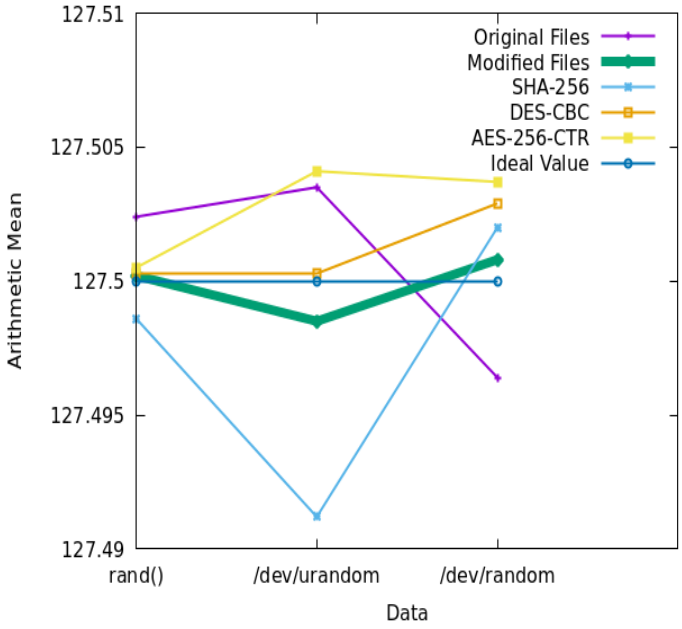

FIG. 3: Variation of Arithmetic Mean of the Original Data and after Various Modifications

ENT and NIST tests were performed on them. All the files were modified using entropy expansion using 32 matrices of size 8192×8192.

For file 1, it was observed that all the ENT parameters improved (Table V). The original

file failed two NIST tests while the modified file passed all of the NIST tests .

For file 2, all the ENT parameters with the exception of the Monte Carlo value of π showed an improvement after entropy expansion (Table V). In this case, it was seen that the original file performed poorly on the NIST test and showed a very low P-value for more than 40 tests. After entropy expansion, the randomness showed significant improvement and the modified file only failed 1 NIST test .

For file 3, after entropy expansion the arithmetic mean and serial correlation coefficient improved while the chi-square distribution and Monte Carlo value of π were better for the original file (Table V). In the NIST tests, the original data failed 2 tests , but after application of entropy expansion it was observed that the modified data passed all the NIST tests .

Next, we took 125 MB of data from ID Quantique's QRNG resource library [15] and performed the ENT test on the file and its modified version using entropy expansion. It was seen that all parameters showed an improvement after entropy expansion (Table VI).

100 MB of data was also taken from Crypta Labs [15] and similarly modified using entropy expansion. The ENT test was performed and it was observed that all parameters except the Monte Carlo value of π showed an improvement (Table VI).

V. CONCLUSIONS

Random numbers are essential for scientific investigations and technological applications. High entropic random numbers are critical for cryptography. Deterministic PRNGs and low entropic TRNGs create a situation of entropy starvation, thus exposing data for cyber attacks. Recently, QRNGs have gained much attention because of their potential to harness ontic randomness from simple quantum phenomenon. However, manufacturing bias and technological imperfections give rise to imperfect randomness even in QRNGs. Hence, there is a requirement of post-processing to remove these biases. The existing methods on one hand require compression, thus decreasing the data rate and on other hand do not improve all the randomness parameters.

In this work, we have addressed the problem of universal whitening algorithm for the commercial random number generators by using n-qubit permutation matrices. We have demonstrated the efficacy of our algorithm in several scenarios: PRNGs, classical TRNG as well as commerial QRNGs and also comparead it with popular cryptographic algorithms.

We have achieved significant improvement in the Chi-Square distribution value in majority (more than 70%) of the instances. Note that other parameters also improve almost in every instance. The modified random data files after the application of our algorithm passes the NIST SP 800-22 tests in both the cases: 1. The raw file does not pass all the tests. 2. The raw file also passes all the tests. Our algorithm performs well in every scenario, thus has the potential to be a universal whitening algorithm for the commercial random number generators.

There are several offshoots of the present work. We have demonstrated whitening mainly for 1 GB file size using 13-qubit permutation matrices (32 such matrices). One can determine the optimum matrix size and number of matrices for a given file size. The true potential of the algorithm will be shown with FPGA implementation and modification of the generated imperfect data at the runtime with the speed in Gbps. To ensure indeterminacy, one can use the raw data for the creation and selection of the permutation matrices.

ACKNOWLEDGEMENT

S.G. acknowledges the QuNu Labs Pvt. Ltd. for the financial support.

DECLARATION

A. Availability of supporting data

The datasets used and/or analysed during the current study are available at this private repository. It will be made available from the corresponding author on reasonable request.

---

[1] Cusick, Thomas W. and Pantelimon Stanica, Cryptograhic Boolean Functions and Applications, Academic Press, 2009, pp. 19; Thamrin, N.M., G. Witjaksono. et. al., An Enhanced Hardware-based Hybrid Random Number Generator For Cryptosystem, ICIME '09 Proceedings of the International Conference on Information Management and Engineering, April 03 – April 05 2009, pp. 152-156; Sam Mitchum, Digital Implementation of a True Random Number Generator, Dissertation, Virginia Commonwealth University.

[2] Manabendra Nath Bera, Antonio Acín, Marek Kuś, Morgan W Mitchell and Maciej Lewenstein, Randomness in quantum mechanics: philosophy, physics and technology, Rep. Prog. Phys. 80 124001 (2017); Vaisakh Mannalath, Sandeep Mishra, Anirban Pathak, A Comprehensive Review of Quantum Random Number Generators: Concepts, Classification and the Origin of Randomness, arXiv: 2203.00261 (2022).

[3] ID Quantique, IDQ Random Number Generation, (2017); QNu Labs, Tropos - Quantum Random Number Generator (2020); Quintessence Labs, qStream quantum random number generator (2020). V. Lovic, D.G. Marangon, M. Lucamarini, Z. Yuan, and A.J. Shields, Characterizing Phase Noise in a Gain-Switched Laser Diode for Quantum Random-Number Generation, Phys. Rev. Applied 16, 054012 (2021).

[4] Toni Stojanovski and Ljupco Kocarev, Chaos-Based Random Number Generators—Part I: Analysis, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: FUNDAMENTAL THEORY AND APPLICATIONS, VOL. 48, NO. 3, MARCH (2001); Toni Stojanovski, Johnny Pihl, and Ljupco Kocarev, Chaos-Based Random Number Generators—Part II: Practical Realization, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: FUNDAMENTAL THEORY AND APPLICATIONS, VOL. 48, NO. 3, MARCH (2001).

[5] F.James, *A review of pseudorandom number generators*, Computer Physics Communications Volume 60, Issue 3, Pages 329-344 (1990)

[6] Darren Hurley-Smith and Julio Hernandez-Castro, *Quam Bene Non Quantum: Bias in a Family of Quantum Random Number Generators*, Cryptology ePrint Archive, Paper 2017/842 (2017)

[7] Amalia Beatriz Orúe, Luis Hernández Encinas, Veronica Fernández and Fausto Montoya, A Review of Cryptographically Secure PRNGs in Constrained Devices for the IoT, Part of the Advances in Intelligent Systems and Computing book series (AISC,volume 649) (2017); Luca Baldanzi, Luca Crocetti, Francesco Falaschi, Matteo Bertolucci, Jacopo Belli, Luca Fanucci and Sergio Saponara, Cryptographically Secure Pseudo-Random Number Generator IP-Core Based on SHA2 Algorithm, Sensors 2020, 20(7), 1869 (2020). Frederick James and Lorenzo Moneta, Review of High-Quality Random Number Generators , Computing and Software for Big Science volume 4, Article number: 2 (2020); George Marsaglia, Xorshift RNGs, Journal of Statistical Software, 8(14), 1–6 (2003); Sebastiano Vigna, Further scramblings of Marsaglia's xorshift generators, arXiv:1404.0390 (2014)

[8] You-Qi Nie, Leilei Huang, Yang Liu, Frank Payne, Jun Zhang, and Jian-Wei Pan, The generation of 68 Gbps quantum random number by measuring laser phase fluctuations, Review of Scientific Instruments 86, 063105 (2015); Xiongfeng Ma, Xiao Yuan, Zhu Cao, Bing Qi and Zhen Zhang, Quantum random number generation, npj Quantum Information volume 2, Article number: 16021 (2016); Tobias Gehring, Cosmo Lupo, Arne Kordts, Dino Solar Nikolic, Nitin Jain, Tobias Rydberg, Thomas B. Pedersen, Stefano Pirandola and Ulrik L. Andersen, Homodyne-based quantum random number generator at 2.9 Gbps secure against quantum side-information, Nature Communications volume 12, Article number: 605 (2021).

[9] M. Hayashi and T. Tsurumaru, More Efficient Privacy Amplification With Less Random Seeds via Dual Universal Hash Function, IEEE Transactions on Information Theory, vol. 62, no. 4, pp. 2213-2232, (2016); D. Li, P. Huang, Y. Zhou, Y. Li and G. Zeng, Memory-Saving Implementation of High-Speed Privacy Amplification Algorithm for Continuous-Variable Quantum Key Distribution, IEEE Photonics Journal, vol. 10, no. 5, pp. 1-12, Art no. 7600712 (2018); Tang, BY., Liu, B., Zhai, YP. et al., High-speed and Large-scale Privacy Amplification Scheme for Quantum Key Distribution, Sci Rep 9, 15733 (2019).

[10] Randy Kuang and Nicolas Bettenburg, Shannon Perfect Secrecy in a Discrete Hilbert Space, IEEE International Conference on Quantum Computing and Engineering, QCE 2020, Denver, CO, USA, October 12-16, pp. 249-255, (2020); Randy Kuang and Michel Barbeau, Quantum permutation pad for universal quantum-safe cryptography, Quantum Information Processing volume 21, Article number: 211 (2022); Randy Kuang, Dafu Lou, Alex He, Alexandre Conlon, Quantum Secure Lightweight Cryptography with Quantum Permutation Pad, Advances in Science, Technology and Engineering Systems Journal, Volume 6, Issue 4, Page No 401-405, (2021); R. Kuang, D. Lou, A. He, C. McKenzie and M. Redding, Pseudo Quantum Random Number Generator with Quantum Permutation Pad, IEEE International Conference on Quantum Computing and Engineering (QCE), pp. 359-364 (2021); Shende, V.V., Synthesis of reversible logic circuits, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[11] John Walker, *Ent. A pseudo-random number sequence testing program*, ENT, (2022)

[12] National Institute of Standards and Technology, *NIST computer security resource center (CSRC)*, NIST, (2022)

[13] J K M Sadique Uz Zaman, *A Review Study of NIST Statistical Test Suite: Development of an indigenous Computer Package*, arXiv:1208.5740 (2012)

[14] Avval Amil, Shashank Gupta, *Quantum entropy expansion using n-qubit permutation matrices in Galois field*, arXiv:2207.07‘48 (2022)

[15] ID Quantique, QRNG resource library; Crypta Labs QRNG data, https://cryptalabs.com/

Avval Amil is an undergraduate at IIT Delhi in New Delhi, 110016, India. His research interests include quantum computation and theoretical aspects of computer science and physics. Amil is currently pursuing his B.Tech in CSE from IIT Delhi. Contact him at avval2801@gmail.com.

Shashank Gupta is a Senior Research Associate at QuNu Labs Pvt. Ltd. in Bangalore, 560025, India. His research interests include quantum cryptography, quantum information theory, and quantum communication. He was a part of the Integrated Ph.D. program in Physical Sciences at S. N. Bose National Centre for Basic Sciences, Kolkata, 700106, India. Contact him at shashank@qnulabs.com.