Pseudo Quantum Random Number Generator with Quantum Permutation Pad

Randy Kuang
Quantropi Inc.
Ottawa, Canada
randy.kuang@quantropi.com

Dafu Lou
Quantropi Inc.
Ottawa, Canada
dafu.lou@quantropi.com

Alex He
Quantropi Inc.
Ottawa, Canada
alex.he@quantropi.com

Chris McKenzie
Quantropi Inc.
Ottawa, Canada
chris.mckenzie@quantropi.com

Michael Redding
Quantropi Inc.
Ottawa, Canada
michael.redding@quantropi.com

***Abstract*— Cryptographic random number generation is critical for any quantum-safe encryption. Based on the natural uncertainty of some quantum processes, a variety of quantum random number generators, or QRNGs, have been created with physical quantum processes. These typically generate random numbers with good unpredictable randomness. Of course, physical QRNGs are costic and require physical integrations with computing systems. This paper proposes a pseudo quantum random number generator with a quantum algorithm called a quantum permutation pad, or QPP, leveraging the high entropy of quantum permutation space for its bijective transformation. Unlike Boolean algebra, where the size of information space is 2^n for an n-bit system, an n-bit quantum permutation space consists of $2^n!$ quantum permutation matrices, representing all quantum permutation gates over an n-bit computational basis. This permutation space holds an equivalent Shannon information entropy of $\log_2(2^n!)$. A QPP can be used to create a pseudo-QRNG or pQRNG capable of integration with any classical computing system, or directly with any application, for good-quality deterministic random number generation. Using a QPP pad with 64 8-bit permuation matrices, a pQRNG holds 107,776 bits of entropy for pseudo-random number generation, compared with 4,096 bits of entropy in Linux /dev/random. It can be used as a deterministic PRNG or as an entropy booster for other PRNGs. It can also be used as a whitening algorithm for any hardware random number generator, including QRNGs, without discarding physical bias bits.**

Keywords—QPP, quantum permutation pad, quantum permutation gates, PRNG, QRNG, pQRNG, entropy booster

I. Introduction

Random number generations can be categorized into two classes: hardware random number generators, or TRNGs, and software/pseudo random number generators, or PRNGs. An HRNG is a device which generates random numbers from a specific physical process, such as noise sampling, free running oscillators, chaos or quantum effects. These processes are generally considered to be unpredictable. Among HRNGs, quantum random number generators specifically refer to optical devices. Rarity, Owens and Tapster (1994) [1] reviewed the early status of interferometry-based quantum cryptography and compared photon-pair and faint-pulse schemes. Stefanov et al. (2000) [2] reported on their optical quantum random number generator, a simple beam splitter, where the random events were realized from the choice of single photons between two outputs of a beam splitter. Ma et al. (2016) recently published a review of quantum random number generators [3], wherein they classified QRNGs into three categories: practical QRNGs, self-testing QRNGs, and semi-self-testing QRNGs. Practical QRNGs are built on fully trusted and calibrated devices and produce good randomness at high speed. Self-testing QRNGs generate verifiable randomness without trusting the actual implementation. Semi-self-testing QRNGs provide a tradeoff between the trustworthiness of the device and the generation speed. Gehring et al., in 2020, reported ultra-fast quantum random number generation at a speed of 8 Gbps, based on quadrature measurements of vacuum fluctuations [4]. In 2021, Gehring et al. measured their homedyne-based quantum random number generator at 2.9 Gbps, using a different technique for quantum random number generation through measurements of laser phase fluctuations. Nie et al., in 2015 [6], reported extremely high generation speeds up to 68 Gbps.

Some QRNGs are commercially available on the market. ID Quantique's Quantis QRNG offers two form factors: PCI card and USB [7], with generation speeds of 4 Mbps and 16 Mbps, respectively. Quintessence Labs offers their QRNG qStream PCIe card with 8 Gbps of source quantum entropy, reduced to 1 Gbps of unconditional entropy after application of the whitening algorithm. Commercial QRNGs usually come with certain whitening algorithms to remove biases in the outputs of a physical generator.

Although QRNGs can produce truly unpredictable random numbers, they are generally expensive and unsuitable for integration in certain computing systems, such as user end devices. The most common way to generate good randomness is to use a pseudo-random number generator. James and Moneta (2020) [9] reviewed pseudo-random number generators based on the Kolmogorov–Anosov theory of mixing in classical mechanical systems. Orúe et al. (2017) [10] reported on their deep review of cryptographic secure PRNGs for IoT devices. In 2019, Baldanzi et al. presented a cryptographically secure PRNG based on a SHA2 hash algorithm [11]. They analyzed different cryptographic algorithms, such as SHA2, AES-256 CTR mode, and triple DES, in order to build deterministic random bit generators, or

DRBGs. The highest security strength being 256 bits of entropy, the DRBG in question, based on SHA256 cryptographic primitive, passed NIST randomness testing with a high pass rate. Baldanzi et al. implemented it on FPGA and ASIC standard-cell technologies; with those hardware accelerations, the cryptographic secure PRNGs demonstrated high throughput pseudo-random number generations. Mandal et al. (2013) [12] designed and analyzed a new lightweight cryptographic pseudo-random number generator called a Warbler PRNG for smart devices, which demonstrates good randomness and passes NIST randomness testing suite; however, it has a security level of only 45-bits of entropy.

Of all the PRNGs in existence, xorShift is worth a special mention, although, generally speaking, it figures among the non-cryptographically secure random number generators. Marsaglia created it in 2003 [13]; since then, multiple variations and improvements have been developed, such as xorshift* to use an invertible multiplication to its outputs, xorshift+ (64+ or 128+) to use addition for faster non-linear transformations, xoshiro, and xoroshiro with rotations in addition to additions. The unique benefit of the xorshift family of PRNGs is their high generation speed. They can simply generate pseudo-random numbers at a speed of Gigabytes per second. Vigna (2016) [15] analyzed xorshift PRNGs and found xorshift128+ to be the fastest generator capable of successfully passing BigCrush testing.

One of the major issues associated with existing PRNGs is the limited entropy injected with a seed. To our knowledge, the highest entropy accepted by a PRNG algorithm is 1,024 bits in xorshift1024+/xorshift1024* where statistical tests also failed for linearity, indicating that increasing the seed length may not fix those failures.

Kuang and Bettenburg in 2020 [16] proposed a new algorithm based on quantum permutation logic gates (a quantum permutation pad, or QPP) over a quantum computational basis. AbdAllah1, Kuang, and Huang also applied QPP to generate Just-in-Time shared keys (JIT-SK) for TLS 1.3 Zero roundtrip time (0-RTT) [17]. Kuang et al., in 2021 [18], proposed a quantum-safe lightweight cryptographic algorithm obtained by replacing SubBytes and AddRoundKey with the same QPP in an AES algorithm, and achieved a round reduction by two-thirds. Kuang and Barbeau (2021) [19] propose a universal quantum safe cryptography with QPP. This paper proposes building a pseudo quantum random number generator, or pQRNG, with QPP based on a quantum computing algorithm.

In the remaining sections, we will briefly introduce QPP (section 2), then propose a pQRNG and perform randomness analysis (section 3), with a conclusion being drawn at the end.

II. QUANTUM PERMUTATION PAD

Classical computing systems are built on Boolean algebra with a set of basic Boolean logic gates, such as AND, OR, NAND, NOR, and XOR. All are bitwise operations. Quantum computers are built on linear algebra over Hilbert space, also termed a computational basis in quantum computing, with operations represented by quantum logic gates, such as Hadamard gates and permutation gates. The mathematical expressions of quantum logic gates are all unitary and reversable square matrices over the computational basis. Quantum gates are classified into two categories: non-classical behavior gates, and classical behavior gates. The former represent quantum superpositions and entanglements and the latter represent a deterministic transformation from an input state to an output state of the system, or simply a state of permutation. For an n-qubit system with a 2^n information state represented by a Galois field $GF(2^n)$, the entire permutation state forms the symmetric group S_{2^n}, with a total of $2^n!$ unique permutations. A generic permutation gate can be physically implemented with an algorithm proposed by Shende et al. in 2003 [20], using quantum NOT, CNOT and TOFFOLI gates in a quantum computing system, and can be also mathematically expressed using a permutation matrix in classical computing systems.

An n-qubit permutation gate can be represented by a $2^n x 2^n$ permutation matrix $P[2^n, 2^n]$ over a quantum computational basis: $\{|0\rangle, |1\rangle, \ldots, |2^n-1\rangle\}$, with only one element to be 1 in each row and each column, and all others to be 0. Each permutation matrix represents a bijective mapping from input information space to output space. There exist $2^n!$ unique bijective mappings between the input and output information space over the computational basis (note: only 2^n mappings under Boolean algebra). The entire permutation matrices form a special space, called permutation space, of $2^n!$ dimensions, associated with an equivalent Shannon entropy of $e = \log_2(2^n!) \approx 2^n (n - 0.42)$ bits at a larger n. For n=8 bits, the corresponding entropy is 1,684 bits. Therefore, an n-bit permutation space can be considered as an entropy expansion from the classical Boolean information space or Galois field $GF(2^n)$ to quantum permutation space, or S_{2^n}. This huge entropy from the quantum permutation space lays a foundation for quantum safe cryptography with the properties of Shannon perfect secrecy [16].

An n-bit permutation matrix can be randomly selected through the Fisher-Yates shuffling algorithm with a true random seed of length $n2^n$ bits, as shown in Algorithm 1 for n = 8. For a QPP pad with M permutation matrices, we can repeat the Algorithm 1 for M times to create the pad with $nM2^n$ bits of entropy. A typical QPP pad with M=64 and n=8 can have an equivalent Shannon entropy equal to 107,744 bits. Such a high entropy can be used to build a pQRNG.

Algorithm 1. Pseudo code of QPP mapping from the secret key

```
-- only illustrate a single permutation matrix selection
-- state array S[256] → a permutation matrix P[256][256]
-- initialize P[256][256]to all zeros
for i from 0 to 255
      S[i] = i
-- input random key k[N] in bytes with N =256
for i from 255 down to 1 do
   j = k[i]
   swap S[j] and S[i]
for i from 0 to 255
      P[i][S[i]] = 1
```

III. PSEUDO QUANTUM RANDOM NUMBER GENERATOR

As described in section II, QPP is a quantum algorithm which can be implemented in both a quantum computing system and a classical computing system. It has been proven to be a quantum-based cryptographic algorithm with the property of Shannon perfect secrecy [16]. It is our intention to build a new quantum algorithm-based pseudo-random number generator, or pQRNG.

Figure 1 illustrates a deterministic pQRNG, with either an input seed or one directly retrieved from the local system, such as /dev/random or /dev/urandom in a Linux system. The length of the seed is 64x256 Bytes = 16KB. Thus, a pQRNG

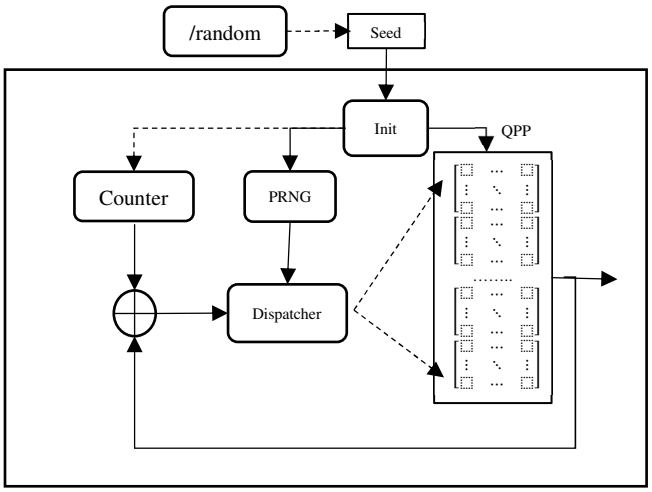

Figure 1. An illustration of a deterministic pQRNG. QPP consists of 64 8-bit permutation matrices, to be randomly selected with an external random seed of a length up to 16KB.

has a theoretical internal state of $2^{131,072}$ …amazing! The PRNG is seeded with the input seed so it can deterministically produce pseudo-random numbers to control a dispatcher in order to select a specific permutation matrix in QPP. The Counter is initialized by the supplied seed also. The feedback from the output is XORed, then randomly dispatched to a certain permutation matrix for transformation. An input byte is dispatched to the permutation matrix with index = x >> 2, or right-shift 2 bits, where x is a pseudo-random byte produced by the PRNG. The output from QPP is considered as a set of pseudo-quantum random numbers, or pQRNs.

We use the industry-recognized randomness testing suites NIST 800-22, Dieharder, and ENT to test pQRNs from pQRNG. For NIST testing, here are the testing parameters:

- Block frequency: 20,000
- Non-overlapping Template Matching: 9
- Overlapping Template Matching: 9
- Approximate Entropy: 10
- Serial: 10
- Linear Complexity: 500

Table 1 displays the results of NIST 800-22 randomness testing, wherein we generated 1GB of random numbers, stored it in a binary file and then supplied it to the NIST 800-22 testing suite. For comparison, we display these testing results together with pseudo-random numbers generated from the system rand() through C library and xorshift128+. It can be clearly seen in Table 1 that the pseudo-random numbers generated from both pQRNG and xorshift128+ pass all 15 NIST testing cases. However, the pseudo-random numbers from C library rand() failed. The same testing results are shown for Dieharder testing in Table 2. Both xorshift128+ and pQRNG have zero failures, while rand() has 2 failures.

Table 1. NIST 800-22 testing reports are illustrated with other two PRNGs. The first PRNG is from the system standard C library, the second PRNG is xorshift128+ and the third is pQRNG. pQRNG is seeded with a 16KB seed. The testing file size is 1GB.

NIST 800-22	rand()	xorshift128+	pQRNG
Frequency	Success	Success	Success
Block Frequency	Success	Success	Success
Cumulative Sums	Success	Success	Success
Runs	Success	Success	Success
Longest Run	Success	Success	Success
Rank	Success	Success	Success
FFT	Failure	Success	Success
Non-Overlapping Template	Success	Success	Success
Overlapping Template	Failure	Success	Success
Universal	Success	Success	Success
Approximate Entropy	Success	Success	Success
Random Excursions	Success	Success	Success
Random Excursions Variant	Success	Success	Success
Serial	Success	Success	Success
Linear Complexity	Success	Success	Success

Table 2. Dieharder testing is displayed with the same three PRNGs as in Table 1.

Dieharder	rand()	xorshift128+	pQRNG
Pass	99/114	113/114	108/114
Weak	3/114	1/114	6/114
Fail	2/114	0	0

The ENT randomness testing suite can generally catch the byte-level bias from supplied random data files. Hurley-Smith, Patsakis and Hernandez-Castro [21] recently identified biased QRNG random generations from a popular commercial QRNG family called Quantis [7] with ENT, where Chi square demonstrates a huge deviation from the idea value 256. In ENT testing, the Arithmetical Mean has an ideal value of 127.50 and the Serial Correlation Coefficient measures the extent to which each byte in the file depends upon the previous byte; for true random, it should be zero. Monte Carlo π indicates the Monte Carlo Value for PI to be, ideally, 3.14159265. Chi Square should be around 256 with a pvalue between 0.01 and 0.99 for good randomness data. An ENT test report with a pQRNG is tabulated in Table 3. Again, both xorshift128+ and pQRNG show very good

randomness, especially for Chi square report. The Chi square is 231.04 for pQRNG, and 263 for xorshift128+, respectively. However, rand() fails ENT testing with a Chi square of 107.35 and a p-value of 1.0, which indicates that the data is not sure to be random, although three random generators show very close testing results for Arithematical Mean, Monte Carlo π and Serial Correlation. Hence, Chi Square testing can identify whether input data is random at byte level.

Table 3. ENT testing is illustrated with the same three PRNGs as in the Table 1.

ENT	rand()	Xorshift128+	pQRNG
Entropy (bits)	8.000000	8.000000	8.000000
Chi Square	107.35	263.79	231.04
p-Value	1.00	0.34	0.86
Arith. Mean	127.5013	127.5023	127.4995
Monte Carlo π	3.141580069	3.141349333	3.141659557
Serial Correlation	0.000052	0.000019	0.000008

One interesting point from Table 3 is the serial correlation value. Of course, the ideal random data should have no correlation to one another. Therefore, the smaller of the serial correlation values posesses better randomness. Table 3 shows that the serial correlation is $8x10^{-6}$ from pQRNG, $1.9x10^{-5}$ from xorshift128+ and $5.2x10^{-5}$ from rand(), respectively.

It would be interesting to see a comparison between a physical QRNG and a pQRNG. We use a QRNG called qStream from Quintessence Labs. The qStream QRNG can generate 1 Gigabit of good random numbers per second, one of the highest throughputs on the market. Although both the qStream and pQRNG pass NIST and Dieharder randomness test suites, we would like to show the test reports for ENT, as ENT randomness testing is very sensitive to byte-level bias [21]. Table 4 lists three sets of reports, two from pStream and one from pQRNG. For the pStream QRNG, we perform ENT randomness testing with 300 MB and 1 GB of random numbers. All three reports pass ENT testing without visible byte-level bias. Chi square values are around the ideal value of 256, with good p-values. However, it is surpringly to note that testing results from the pStream 300 MB are extremely close to the pQRNG for all testing cases. It is hard to say which testing data set is more random within the acceptable p-value range of 0.01 to 0.99 for Chi Square. However, serial correlation value is worth a close look, as it shows the correlation between each byte and its previous byte. The pStream's serial correlation is $-4.0x10^{-5}$ for 300 MB and $1.7x10^{-5}$ for 1 GB, but pQRNG's serial correlation is $8x10^{-6}$. This suggests that the pQRNG demonstrates slightly lesser serial correlation than the qStream QRNG in this comparison.

Table 4. ENT randomness testing is tabulated for comparisons between the pStream physical QRNG from Quintessence Labs and the pQRNG. We report the testing results of two pStream data sizes.

ENT	pStream 300MB	pStream 1GB	pQRNG 1GB
Entropy (bits)	8.000000	8.000000	8.000000
Chi Square	231.03	259.41	231.04
p-Value	0.86	0.41	0.86
Arith. Mean	127.5035	127.5016	127.4995
Monte Carlo π	3.14141912	3.14147598	3.141659557
Serial Correlation	-0.00004	0.000017	0.000008

Figure 2 plots a slight variation of the deterministic pQRNG shown in Figure 1, used to create a quantum entropy booster, or qeBooster. As an entropy booster, the qeBooster injects entropy to improve the input PRNG's randomness. Linux /dev/random is an HRNG taking entropy from the system hardware. A typical Linux /dev/random has an entropy pool of 4,096 bits. If the pool is not full, any random number request will be blocked until the pool is full, with /dev/urandom being created to allow non-blocking random number generation. urandom is thus a special PRNG associated with /dev/random. If the entropy pool is always full, then urandom should demonstrate excellent randomness but, if the pool is always nearly empty, which may be the case with cloud servers, the corresponding state would be very poor. Where servers are greatly lacking in entropy, they will generate keys with low entropy and thus reduce security. In this case, /dev/urandom can be piped with a qeBooster to boost its entropy.

Other PRNGs can be also piped with a qeBooster to boost their entropy for cryptographic pseudo-random number generation. As an example, we take a popular fast PRNG created from C library rand() as the input prng for qeBooster.

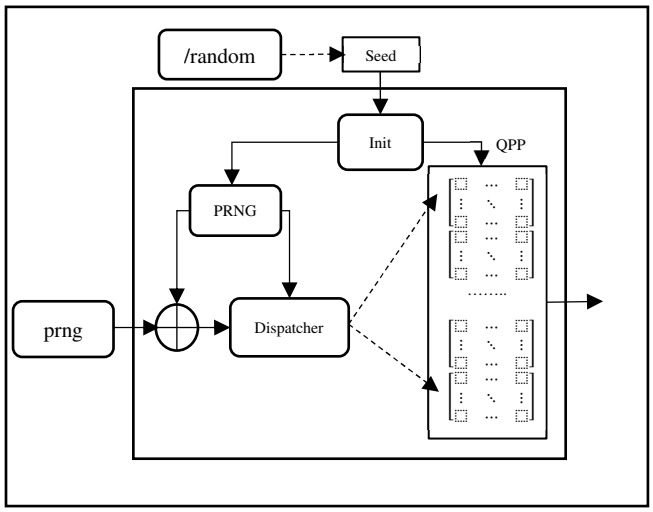

Figure 2. qeBooster behaves as an entropy booster for a low-entropy pseudo-random number generator.

Table 5. NIST testing is tabulated for rand() with a geBooster as its entropy booster.

NIST 800-22	Rand()	with qeBooster
Frequency	Success	Success
Block Frequency	Success	Success
Cumulative Sums	Success	Success
Runs	Success	Success
Longest Run	Success	Success
Rank	Success	Success
FFT	Failure	Success
Non-Overlapping Template	Success	Success
Overlapping Template	Failure	Failure
Universal	Success	Success
Approximate Entropy	Success	Success
Random Excursions	Success	Success
Random Excursions Variant	Success	Success
Serial	Success	Success
Linear Complexity	Success	Success

Tables 5 - 7 demonstrate the testing results with rand() and rand() + qeBooster. After qeBooster, the boosted pseudo random numbers from rand() demonstrate good randomness improvements:

- one failure case disappears in NIST testing;
- two failure cases in Dieharder also disappear, and one weak case is reduced;
- in ENT testing, the Chi square improves from 190.91 with p-value = 1.00, to 284.43 with p-value = 0.10; the byte-level bias is also significantly improved (N.B.: in comparison with Table 3, we note that pseudo-random numbers generated by rand() at different times show different randomness);
- in Dieharder testing, rand() shows 6 weak and 2 failed results before qeBooster, versus 5 weak and 0 failed results after qeBooster.

Table 46. Dieharder testing is tabulated for rand() with qeBooster as an entropy booster.

Dieharder	rand()	Rand() + qeBooster
Pass	106/114	109/114
Weak	6/114	5/114
Fail	2/114	0

Table 57. ENT testing is tabulated for rand() with qeBooster as an entropy booster.

ENT	rand()	Rand() + qeBooster
Entropy (bits)	7.999999	7.999998
Chi Square	190.91	284.43
p-Value	1.00	0.10
Arith. Mean	127.4939	127.5054
Monte Carlo π	3.141834126	3.141980526
Serial Corr.	0.000007	0.000022

The qeBooster posseses huge entropy (over 100Kb) and adapts QPP as its entropy injection algorithm. It can be applied to any input data, even with statistically biased plaintexts. Our intent is to demonstrate its capability with 100 MB of English characters to see how powerful it would need to be to blend any data into randonness. Table 8 tabulates the results of ENT testing, where the plaintext file fails all ENT test cases:

- the entropy per 8 bits is 4.22, indicating that the input data are independent English symbols [25];
- Chi square is 1821992676.77 with p-value 0.0001, indicating a total bias;
- the Arithematic mean is 97.9686, but the ideal value is 127.5;
- the Monte Carlo π value is 4.00, not 3.14159265, so it is a unit square, not a unit circle;
- Serial correlation is -0.138722, showing a strong correlation for each byte to its previous byte.

Following application of a qeBooster, the output file demonstrates good randomness for all ENT test cases:

- entropy per 8 bits is 7.999998, and no longer contains English characters, with 0% compression rate;
- Chi square is 233.20 with p-value 0.83 and no visible byte-level bias;
- the Arithematic mean is 127.4953, or very close to the ideal of 127.5;
- the Monte Carlo π value is 3.141981640, with an error of ~0.01%.
- Serial correlation is -9.3×10^{-5}, down from -0.139.

It can be clearly seen from this extreme case that a qeBooster injects excellent-quality entropy into input data and accords it good randomness, thanks to the quantum permutation pad. In this case, the qeBooster acts as a data encryptor with boosted data as ciphertexts of input plaintexts. We also display the results of tests using the qStream QRNG with 200 MB of random data as a comparison to the output of our qeBooster. Both sets of data show similar randomness.

Table 8. ENT testing is tabulated for statistically biased plaintext inputs with qeBooster as an entropy booster.

ENT	Plaintexts	+ qeBooster	qStream 200MB
Entropy (bits)	4.224280	7.999998	8.000000
Chi Square	1821992676.77	233.20	240.45
p-Value	0.0001	0.83	0.73
Arith. Mean	97.9686	127.4953	127.501
Monte Carlo π	4.000000000	3.141981640	3.14121543
Serial Corr.	-0.138722	- 0.000093	-0.000004

A qeBooster may be a good whitening algorithm candidate for QRNGs or HRNGs. A physical quantum random number generator naturally produces output random numbers with certain biases. In order to remove such biases, a whitening algorithm must be used to produce true random numbers. John von Neumann developed an algorithm to discard all '00' and '11' bits and convert '10' to '1' and '01' to '0'. This algorithm works nicely, but directly wastes 75% of all bits. It is possible to use a qeBooster with extremely high entropy to "smooth out" the bias. In this way, we will not waste any

valuable bits generated from a physical QRNG, then increase its physical throughput by 4-8x.

IV. Conclusion

This paper proposes to use a quantum permutation pad, or QPP, as: a fundamental building block for a pseudo quantum random number generator, or pQRNG; an entropy booster for low entropy PRNGs; and a whitening algorithm for HRNGs, including QRNGs, to increase their physical random number generation speeds. A pQRNG demonstrates excellent randomness in random number generations, at Gigabytes per second. As an entropy booster, it can dramatically improve the randomness of any input data. It has a small footprint at 2.5KB, so can be embedded in any system to boost system pseudo random number generations such as /dev/urandom in Linux. We will perform further benchmarking explorations in the near future.

References

[1] JG Rarity, PCM Owens, and PR Tapster. Quantum random-number generation and key sharing. Journal of Modern Optics, 41(12):2435-2444, 1994.

[2] Andr_e Stefanov, Nicolas Gisin, Olivier Guinnard, Laurent Guinnard, and Hugo Zbinden. Optical quantum random number generator. Journal of Modern Optics, 47(4):595-598, 2000.

[3] Ma, X., Yuan, X., Cao, Z. *et al.* Quantum random number generation. *npj Quantum Inf* **2,** 16021 (2016). https://doi.org/10.1038/npjqi.2016.21.

[4] Tobias Gehring, Cosmo Lupo, Arne Kordts, Dino Solar Nikolic, Nitin Jain, Tobias Rydberg, Thomas B. Pedersen, Stefano Pirandola, Ulrik L. Andersen. "Ultra-fast real-time quantum random number generator with correlated measurement outcomes and rigorous security certification". https://arxiv.org/abs/1812.05377v3

[5] Gehring, T., Lupo, C., Kordts, A. *et al.* Homodyne-based quantum random number generator at 2.9 Gbps secure against quantum side-information. *Nat Commun* **12,** 605 (2021). https://doi.org/10.1038/s41467-020-20813-w.

[6] Nie YQ, Huang L, Liu Y, Payne F, Zhang J, Pan JW. The generation of 68 Gbps quantum random number by measuring laser phase fluctuations. Rev Sci Instrum. 2015 Jun;86(6):063105. doi: 10.1063/1.4922417. PMID: 26133826.

[7] IQ Quantique. IDQ Random Number Generation. IQ Quantique, http://www.idquantique.com/random-number-generation/, 2017.

[8] Quintessence qStream quantum random number generator, https://www.quintessencelabs.com/products/qstream-quantum-true-random-number-generator/.

[9] James, F., Moneta, L. Review of High-Quality Random Number Generators. *Comput Softw Big Sci* **4,** 2 (2020). https://doi.org/10.1007/s41781-019-0034-3.

[10] Orúe A.B., Hernández Encinas L., Fernández V., Montoya F. (2018) A Review of Cryptographically Secure PRNGs in Constrained Devices for the IoT. In: Pérez García H., Alfonso-Cendón J., Sánchez González L., Quintián H., Corchado E. (eds) International Joint Conference SOCO'17-CISIS'17-ICEUTE'17 León, Spain, September 6–8, 2017, Proceeding. SOCO 2017, ICEUTE 2017, CISIS 2017. Advances in Intelligent Systems and Computing, vol 649. Springer, Cham. https://doi.org/10.1007/978-3-319-67180-2_65.

[11] Baldanzi, L., Crocetti, L., Falaschi, F., Bertolucci, M., Belli, J., Fanucci, L., & Saponara, S. (2020). Cryptographically Secure Pseudo-Random Number Generator IP-Core Based on SHA2 Algorithm. *Sensors (Basel, Switzerland)*, *20*(7), 1869. https://doi.org/10.3390/s20071869.

[12] Kalikinkar Mandal, Xinxin Fan and Guang Gong (2013). Warbler: A Lightweight Pseudorandom Number Generator for EPC C1 Gen2 Passive RFID Tags. International Journal of RFID Security and Cryptography (IJRFIDSC), Volume 2, Issue 2, December 2013

[13] George Marsaglia, XorShift RNG's, Journal of Statistical Software volume 8 issue 14, July 2003. doi:10.18637/jss.v008.i14

[14] Daniel Lemire and Melissa E. O'Neill. Xorshift1024*, Xorshift1024+, Xorshift128+ and Xoroshiro128+ Fail Statistical Tests for Linearity. https://arxiv.org/abs/1810.05313. 10.1016/j.cam.2018.10.019

[15] Sebastiano Vigna (2016). Further scramblings of Marsaglia's xorshift generators, https://arxiv.org/abs/1404.0390

[16] R. Kuang and N. Bettenburg, "Shannon Perfect Secrecy in a Discrete Hilbert Space," *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Denver, CO, USA, 2020, pp. 249-255, doi: 10.1109/QCE49297.2020.00039.

[17] Eslam G. AbdAllah1, Randy Kuang, and Changcheng Huang. Generating Just-in-Time Shared Keys (JIT-SK) for TLS 1.3 Zero RoundTrip Time (0-RTT). *International Journal of Machine Learning and Computing, 2021, to be pubished.*

[18] R. Kuang, D. Lou, A, He, and A. Conlon, "Quantum Safe Lightweight Cryptography with Quantum Permutation Pad", 2021 The 6th International Conference on Computer and Communication Systems (ICCCS 2021), April 23rd -26th , Chengdu, China.

[19] R. Kuang and M. Barbeau, "Quantum Permutation Pad for Universal Quantum Safe Cryptography", *manuscript submitted to Quantum Information Processing 2021.*

[20] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of reversible logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 6, pp. 710-722, June 2003.

[21] D Hurley-Smith, J Hernandez-Castro. "Quam Bene Non Quantum: Bias in a Family Quantum Random Number Generators." *IACR Cryptol. ePrint Arch*. 2017, 842.

[22] Yevgeniy Dodis, Shien Jin Ong, Manoj Prabhakaran, and Amit Sahai. On the (im) possibility of cryptography with imperfect randomness. In Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on, pages 196-205. IEEE, 2004.

[23] National Institute of Standards and Technology. NIST computer security resource center (CSRC). Retrieved from: http://csrc.nist.gov/groups/ST/toolkit/rng/index.html 13:53 07/09/2016.

[24] John Walker. Ent. A pseudo-random number sequence testing program. Retrieved from: https://www.fourmilab.ch/random/ 13:52 07/09/2016.

[25] Bill Young. Entropy of English, Retrieved from: https://www.cs.utexas.edu/~byoung/cs361/lecture35.pdf.