# 

# Cloud Quantum Computer Random Number Generation for Cryptography

Andrew Dang Khoa Pham

A Thesis in the Field of Software Engineering

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

May 2022

Copyright 2022 Andrew Pham

Abstract

Creating a true random number generator is a difficult problem that may be solved by leveraging quantum computing. A simple quantum random number generation program involves applying the Hadamard gate to all qubits then measuring each qubit for random data, repeating the program to obtain the desired bitstring length. In this thesis, we evaluate and assess the quality of random bits generated by implementing this algorithm on IBM and Rigetti quantum computers and present the implications of those results.

## Author’s Biographical Sketch

Andrew Pham graduated with a Bachelor of Science in Computer Science from California State University at Fullerton. Directly following his graduation Andrew came to Harvard University to study Software Engineering. Professionally he works as a software developer holding a commission in the US Space Force.

## Dedication

This thesis is dedicated to my parents; without their support, none of my studies would have been possible.

## Acknowledgments

I’d like to thank all the brilliant professors, lecturers, fellows, and friends at Harvard whose teachings have become the foundation of this thesis. A special thank you to Dr. Hongming Wang who guided my thesis and led me to accomplish my goals.

Table of Contents

Author’s Biographical Sketch iv

Dedication v

Acknowledgments vi

Chapter I. Introduction 11

1.1 The Random Number Generation Problem in Cryptography 11

1.2 Quantum Solutions to the RNG Problem 13

1.3 Quantum Computing Theory 14

1.3.1 Superposition 14

1.3.2 The Hadamard Gate 16

1.4 Purpose of this Study 16

Chapter II. Baseline Studies 18

2.1 IBM QX4 Tenerife 18

2.2 IBM 20Q Tokyo 19

Chapter III. Statistical Testing Methods and Algorithms 21

Chapter IV. Experiments 25

4.1 Minimum Sample Size 25

4.2 Minimum Input Size 25

4.2.1 Frequency, Block Frequency, Cumulative Sums 26

4.2.2 Longest Runs of Ones 26

4.2.3 Binary Matrix Rank 26

4.2.4 Discrete Fourier Transform 27

4.2.5 Non-overlapping Template Matching 27

4.2.6 Overlapping Template 28

4.2.7 Maurer’s Universal Statistical 28

4.2.8 Linear Complexity 29

4.2.9 Serial 29

4.2.10 Approximate Entropy 29

4.2.11 Cumulative Sums 30

4.2.12 Random Excursions and Random Excursions Variant 30

4.2.13 Overall Test Input Size Requirements 30

4.3 Testing Parameter Calculations 32

4.3.1 Frequency Within a Block 32

4.3.2 The Non-overlapping Template Matching 33

4.3.3 Serial 33

4.3.4 Approximate Entropy 34

4.3.5 Overall Testing Parameters 34

4.4 Testing Environment 35

Chapter V. Results 37

5.1 Testing on IBMQ Manila 37

5.1.1 Generating Random Numbers 37

5.1.2 Manila Results 41

5.1.3 Frequency 42

5.1.4 Block Frequency 43

5.1.5 Cumulative Sums 44

5.1.6 Runs 45

5.1.7 Longest Runs 46

5.1.8 Discrete Fourier Transform 47

5.1.9 Non-Overlapping Template 48

5.1.10 Approximate Entropy 49

5.1.11 Serial 50

5.2 Testing on IBMQ Manila 51

5.2.1 Generating Random Numbers 51

5.2.2 Aspen 9 Results 54

5.2.3 Frequency 55

5.2.4 Block Frequency 56

5.2.5 Cumulative Sums 57

5.2.6 Runs 58

5.2.7 Longest Runs 59

5.2.8 Discrete Fourier Transform 60

5.2.9 Non-Overlapping Template 61

5.2.10 Approximate Entropy 62

5.2.11 Serial 63

Chapter VI. Discussion 64

6.1 IBM Quantum Computers 64

6.2 Modern Quantum Computers 65

6.3 Comparing All the Quantum Computers 67

Chapter VII. Conclusion 68

Appendix 1. Glossary 70

References 72

## Chapter I. Introduction

Quantum computing is a developing field that has the potential to revolutionize data security by improving the random number generation process necessary for seeding modern encryption methods with truly random and unpredictable numbers. Quantum random number generation could mark one of the first practical uses of quantum computers as they possess the ability to measure inherently random systems unlike classical computers (Deshpande et al., 2020).

### 1.1 The Random Number Generation Problem in Cryptography

Cryptographic techniques used in computers today rely on a key seeded by random number generators with cryptographic strength depending on the randomness the seed. If hackers can predict or narrow down the range of the random seed, the data can be decrypted since the encryption algorithm itself is deterministic (Barker et al., 2015).

Therefore, the foundation of modern cryptographic security relies on random number generation. Bad RNGs can and have been the demise of many encrypted systems (Kelsey et al., 1998). A random number generator should output equal probability of being 0 or 1 with each bit being independent of the others. A random number generator must also be unpredictable, with no way to determine the next output. Currently, the process of generating random numbers has been divided into two main approaches, Pseudo Random Number Generators and Hardware Random Number Generators. Typically, a combination of HRNG and PRNG is used to create the random seed.

PRNGs utilize an algorithm to generate “random” numbers via a series of mathematical formulas. However, by nature of an algorithm, PRNGs are deterministic and can be predicted if the state of the PRNG is known. PRNGs require inputs called seeds which adds unpredictability. PRNGs alone are not truly random because sequences generated by PRNGs are determined by the seed. The seed itself must be random and unpredictable which brings us back to our original problem. To solve this, PRNGs are often seeded with a HRNG in a process called entropy input (Baker and Kelsey, 2015). Since PRNGs must be seeded with a random number, it may seem to defeat the purpose of using them, already needing a random number to begin the process. However, PRNGs are useful because they have better statistical properties for randomness and produce numbers faster than pure HRNGs.

HRNGs produce random numbers by taking data from a physical process. Usually, sensors are trained on statistically random signals such as environmental noise. HRNG often relies on processes that are difficult to simulate and model but may not inherently be random, such as camera data pointed at an entropic scene (Noll et al., 1998). It’s worth noting that random data has a high level of entropy, but data that has high entropy is not necessarily very random. There are requirements other than entropy that are recommended for cryptographic RNGs (E. B. Barker & Kelsey, 2015). Other methods of HRNG come from sources such as keyboard delays, mouse movement, or disk drive timing information. These methods are “weakly random” and need to be run through a randomness extractor to pass for use in cryptographic standards (Trevisan & Vadhan, 2000). Other methods of quantum random number generation also typically require randomness extraction (Haw et al., 2015). The problem of producing a HRNG that cannot be controlled, calculated, or predicted remains.



Figure 1. Wall of Entropy at Cloudflare.

This wall of lava lamps is used as an entropy generator or HRNG for encryption at website security company, Cloudflare. It’s been sensationally reported that this wall encrypts a tenth of all internet traffic. Photo licensed to Andrew Pham. © David Edelman

### 1.2 Quantum Solutions to the RNG Problem

HRNGs that rely on processes dictated by classical physics can theoretically be determined with enough information of the initial state. Quantum processes have non-deterministic fundamental unpredictability. Multiple measurements made on quantum processes in identical superimposed states will not always give the same result (Ma et al., 2016). In other words, quantum systems are inherently random. This makes them an ideal candidate as a source of entropy. Quantum RNGs could perhaps be the ultimate iteration of HRNGs and a potential solution to the RNG problem.

### 1.3 Quantum Computing Theory

A quantum computer uses the properties of quantum physics to store data and perform computations, namely the quantum properties of superposition and entanglement. They are distinguished from classical computers that process information in bits, the fundamental unit of memory which has value 0 or 1. Quantum computers utilize qubits which can represent the concept of superposition, allowing qubits to have the probability of being 0 or 1 at the same time.

### 1.3.1 Superposition

Quantum systems can be in multiple states at the same time, this property is known as superposition. The Werner Heisenberg’s Uncertainty Principal states that we cannot simultaneously know the exact position and velocity of a quantum particle. Because of the particle and wave nature of quantum systems, reducing the uncertainty of either position or velocity increases the uncertainty of the other (Heisenberg et al., 1984). Therefore, superposition is expressed as a probability of the state.

Physically, superposition can be seen in the spin of an atom. The Stern-Gerlach experiment showed that atomic scale systems intrinsically have quantum properties (Gerlach & Stern, 1922). Because of this angular-momentum quantization, the direction of the spin is in superposition until the time of observation. Observation collapses the state into one of two Eigen states, spin up or spin down, allowing us to store the data as a qubit and utilize the data as a distinct binary output, 0 for spin up and 1 for spin down.

IBM and Rigetti, the two manufacturers of quantum devices experimented on in this research, both utilize synthetic atoms, or superconducting qubits, as the physical process of superposition in their quantum devices. Other quantum computers may make use of superposition observed in other phenomena such as in photon polarization or trapped ions. These delicate systems need to be isolated from environmental interference and are kept near absolute zero to assist qubits in holding accurate value.

A picture containing tube

Description automatically generated

Figure 2. IBM Dilution Refrigerator.

Refrigerator developed by IBM to keep quantum hardware at around 15 milli-kelvins. Creative Commons BY-ND 2.0 © Graham Carlow

### 1.3.2 The Hadamard Gate

To create a random number, we want to create a quantum state where the superposition has equal probability of collapsing into spin up or spin down. In a quantum computer this is done by utilizing the Hadamard gate, an operation that puts a single qubit into superposition with equal probability of collapsing into 0 or 1 (Brylinski et al., 2019). This is commonly called Hadamard initialization. Now upon measurement, the qubit has an equal chance of collapsing into the 0 state or the 1 state. Reading out a Hadamard initialized qubit should theoretically generate 0s and 1s with equal probability. Repeating this process will give us a sequence of random bits.

A picture containing schematic

Description automatically generated

Figure 3. Hadamard Gate Visualized.

Output of quantum Hadamard gate on a qubit as visualized in IBM Q Experience

### 1.4 Purpose of this Study

Many companies sell QRNG hardware that utilizes quantum phenomena; however, QRNG hardware is not widespread for the average consumer (Symul et al., 2011). Those hoping to take advantage of QRNG for encryption typically need to purchase expensive devices or utilize research grade resources. With the introduction of Amazon Braket, a cloud-based quantum computing service, quantum computers have become commercially available by renting computing time on the quantum machines. However, it is unknown if these quantum devices produce cryptographically strong random bits. Factors such as noise and measurement error interfere with quantum computers potentially skewing the results of random quantum processes. Previous studies have shown that older quantum computers failed to produce cryptographically strong random bits.

The purpose of this study is to determine whether randomly generated numbers from newer research grade and consumer grade quantum machines produce cryptographically strong random bits by using them to generate random sequences and subjecting those sequences to statistical testing.

## Chapter II. Baseline Studies

The International Business Machines Corporation (IBM) has made quantum computers available to research scholars for years now developing their own online quantum computing portal with access to IBM quantum hardware and developing an open-source quantum software framework, Qiskit. Other scholars have performed similar testing on older IBM computers which we will use as baselines for our study. Analogous to our study, the baselines utilize quantum computers to generate a random bitstring then subject the bitstring to the NIST STS.

### 2.1 IBM QX4 Tenerife

One of the first studies to evaluate IBM computers for cryptographic random number generation was completed in the study “True Random Number Generator using Superconducting Qubits” (Ash-Saki et al., 2019). In their implementation, they use IBM’s quantum computer QX4, also known as ibmq\_tenerife, released in 2016. They noticed that the frequency test shows a deviation of 35% from the ideal ratio. In an attempt to address the issue, they swap the readout of the worst performing qubits to qubits with higher fidelity readout. However, the data generated still failed 8 of the 15 NIST randomness tests at best.

Table 1. NIST STS Results from IBM Tenerife RNG.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Run 1 | Run 2 | Run 3 | Run 4 |
| Frequency Test | Fail | Fail | Fail | Fail |
| Frequency Test within a Block | Fail | Fail | Fail | Fail |
| Runs Test | Fail | Fail | Fail | Fail |
| Test for the Longest Run of Ones in a Block | Fail | Fail | Fail | Fail |
| Binary Matrix Rank Test | Pass | Pass | Pass | Pass |
| Discrete Fourier Transform Test | Pass | Fail | Fail | Fail |
| Non-overlapping Template Matching Test | Fail | Fail | Fail | Fail |
| Overlapping Template Matching Test | Pass | Fail | Pass | Pass |
| Maurer’s Universal Statistical Test | Pass | Pass | Pass | Pass |
| Linear Complexity Test | Pass | Fail | Fail | Pass |
| Serial Test | Fail | Fail | Fail | Fail |
| Approximate Entropy Test | Fail | Fail | Fail | Fail |
| Cumulative Sums Test | Fail | Fail | Fail | Fail |
| Random Excursions Test | Pass | Pass | Pass | Pass |
| Random Excursions Variant Test | Pass | Pass | Pass | Pass |

NIST STS results generated by Ash-Saki et al. using Tenerife as the QRNG.

Error data from the Ash-Saki et al. experiment can be found on figure 3 of their report. Specifications for the retired ibmq\_tenerife computer has been removed from the IBM quantum computing site but some can still be found in the ibmq-device-information repository of the Qiskit GitHub.

### 2.2 IBM 20Q Tokyo

Another study utilizing IBM hardware for the same purpose was “Quantum Random Number Generation with the Superconducting Quantum Computer IBM 20Q Tokyo” by Kentaro Tamura and Yutaka Shikano (Tamura and Shikano., 2020). They utilized the 20-qubit quantum computer, IBM 20Q Tokyo, released in 2017. Tokyo was considered a leap in quantum computing technology with twice the quantum volume of Tenerife. Tamura and Shikano obtained a sample length of 43,560 bits. Their statistical analysis showed that the sample was biased and correlated. They observed that their sample was not uniform and failed at least 4 of the NIST Test Suite’s tests; however, they only applied the first 6 of 15 tests to the sample. They also revealed a method to pass the STS by applying the von Neumann and Samuelson randomness extractors to the sequence, the effectiveness of this method is unclear. Unlike the last study, this research was limited by sample size and thus they only performed 6 tests. As newer quantum computers are created with more qubits for computation, they will be able to generate more random numbers per cycle and increase the rate of random number generation.

Table 2. NIST STS Results from IBM Tenerife RNG.

|  |  |
| --- | --- |
| Test | p-value |
| Frequency | 0.0000 |
| Frequency Block | 0.0000 |
| Runs | 0.0000 |
| Runs Block | 0.0000 |
| Matrix Rank | 0.5285 |
| DFT | 1.0000 |

NIST STS results generated by Ash-Saki et al. using Tokyo as the QRNG.

## Chapter III. Statistical Testing Methods and Algorithms

The National Institute of Standards and Technology is a physical science lab and an agency of the US Department of Commerce. Their mission is to promote innovation and publish science in a wide variety of fields to further that mission. The NIST publishes a standard for testing random number generators along with a recommendation of cryptographically secure random number generators. These papers are widely recognized as the de facto standard for testing and implementing random number generation in cryptography.

The NIST Statistical Test Suite (STS) for Random and Pseudorandom Number Generators for Cryptographic Applications is a testing suite created by the NIST for determining whether a random number generator is suitable for cryptographic applications (Bassham et al., 2010). While others have devised tests and test suites for randomness, we have chosen this test suite because the NIST STS is from a US government agency, has strong documentation, and provides tests specifically for cryptographic applications, not just randomness in general.

The NIST STS will show P-values and number of passing sequences as part of its analysis. For the purpose of this study and other studies using the NIST STS, the null hypothesis is that “the sequence tested is random”. Tests can either fail by not having enough passing sequences or by showing an uneven distribution of P-values.

The following is a list of the NIST statistical tests for randomness and a short summary of their purpose:

1. Frequency (Monoids) Test: To determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence. The test assesses the closeness of the proportion of ones to 0.5, that is, the number of ones and zeroes in a sequence should be about the same (Chung, 1979).
2. Test for Frequency within a Block (Block Frequency): To determine whether the frequency of ones in an M-bit block is approximately M/2. This test is the same as the frequency test performed within an M-bit block. (Maclaren, 2005)
3. Runs Test: To determine whether the oscillation between substrings is too fast or too slow (Godbole, 1994).
4. Test for the Longest Run of Ones in a Block (Longest Run): To determine whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones that would be expected in a random sequence (David & Barton, 1962).
5. Random Binary Matrix Rank Test: To check for linear dependence among fixed length substrings of the original sequence (Kovalenko, 1973).
6. Discrete Fourier Transform (Spectral) Test or Fast Fourier Transform Test (FFT): To detect periodic features (i.e., repetitive patterns that are near each other) in the tested sequence that would indicate a deviation from the assumption of randomness (Bracewell, 1986).
7. Non-overlapping (Non-periodic) Template Matching: To reject sequences that exhibit too many occurrences of a given non-periodic (aperiodic) pattern (Barbour et al., 1992).
8. Overlapping Template Matching Test: To reject sequences that show deviations from the expected number of runs of ones of a given length (Chrysaphinou and Papastavridis, 1988).
9. Maurer's Universal Statistical Test (Universal Statistical Test): To detect whether the sequence can be significantly compressed without loss of information. An overly compressible sequence is non-random (Maurer, 1992).
10. Linear Complexity Test: To determine whether the sequence is complex enough to be considered random (Menezes, 1997).
11. Serial Test: To determine whether the number of occurrences of the 2m m-bit overlapping patterns is approximately the same as would be expected for a random sequence. The pattern can overlap. (Good, 1953)
12. Approximate Entropy Test: To compare the frequency of overlapping blocks of two consecutive/adjacent lengths (m and m+1) against the expected result for a random sequence. (Rukhin, 2000)
13. Cumulative Sum (CuSum) Test: To determine whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences (Revesz, 1990).
14. Random Excursions Test: To determine if the number of visits to a particular state within a cycle deviate from what one would expect for a random sequence (Spitzer, 1964).
15. Random Excursions Variant Test: To detect deviations from the expected number of visits to various states in a random walk (Baron et al., 1999

## Chapter IV. Experiments

In our experimentation, we will replicate the baseline studies on newer quantum hardware. Random numbers will be generated by operating on all qubits of a quantum computer with a Hadamard gate putting them into equal probability 0 or 1 superposition then measuring each qubit. The measurements are saved and this process is repeated until there is enough data to satisfy the sample size. After the random data is collected, the sequences will be evaluated via the NIST Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. We will perform this experiment twice, first on IBM quantum computers available to researchers and second, on Rigetti quantum computers available via Amazon Web Services to consumers.

### 4.1 Minimum Sample Size

The significance level for these tests will be set to 0.01, the minimum recommended by the NIST. “The sample should be on the order of the inverse of the significance level”, following this rule, we will have 100 sequences in our sample size. This means each test will be run on 100 different sequences (bitstreams). For our RNG to be suitable for cryptographic applications, the random sequences should pass each test in the suite with an approximately 96% pass rate for a sample size of 100 binary sequences and a uniform distribution of P-values (Bassham et al., 2010)

### 4.2 Minimum Input Size

Each test in SP 800-22 dictates a minimum number of bits required for evaluation of each test. This section describes the selection and calculation of parameters minimizing variable “n” which will have consistent definition as the length of the input bit string throughout this paper. Since we do not have unlimited use of quantum computers, we want to generate a sample that will allow us to perform the least demanding tests in the suite before continuing onto the tests that require large samples and therefore significant computing resources to generate.

### 4.2.1 Frequency, Block Frequency, Cumulative Sums

The NIST recommends the Frequency, Frequency within a Block, and Runs Test be performed with a minimum of input size of 100 bits

### 4.2.2 Longest Runs of Ones

The NIST Test for the Longest Run of Ones in a Block is structured to accommodate three minimum lengths of the input as defined by the table below where M is the length of each block.

Table 3. Longest Runs of Ones Minimum n

|  |  |
| --- | --- |
| Minimum n | M |
| 128 | 8 |
| 6272 | 128 |
| 750,000 | 100,000 |

Preset values of M accommodated by the NIST and their required minimum input sizes.

### 4.2.3 Binary Matrix Rank

The minimum input length of the Binary Matrix Rank Test is determined by the number of rows and columns in each matrix. In the testing suite, probabilities for a 32 row and column matrix have been built in.

|  |
| --- |
| n ≥ 38MQ  where M is the number of rows in each matrix  and Q is the number of columns in each matrix  M = Q = 32  using the preset 32 for M and Q  n ≥ 38,912  n should be 38,912 at minimum. |

Figure 4. Binary Matrix Rank n Calculation

Calculation for the minimum input sized required for Binary Matrix Rank Test.

### 4.2.4 Discrete Fourier Transform

The NIST recommends the Discrete Fourier Transform (Spectral) Test be performed with a minimum of input size of 1,000 bits.

### 4.2.5 Non-overlapping Template Matching

The minimum input size of the Non-overlapping Template Matching Test is determined by the length in bits of each template where the template is the target string (m), the length in bits of substring of the RNG sequence (M) and the number of independent blocks (N = n/M). The NIST imposes the following requirements:

* Requirement 1: m = 9, m must be at least 9 for meaningful results
* Requirement 2: N ≤ 100, N=8 by default satisfying this requirement
* Requirement 3: M > 0.01 • n
* Requirement 4: N = ⌊n/M⌋

Since we cannot have a fractional number of bits as the block length, n must be at least 8. This minimum makes intuitive sense because it would correspond to one block per bit.

### 4.2.6 Overlapping Template

The NIST has prebuilt the overlapping template test with parameters requiring a minimum input of 1,000,000 bits.

### 4.2.7 Maurer’s Universal Statistical

Maurer’s Universal Statistical Test requires a sequence of bits (n ≥ (Q + K) L) which are divided into two segments of L-bit blocks, where L should at least 6. For an L of 6, n should be 387,840 as prescribed by the following table:

Table 4. Maurer’s Universal Statistical Test Variables.

|  |  |  |
| --- | --- | --- |
| n | L | Q = 10 \* 2L |
| ≥ 387,840 | 6 | 640 |
| ≥ 904,960 | 7 | 1280 |
| ≥ 2,068,480 | 8 | 2560 |
| ≥ 4,654,080 | 9 | 5120 |
| ≥ 10,342,400 | 10 | 10240 |
| ≥ 22,753,280 | 11 | 20480 |
| ≥ 49,643,520 | 12 | 40960 |
| ≥ 107,560,960 | 13 | 81920 |
| ≥ 231,669,760 | 14 | 163840 |
| ≥ 496,435,200 | 15 | 327680 |
| ≥ 1,059,061,760 | 16 | 655360 |

Table of minimum inputs required to test segments of L bit blocks. Source: NIST

### 4.2.8 Linear Complexity

The NIST recommends the Linear Complexity Test be performed with a minimum of input size of 1,000,000 bits.

### 4.2.9 Serial

The serial test requires that m < ⌊log2 n⌋ – 2 where m is the length of bits in each block. We will choose m = 2 in our minimum calculation, the NIST has not recommended a value for this variable.

|  |
| --- |
| m < ⌊log2 n⌋ – 2  2 < ⌊log2 n⌋ – 2  n > 16, solving for n |

Figure 5. Linear Complexity n Calculation

Calculation for the minimum input sized required for Linear Complexity Test.

### 4.2.10 Approximate Entropy

The test requires that m < ⌊log2 n⌋ – 5 where m is the length of bits in each block. We will choose m = 2 in our minimum calculation, the NIST has not recommended a value for this variable.

|  |
| --- |
| m < ⌊log2 n⌋ – 5  2 < ⌊log2 n⌋ – 5  n > 128, solving for n |

Figure 6. Approximate Entropy n Calculation

Calculation for the minimum input sized required for Approximate Entropy Test.

### 4.2.11 Cumulative Sums

The NIST recommends each sequence in the Cumulative Sums test be run with at least 100 bits.

### 4.2.12 Random Excursions and Random Excursions Variant

The NIST recommends each sequence in the random excursions and its variant test be run with at least 1,000,000 bits.

### 4.2.13 Overall Test Input Size Requirements

Grouping the tests by minimum input size, we find 9 tests can be run with less than 1,000 bits per sequence, 11 tests can be run with less than 400,000 bits per sequence, and all 15 tests can be run with 1,000,000 bits per sequence. These minimums explain the limited scope of testing in other studies examined as baselines. In our experiment we are going to test the 9 tests that can be run with less than 1,000 bits per sequence. If most of those 9 tests pass, we can continue the analysis by spending more compute time and resources to generate more data.

Table 5. Test Input Size Requirements.

|  |  |
| --- | --- |
| Test | Minimum Input Size Recommendation for Each Sequence in Bits |
| Frequency Test | 100 |
| Frequency Test within a Block | 100 |
| Runs Test | 100 |
| Test for the Longest Run of Ones in a Block | 128 |
| Binary Matrix Rank Test | 38,912 |
| Discrete Fourier Transform Test | 1000 |
| Non-overlapping Template Matching Test | 8 |
| Overlapping Template Matching Test | 1,000,000 |
| Maurer’s Universal Statistical Test | 387,840 |
| Linear Complexity Test | 1,000,000 |
| Serial Test | 17 |
| Approximate Entropy Test | 129 |
| Cumulative Sums Test | 100 |
| Random Excursions Test | 1,000,000 |
| Random Excursions Variant Test | 1,000,000 |

Minimum input sizes for each test with those requiring less than 1,000 bits per sequence highlighted.

Text

Description automatically generated

Figure 7. NIST STS Test Selection Configuration

Terminal configuration of the 9 tests chosen to be performed in experiments.

### 4.3 Testing Parameter Calculations

Some tests in the suite have parameters that must be calculated with bitstream length in mind. We have set the bitstream length (n) for both of our experiments at 10,000 bits.

### 4.3.1 Frequency Within a Block

The Frequency Test Within a Block requires us to choose block size (M) with limitation that n ≥ MN, M ≥ 20, M > .01n, and N < 100.

|  |
| --- |
| n ≥ MN  The block size M should be selected such that  M ≥ 20, M > .01n and N < 100.  M > .01n  M > 100, substituting 10,000 for n  Therefore, we choose M = 101  This satisfies M ≥ 20  N = ⌊n/M⌋  N = ⌊10,000/101⌋ = 99  Which satisfies N < 100 |

Figure 8. Frequency Test Within a Block m Calculation

Calculation for the block size (m) in Frequency Within a Block Test.

### 4.3.2 The Non-overlapping Template Matching

The Non-overlapping Template Matching Test uses an m bit window to search for an m bit pattern. The NIST recommends m = 9 for meaningful results.

### 4.3.3 Serial

The serial test focuses on the frequency of m-bit patterns across the sequence. m must be chosen so that m < ⌊log2 n⌋ - 2

|  |
| --- |
| Choose m and n such that m < ⌊log2 n⌋ - 2.  m < ⌊log2 10,000⌋ – 2  m < 11  We will use m = 10 |

Figure 9. Serial m Calculation

Calculation for the length of bits in each block (m) in Serial Test.

### 4.3.4 Approximate Entropy

Similar to the Serial Test, the Approximate Entropy test focuses on the frequency of m-bit patterns across the sequence. m must be chosen so that m < ⌊log2 n⌋ – 5.

|  |
| --- |
| Choose m and n such that m < ⌊log2 n⌋ – 5  m < ⌊log2 10000⌋ – 5  m < 8  We will use m = 7 |

Figure 10. Approximate Entropy m Calculation

Calculation for the length of bits in each block (m) in Approximate Entropy Test.

### 4.3.5 Overall Testing Parameters

Table 6. Test Parameters.

|  |  |
| --- | --- |
| Test | Parameters |
| Frequency Test within a Block | m = 101 |
| Non-overlapping Template Matching Test | m = 9 |
| Serial Test | m = 10 |
| Approximate Entropy Test | m = 7 |

Quick reference of test parameters used in experimentation.

Text

Description automatically generated

Figure 8. NIST STS Parameter Configuration

Terminal configuration of parameters in our experiment.

### 4.4 Testing Environment

In our testing we will be using a modified NIST Test Suite called Faster Randomness Testing. Faster Randomness Testing is a project from the Faculty of Informatics at Masaryk University optimizing the NIST statistical tests for randomness to run in less time but functionally completes the same statistical tests (Sýs & Říha, 2014). This test suite is provided via online download as a Microsoft Visual Studio solution to be run locally.

Table 7. Software Versions.

|  |  |
| --- | --- |
| Faster Randomness Testing | 2.1.2 |
| Microsoft Visual Studio | Enterprise 2019 Version 16.11.1 |
| Operating System | Windows 10 Pro  Version 20 H2  OS Build 19042.1415 |

Software and versions used to run the testing.

Table 8. Testing Computer Hardware.

|  |  |
| --- | --- |
| CPU | AMD Ryzen 5 3600 6-Core Processor |
| RAM | 16 GB |

Details on hardware specifications of the computer used to run the testing.

## Chapter V. Results

### 5.1 Testing on IBMQ Manila

The IBM quantum computers Tenerife and Tokyo have since been retired but IBM has continued development of their quantum computers. We duplicate their baseline studies on a similar but more modern IBM machine, Manila (ibmq\_manila).

Table 9. ibmq\_manila specifications.

|  |  |
| --- | --- |
| Qubits | 5 |
| Quantum Volume | 32 |
| Circuit Layer Operations Per Second | 2.8 |
| Processor | Falcon r5.11 |
| Version | 1.0.22 |
| Avg. CNOT Error | 8.509e-3 |
| Avg. Readout Error | 2.976e-2 |

Technical specifications of Manila quantum machine.

### 5.1.1 Generating Random Numbers

Manila has 5 qubits. We Hadamard initialize and measure each qubit generating 5 random bits at a time. Repeating this process, we require 200,000 shots to achieve 1,000,000 random bits. The max shot size allowed is 20,000 shots. We will run 10 batches of 20,000 shots giving us a sample size of 1,000,000 bits which we can divide into 100 sequences of 10,000 bits for our testing.

000001001100000000110111011010011110111000110101110110111000110010001000000110101001010110000110011001010100011101000110011011100011010111000110111100001010001000001000010110110111100101011111011000

Figure 12. Sample of Tenerife Bitstring

First 200 bits of 1,000,000 randomly generated from ibmq\_tenerife.

# Importing standard Qiskit libraries

from qiskit import QuantumCircuit, transpile, Aer, IBMQ

from qiskit.tools.jupyter import \*

from qiskit.visualization import \*

from ibm\_quantum\_widgets import \*

from qiskit.providers.aer import QasmSimulator

from qiskit.providers.ibmq import least\_busy

# Loading your IBM Quantum account

provider = IBMQ.load\_account()

from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, execute

qreg\_q = QuantumRegister(5, 'q')

creg\_c = ClassicalRegister(5, 'c')

circuit = QuantumCircuit(qreg\_q, creg\_c)

circuit.h(qreg\_q[0])

circuit.h(qreg\_q[1])

circuit.h(qreg\_q[2])

circuit.h(qreg\_q[3])

circuit.h(qreg\_q[4])

circuit.measure(qreg\_q[0], creg\_c[0])

circuit.measure(qreg\_q[1], creg\_c[1])

circuit.measure(qreg\_q[2], creg\_c[2])

circuit.measure(qreg\_q[3], creg\_c[3])

circuit.measure(qreg\_q[4], creg\_c[4])

job = execute(circuit, provider.backend.ibmq\_manila, shots=20000, memory=True)

data = job.result().get\_memory()

Figure 13. ipynb code used to generate random numbers on IBM Quantum Lab

Code used for single batch of RNG on manila through IBM Quantum Lab.

Table 10. Manila NIST Results Output.

|  |
| --- |
| ------------------------------------------------------------------------------  RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES  ------------------------------------------------------------------------------  generator is ibmq\_tenerife  ------------------------------------------------------------------------------  C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 P-VALUE P-value(KS) PROPORTION STATISTICAL TEST  ------------------------------------------------------------------------------  95 2 1 0 1 0 0 0 1 0 0.000000 \* 0.000000 \* 0.1200 \* Frequency  42 16 15 9 7 3 4 2 2 0 0.000000 \* 0.000000 \* 0.8500 \* BlockFrequency  95 0 3 0 0 1 0 1 0 0 0.000000 \* 0.000000 \* 0.1200 \* CumulativeSums  95 1 1 1 1 0 1 0 0 0 0.000000 \* 0.000000 \* 0.1700 \* CumulativeSums  8 6 8 6 8 4 3 4 7 5 0.554420 0.252298 1.0000 Runs  31 15 13 11 10 7 6 3 4 0 0.000000 \* 0.000000 \* 0.9400 \* LongestRun  9 13 5 13 10 11 5 13 10 11 0.534146 0.632550 0.9700 FFT  38 6 9 8 6 8 4 6 7 8 0.000000 \* 0.000000 \* 0.8500 \* NonOverlappingTemplate  22 15 9 5 9 6 10 9 9 6 0.006196 0.005295 0.9200 \* NonOverlappingTemplate  25 9 9 8 10 11 7 9 8 4 0.000883 0.003781 0.9000 \* NonOverlappingTemplate  16 12 13 6 8 9 7 9 10 10 0.534146 0.093353 0.9800 NonOverlappingTemplate  18 14 15 10 13 5 9 4 9 3 0.007160 0.000117 0.9800 NonOverlappingTemplate  17 8 7 15 13 4 7 11 10 8 0.102526 0.116825 0.9600 \* NonOverlappingTemplate  18 14 7 11 9 11 6 12 9 3 0.062821 0.060349 0.9600 \* NonOverlappingTemplate  8 15 5 9 11 7 8 13 12 12 0.474986 0.535694 0.9900 NonOverlappingTemplate  21 8 15 12 7 9 10 5 6 7 0.010988 0.009579 0.9400 \* NonOverlappingTemplate  19 17 11 9 6 7 11 3 6 11 0.007694 0.001311 0.9500 \* NonOverlappingTemplate  13 8 14 7 14 1 13 8 8 14 0.051942 0.727190 0.9700 NonOverlappingTemplate  14 6 11 5 13 11 12 4 14 10 0.191687 0.800793 0.9800 NonOverlappingTemplate  22 14 3 14 9 8 12 5 9 4 0.000513 0.001491 0.9900 NonOverlappingTemplate  14 6 10 11 11 14 15 6 5 8 0.213309 0.113568 0.9700 NonOverlappingTemplate  13 10 11 11 8 6 13 6 15 7 0.437274 0.510465 0.9600 \* NonOverlappingTemplate  7 10 6 10 11 7 16 13 10 10 0.534146 0.135016 0.9700 NonOverlappingTemplate  14 8 11 12 9 9 11 6 11 9 0.867692 0.761821 0.9800 NonOverlappingTemplate  10 9 16 7 11 10 14 6 8 9 0.494392 0.397303 0.9700 NonOverlappingTemplate  14 9 11 10 11 2 12 10 8 13 0.350485 0.651675 0.9800 NonOverlappingTemplate  13 13 4 9 11 8 11 8 12 11 0.637119 0.731549 0.9800 NonOverlappingTemplate  9 7 8 10 5 12 6 13 13 17 0.181557 0.038103 0.9700 NonOverlappingTemplate  15 12 9 8 10 9 9 13 5 10 0.637119 0.633219 0.9600 \* NonOverlappingTemplate  4 8 9 8 13 13 12 8 11 14 0.455937 0.110910 1.0000 NonOverlappingTemplate  17 6 11 7 11 6 8 15 10 9 0.202268 0.437018 0.9700 NonOverlappingTemplate  20 4 8 8 11 11 15 6 9 8 0.023545 0.151467 0.9700 NonOverlappingTemplate  9 7 15 7 14 10 11 12 10 5 0.437274 0.725058 0.9900 NonOverlappingTemplate  15 12 7 5 11 6 9 14 9 12 0.334538 0.202854 0.9800 NonOverlappingTemplate  6 7 9 9 14 14 11 16 7 7 0.249284 0.176044 0.9900 NonOverlappingTemplate  11 10 5 9 12 8 8 10 10 17 0.455937 0.328072 0.9500 \* NonOverlappingTemplate  12 7 10 6 7 12 14 7 9 16 0.319084 0.310808 0.9900 NonOverlappingTemplate  27 10 8 11 8 8 8 9 7 4 0.000055 \* 0.000873 0.9500 \* NonOverlappingTemplate  16 11 10 13 4 6 13 7 11 9 0.224821 0.174181 0.9500 \* NonOverlappingTemplate  7 13 2 11 14 13 9 8 12 11 0.224821 0.205738 0.9900 NonOverlappingTemplate  12 15 5 7 10 6 14 14 7 10 0.213309 0.508630 0.9800 NonOverlappingTemplate  10 7 9 10 11 9 11 12 9 12 0.987896 0.519090 0.9800 NonOverlappingTemplate  13 11 6 10 14 9 12 8 8 9 0.779188 0.797253 0.9700 NonOverlappingTemplate  11 8 9 10 11 11 7 12 15 6 0.719747 0.753638 1.0000 NonOverlappingTemplate  12 6 10 8 10 11 8 13 14 8 0.759756 0.750544 0.9800 NonOverlappingTemplate  5 9 7 10 14 13 9 10 13 10 0.637119 0.080730 0.9900 NonOverlappingTemplate  9 8 15 6 7 11 10 12 11 11 0.719747 0.686784 0.9800 NonOverlappingTemplate  11 6 10 6 11 11 17 11 8 9 0.437274 0.442986 0.9800 NonOverlappingTemplate  4 11 6 12 14 13 12 11 8 9 0.419021 0.089537 1.0000 NonOverlappingTemplate  12 10 10 8 11 13 11 13 6 6 0.739918 0.325649 0.9700 NonOverlappingTemplate  12 6 11 17 7 14 10 8 9 6 0.236810 0.383687 0.9900 NonOverlappingTemplate  9 9 5 8 5 11 15 15 11 12 0.262249 0.011875 0.9900 NonOverlappingTemplate  [30 NonOverlappingTemplate tests omitted from this table]  35 21 10 4 10 5 4 5 5 1 0.000000 \* 0.000000 \* 0.8800 \* ApproximateEntropy  19 16 9 12 9 9 9 7 8 2 0.016717 0.001734 0.9800 Serial  8 7 10 17 7 8 12 12 12 7 0.383827 0.676133 0.9900 Serial |

Analysis report of statistical testing performed on random numbers generated by Tenerife. \* indicates failure.

### 5.1.2 Manila Results

The random numbers generated by Manila failed 6/9 of the randomness tests we ran. As a reminder, the null hypothesis is that “the sequence tested is random”. Small P-values indicate a significant result telling us to reject the null hypothesis, that the sequences are not random. The minimum pass rate for each statistical test is approximately 96% for a sample size of 100 binary sequences. The test suite is designed so that p-values are uniformly distributed under the null hypothesis, we should see uniform distribution for passing tests. Tests can either fail via proportional failure, where less than 96% of the binary sequences pass, and/or a uniformity failure, where P-values are not uniformly distributed.

Table 11. Manila NIST STS Results.

|  |  |
| --- | --- |
| Test | Result |
| Frequency Test | Fail |
| Frequency Test within a Block | Fail |
| Runs Test | Pass |
| Test for the Longest Run of Ones in a Block | Fail |
| Discrete Fourier Transform Test | Pass |
| Non-overlapping Template Matching Test | Fail |
| Serial Test | Pass |
| Approximate Entropy Test | Fail |
| Cumulative Sums Test | Fail |

Manila STS Results at a glance with close failures being marked in yellow.

### 5.1.3 Frequency

12/100 sequences passed the Frequency Test, this is well below the minimum pass rate. The frequency test checks the for the number of 0s and 1s in the entire sequence. We can see in the raw data that there are 20,000 more 0s than 1s in the million bits of random numbers making 0s 2% more likely than 1s.

Figure 14. P-Value Distribution of Frequency Test on Manila

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.1.4 Block Frequency

85/100 sequences passed the Block Frequency Test, this is below the minimum pass rate. In our test specifically we test the proportion of 1s within 101-bit blocks. Within each of the blocks, the proportion of 1s is not in line with what is expected of a random sequence.

Figure 15. P-Value Distribution of Block Frequency Test on Manila

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.1.5 Cumulative Sums

12/100 sequences passed the Cumulative Sums Test going forward and 17/100 sequences passed going backwards. This means that the distribution of 1s and 0s are not evenly dispersed throughout the sequences, sometimes all appearing early in the sequence, sometimes all appearing late.

Figure 16. P-Value Distribution of Cumulative Sums Test on Manila (Both Directions)

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.1.6 Runs

100/100 sequences passed the Runs Test making it the only perfect test. The number of uninterrupted sequences of identical bits is as expected for a random sequence and the P-value distribution is uniform.

Figure 17. P-Value Distribution of Runs Test on Manila

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.1.7 Longest Runs

94/100 sequences passed the Longest Run Test making it close to but not above the minimum pass rate. This means that the longest run of ones is not consistent with the length of the longest run we would expect of a random sequence.

Figure 18. P-Value Distribution of Longest Runs Test on Manila

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.1.8 Discrete Fourier Transform

97/100 sequences passed the Discrete Fourier Transform Test, passing the test by one over the minimum pass rate. The test did not detect periodic features in the sequences that would deviate from the assumption of randomness.

Figure 19. P-Value Distribution of Discrete Fourier Transform Test on Manila

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.1.9 Non-Overlapping Template

The Non-overlapping template test was run 147 times, once for each template. The test rejects sequences exhibiting too many or few of a given aperiodic pattern. Of those templates 33/147 (about 22%) failed to pass with a 96% rate. One template had all sequences pass; however, the p-values were not uniformly distributed, meaning that there was a low probability of these values occurring randomly, resulting in a failure for that test. This brings the overall failures to 34/147 (about 23%). We will show this chart below, omitting the charts for all other templates, to demonstrate the case of a uniformity failure rather than the more typical proportional failure or both.

Figure 20. P-Value Distribution of Non-Overlapping Template Test on Manila

The bars represent the number of sequences out of 100 that had P-value in that column’s range. This is an example of where the minimum pass rate for the test was exceeded; however, the p-values were not uniformly distributed resulting in an overall test fail for that template.

### 5.1.10 Approximate Entropy

88/100 sequences passed the Approximate Entropy Test, failing to meet the minimum pass rate. The frequency of overlapping blocks of two consecutive lengths is not what is expected of a random sequence.

Figure 21. P-Value Distribution of Approximate Entropy Test on Manila

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.1.11 Serial

98/100 sequences passed the Serial Test for the first P-value and 99/100 sequences passed for the second P-value, note this test is run twice corresponding to the two P-values generated by the test. This test specifically shows that the number of occurrences of 1024 10-bit overlapping patterns is as expected for a random sequence. Every 10-bit pattern has the same chance of appearing as every other 10-bit pattern.

Figure 22. P-Value Distribution of Serial Test on Manila (Both P-values)

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.2 Testing on IBMQ Manila

Amazon Web Services recently allowed for the purchase of compute time on the Rigetti quantum computer. We will be using the Rigetti Aspen 9 computer. This service has no computation limit, is publicly available, and is delivered via AWS allowing for potentially easy integration in cryptographic applications. Successfully passing the NIST STS would open the door for effective cryptographic quantum RNG as a service.

Table 12. Rigetti Aspen 9 Specifications.

|  |  |
| --- | --- |
| Qubits | 32 |
| Median T1 | 33µs |
| Median T2 | 16µs |
| Median 1Q gate time | 48ns |
| Median 2Q gate time | 168ns |
| Median T1 | 33µs |
| Median T2 | 16µs |
| Median 1Q Fidelity | 99.39% |
| Median 2Q Fidelity | 94.28% |

Technical specifications of Manila quantum machine.

### 5.2.1 Generating Random Numbers

Aspen 9 has 32 qubits meaning it can generate over 6 times as many random numbers per cycle than Manila. We Hadamard initialize and measure each qubit generating 32 random bits at a time. Repeating this process, we generate 1,000,000 bits by running 31,250 shots of the circuit. Unlike with the research grade IBM computer, we can run as many shots as we pay for and don’t need to queue up batches. Those 1,000,000 bits are divided into 100 sequences of length 10,000 bits.

111000010000000000000000000000001010000111001000110101111111011110000010101111101001011111001001100101001110101011000101000000101001100010000000100111111110000000101101111000101110110100010011011101

Figure 23. Sample of Aspen 9 Bitstring

First 200 bits of 1,000,000 randomly generated from Rigetti Aspen 9.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 24. ipynb code used to generate random numbers on AWS

Code used to generate all RNG data on Rigetti Aspen 9 through AWS.

Table 13. Aspen 9 NIST Results Output.

|  |
| --- |
| ------------------------------------------------------------------------------  RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES  ------------------------------------------------------------------------------  generator is AWS Rigetti Aspen 9  ------------------------------------------------------------------------------  C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 P-VALUE P-value(KS) PROPORTION STATISTICAL TEST  ------------------------------------------------------------------------------  78 7 6 5 1 0 0 2 1 0 0.000000 \* 0.000000 \* 0.4900 \* Frequency  39 18 12 5 7 4 0 8 6 1 0.000000 \* 0.000000 \* 0.8900 \* BlockFrequency  79 12 3 3 1 1 0 0 0 1 0.000000 \* 0.000000 \* 0.5200 \* CumulativeSums  77 8 2 4 4 1 3 0 0 1 0.000000 \* 0.000000 \* 0.5300 \* CumulativeSums  20 7 8 8 7 10 6 3 4 7 0.003577 0.008133 0.9625 Runs  21 11 5 7 10 9 9 11 10 7 0.051942 0.013528 0.9500 \* LongestRun  27 13 8 10 8 6 6 11 4 7 0.000015 \* 0.000020 \* 0.9200 \* FFT  33 10 9 12 8 7 4 9 6 2 0.000000 \* 0.000003 \* 0.8800 \* NonOverlappingTemplate  29 15 5 7 9 9 8 9 5 4 0.000000 \* 0.000015 \* 0.9300 \* NonOverlappingTemplate  31 11 6 6 9 6 6 9 7 9 0.000000 \* 0.000027 \* 0.8800 \* NonOverlappingTemplate  18 13 8 13 6 8 8 9 7 10 0.213309 0.084848 0.9200 \* NonOverlappingTemplate  17 7 12 10 9 5 8 13 9 10 0.334538 0.317470 0.9100 \* NonOverlappingTemplate  23 13 3 7 10 9 7 10 10 8 0.002971 0.010393 0.9300 \* NonOverlappingTemplate  15 8 8 15 5 6 12 11 6 14 0.137282 0.633015 0.9700 NonOverlappingTemplate  9 10 10 14 12 5 8 8 9 15 0.534146 0.910210 1.0000 NonOverlappingTemplate  22 16 7 8 11 4 6 12 4 10 0.000757 0.000919 0.9300 \* NonOverlappingTemplate  16 7 5 12 11 7 14 12 7 9 0.249284 0.652749 0.9800 NonOverlappingTemplate  14 7 10 9 8 6 10 9 18 9 0.262249 0.440944 0.9700 NonOverlappingTemplate  14 13 6 8 8 11 12 7 12 9 0.657933 0.563893 0.9600 \* NonOverlappingTemplate  14 14 10 13 6 10 12 9 2 10 0.181557 0.100874 1.0000 NonOverlappingTemplate  16 11 11 13 6 10 15 4 8 6 0.108791 0.037850 1.0000 NonOverlappingTemplate  17 10 5 12 4 11 12 5 17 7 0.016717 0.151456 0.9500 \* NonOverlappingTemplate  8 3 6 13 7 18 13 9 10 13 0.048716 0.007634 0.9900 NonOverlappingTemplate  17 5 10 7 8 14 12 9 7 11 0.224821 0.488799 0.9500 \* NonOverlappingTemplate  16 11 4 10 6 8 10 11 14 10 0.275709 0.518702 0.9800 NonOverlappingTemplate  12 10 9 13 8 16 7 7 5 13 0.304126 0.362412 0.9500 \* NonOverlappingTemplate  13 12 12 7 11 10 7 9 10 9 0.924076 0.599024 0.9800 NonOverlappingTemplate  13 13 12 8 8 11 9 8 7 11 0.867692 0.302940 0.9600 \* NonOverlappingTemplate  14 6 7 7 9 10 14 13 6 14 0.289667 0.067818 0.9600 \* NonOverlappingTemplate  8 13 7 7 6 9 10 14 15 11 0.437274 0.080706 0.9800 NonOverlappingTemplate  16 10 10 12 6 12 12 14 7 1 0.048716 0.035726 0.9700 NonOverlappingTemplate  8 14 7 10 12 12 14 10 6 7 0.554420 0.250070 0.9900 NonOverlappingTemplate  10 10 10 10 11 13 5 10 13 8 0.851383 0.846987 0.9800 NonOverlappingTemplate  15 7 8 10 10 8 16 6 9 11 0.383827 0.623691 0.9500 \* NonOverlappingTemplate  13 9 10 10 10 11 10 11 8 8 0.991468 0.842202 0.9700 NonOverlappingTemplate  11 7 14 9 9 10 14 13 4 9 0.437274 0.517445 0.9800 NonOverlappingTemplate  11 3 5 8 13 13 15 11 9 12 0.171867 0.013465 0.9900 NonOverlappingTemplate  19 10 10 12 11 10 9 7 5 7 0.162606 0.023013 0.9300 \* NonOverlappingTemplate  10 10 8 8 12 12 6 10 12 12 0.911413 0.688997 0.9600 \* NonOverlappingTemplate  18 9 7 7 11 13 15 7 6 7 0.085587 0.205029 0.9100 \* NonOverlappingTemplate  14 6 13 7 8 13 10 12 6 11 0.494392 0.868158 0.9700 NonOverlappingTemplate  5 10 7 13 7 13 11 10 11 13 0.616305 0.205738 0.9900 NonOverlappingTemplate  14 6 11 11 6 9 12 11 9 11 0.759756 0.923674 0.9900 NonOverlappingTemplate  19 6 10 11 17 9 5 7 5 11 0.013569 0.047071 0.9600 \* NonOverlappingTemplate  11 6 9 13 9 12 11 10 9 10 0.946308 0.786548 0.9800 NonOverlappingTemplate  8 7 6 2 19 14 13 11 8 12 0.013569 0.001781 1.0000 NonOverlappingTemplate  12 10 8 10 9 14 9 6 8 14 0.719747 0.930108 0.9900 NonOverlappingTemplate  9 8 8 14 7 15 6 12 8 13 0.419021 0.773550 0.9900 NonOverlappingTemplate  12 3 7 10 7 11 14 6 13 17 0.062821 0.078332 0.9900 NonOverlappingTemplate  12 8 8 10 5 10 11 11 11 14 0.779188 0.174448 0.9800 NonOverlappingTemplate  11 8 12 9 11 8 12 8 14 7 0.851383 0.974684 0.9900 NonOverlappingTemplate  5 13 8 9 13 9 8 13 14 8 0.514124 0.618080 0.9900 NonOverlappingTemplate  [30 NonOverlappingTemplate tests omitted from this table]  41 16 12 8 7 4 4 4 0 4 0.000000 \* 0.000000 \* 0.9400 \* ApproximateEntropy  18 10 15 12 15 12 8 6 1 3 0.001296 0.000013 \* 0.9900 Serial  5 12 8 12 10 14 10 13 6 10 0.554420 0.794833 1.0000 Serial |

Analysis report of statistical testing performed on random numbers generated by Aspen 9. \* indicates failure.

### 5.2.2 Aspen 9 Results

The random numbers generated by Manila failed 7/9 of the randomness tests we ran. As a reminder, the null hypothesis is that “the sequence tested is random”. Small P-values indicate a significant result telling us to reject the null hypothesis, that the sequences are not random. The minimum pass rate for each statistical test is approximately 96% for a sample size of 100 binary sequences. The test suite is designed so that p-values are uniformly distributed under the null hypothesis, we should see uniform distribution for passing tests. Tests can either fail via proportional failure, where less than 96% of the binary sequences pass, and/or a uniformity failure, where P-values are not uniformly distributed.

Table 14. Aspen 9 NIST STS Results.

|  |  |
| --- | --- |
| Test | Result |
| Frequency Test | Fail |
| Frequency Test within a Block | Fail |
| Runs Test | Pass |
| Test for the Longest Run of Ones in a Block | Fail |
| Discrete Fourier Transform Test | Fail |
| Non-overlapping Template Matching Test | Fail |
| Serial Test | Pass |
| Approximate Entropy Test | Fail |
| Cumulative Sums Test | Fail |

Aspen 9 STS Results at a glance with close failures being marked in yellow.

### 5.2.3 Frequency

49/100 sequences passed the Frequency Test, this is well below the minimum pass rate. The frequency test checks the for the number of 0s and 1s in the entire sequence. We can see in the raw data that there are 20,000 more 0s than 1s in the million long bits of random numbers making 0s 2% more likely than 1s.

Figure 25. P-Value Distribution of Frequency Test on Aspen

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.2.4 Block Frequency

89/100 sequences passed the Block Frequency Test, this is below the minimum pass rate. In our test specifically we test the proportion of 1s within 101-bit blocks. Within each of the blocks, the proportion of 1s is not in line with what is expected of a random sequence.

Figure 26. P-Value Distribution of Block Frequency Test on Aspen

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.2.5 Cumulative Sums

52/100 sequences passed the Cumulative Sums Test going forward and 53/100 sequences passed going backwards. This means that the distribution of 1s and 0s are not evenly dispersed throughout the sequences, sometimes all appearing early in the sequence, sometimes all appearing late.

Figure 27. P-Value Distribution of Cumulative Sums Test on Aspen (Both Directions)

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.2.6 Runs

96/100 sequences passed the Runs Test just making it over the minimum required proportion. The number of uninterrupted sequences of identical bits is as expected for a random sequence and the P-value distribution is uniform.

Figure 28. P-Value Distribution of Runs Test on Aspen

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.2.7 Longest Runs

95/100 sequences passed the Longest Run Test making it close to but not above the minimum pass rate. This means that the longest run of ones is not consistent with the length of the longest run we would expect of a random sequence. This test is an example of passing uniformity but not proportionally, most failing tests fail both.

Figure 29. P-Value Distribution of Longest Runs Test on Aspen

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.2.8 Discrete Fourier Transform

92/100 sequences passed the Discrete Fourier Transform Test, failing the test by 4 below the minimum pass rate. The test detected periodic features in the sequences that would deviate from the assumption of randomness.

Figure 30. P-Value Distribution of Discrete Fourier Transform Test on Aspen

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.2.9 Non-Overlapping Template

The Non-overlapping template test was run 147 times, once for each template. The test rejects sequences exhibiting too many or few of a given aperiodic pattern. Of those templates 39/147 (about 26%) failed to pass with a 96% rate. We saw that again, one template had a proportional pass with a uniformity failure, brining the total failures up to 40/147 (about 27%). A passing template is shown in the chart below with all others omitted.

Figure 31. P-Value Distribution of Non-Overlapping Template Test on Aspen

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.2.10 Approximate Entropy

94/100 sequences passed the Approximate Entropy Test, failing to meet the minimum pass rate by a close margin. The frequency of overlapping blocks of two consecutive lengths is not what is expected of a random sequence.

Figure 32. P-Value Distribution of Approximate Entropy Test on Aspen

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

### 5.2.11 Serial

99/100 sequences passed the Serial Test for the first P-value and 100/100 sequences passed for the second P-value, note this test is run twice corresponding to the two P-values generated by the test. This test specifically shows that the number of occurrences of 1024 10-bit overlapping patterns is as expected for a random sequence. Every 10-bit pattern has the same chance of appearing as every other 10-bit pattern.

Figure 33. P-Value Distribution of Serial Test on Aspen (Both P-values)

The bars represent the number of sequences out of 100 that had P-value in that column’s range.

## Chapter VI. Discussion

### 6.1 IBM Quantum Computers

Starting with our baseline, the newer Manila performed about the same as the older IBM machines in RNG. Of the 5 tests run by both baselines and in our experiment, 4 tests produced the same results. Of the 9 tests run on both Tenerife and Manila, 7 produced the same results. Deviations from the baselines include:

* Manila was the only one to pass the Runs Test when both others had failed.
* Manila also passed the Serial test when its much older predecessor, Tenerife, had failed.

Table 15. IBM Quantum Computer STS Result Comparisons.

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Tenerife  (best run) | Tokyo | Manila |
| Frequency Test | Fail | Fail | Fail |
| Frequency Test within a Block | Fail | Fail | Fail |
| Runs Test | Fail | Fail | Pass |
| Test for the Longest Run of Ones in a Block | Fail | Fail | Fail |
| Binary Matrix Rank Test | Pass | Pass |  |
| Discrete Fourier Transform Test | Pass | Pass | Pass |
| Non-overlapping Template Matching Test | Fail |  | Fail |
| Overlapping Template Matching Test | Pass |  |  |
| Maurer’s Universal Statistical Test | Pass |  |  |
| Linear Complexity Test | Pass |  |  |
| Serial Test | Fail |  | Pass |
| Approximate Entropy Test | Fail |  | Fail |
| Cumulative Sums Test | Fail |  | Fail |
| Random Excursions Test | Pass |  |  |
| Random Excursions Variant Test | Pass |  |  |

Pass / Fail comparisons of STS on IBM machine Quantum RNG outputs.

### 6.2 Modern Quantum Computers

Looking at the two modern machines we performed experiments on, they perform very similarly in terms of STS pass / fail with the only deviation in the Discrete Fourier Transform Test where Manila passed and Aspen failed, Aspen failing by a close margin.

Table 16. Modern Quantum Computer STS Result Comparisons.

|  |  |  |
| --- | --- | --- |
| Test | Manila | Aspen 9 |
| Frequency Test | Fail | Fail |
| Frequency Test within a Block | Fail | Fail |
| Runs Test | Pass | Pass |
| Test for the Longest Run of Ones in a Block | Fail | Fail |
| Discrete Fourier Transform Test | Pass | Fail |
| Non-overlapping Template Matching Test | Fail | Fail |
| Serial Test | Pass | Pass |
| Approximate Entropy Test | Fail | Fail |
| Cumulative Sums Test | Fail | Fail |

Pass / Fail comparisons of STS on modern quantum computer RNG outputs.

Though the Aspen 9 lost to Manila in one more test, the proportion of passing sequences slightly favors the Aspen 9 overall. The Aspen 9 performs much better in the Frequency and Cumulative Sums test though still failing both tests. In terms of test performance overall, it seems that each computer has its own strengths with Manila pulling slightly ahead in 3 tests and Aspen being slightly to significantly ahead in 4 tests.

Table 17. Modern Quantum Computer STS Proportion Results.

|  |  |  |
| --- | --- | --- |
| Test | Manila | Aspen 9 |
| Frequency Test | 12 | 49 |
| Frequency Test within a Block | 85 | 89 |
| Runs Test | 100 | 96 |
| Test for the Longest Run of Ones in a Block | 94 | 95 |
| Discrete Fourier Transform Test | 97 | 92 |
| Non-overlapping Template Matching Test | 85 | 88 |
| Serial Test | 98 | 99 |
| Approximate Entropy Test | 88 | 94 |
| Cumulative Sums Test | 12 | 52 |

Number of passing sequences out of 100. Large differences in performance are highlighted. If more than one test was run, the lowest proportion is shown. 96 is the minimum pass rate.

### 6.3 Comparing All the Quantum Computers

None of the raw outputs of any of the quantum machines examined are ready to be used as a RNG for cryptographic applications. The newer quantum computers showed slight improvements, passing the Runs and Serial Tests while their predecessors failed.

Table 18. All STS Result Comparisons.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Tenerife  (best run) | Tokyo | Manila | Aspen 9 |
| Frequency Test | Fail | Fail | Fail | Fail |
| Frequency Test within a Block | Fail | Fail | Fail | Fail |
| Runs Test | Fail | Fail | Pass | Pass |
| Test for the Longest Run of Ones in a Block | Fail | Fail | Fail | Fail |
| Binary Matrix Rank Test | Pass | Pass |  |  |
| Discrete Fourier Transform Test | Pass | Pass | Pass | Fail |
| Non-overlapping Template Matching Test | Fail |  | Fail | Fail |
| Overlapping Template Matching Test | Pass |  |  |  |
| Maurer’s Universal Statistical Test | Pass |  |  |  |
| Linear Complexity Test | Pass |  |  |  |
| Serial Test | Fail |  | Pass | Pass |
| Approximate Entropy Test | Fail |  | Fail | Fail |
| Cumulative Sums Test | Fail |  | Fail | Fail |
| Random Excursions Test | Pass |  |  |  |
| Random Excursions Variant Test | Pass |  |  |  |

Pass / Fail comparisons of STS on all quantum computer RNG outputs.

## Chapter VII. Conclusion

Cryptography is the foundation of modern information security. By evaluating the suitability of random number generators, we secure ourselves against that vector of attack. Comparing quantum computers of the past 5 years, it seems there still is progress to be made to make quantum computers more suited for random number generation. Both quantum machines failed most of the NIST Statistical Test Suite in line with other baseline studies performed within the past few years. Theoretically the quantum state of a qubit should collapse to 0 or 1 with equal probability and therefore produce random numbers. Why exactly quantum computers fail to produce random numbers is not fully understood. It has been observed that various noise sources such as gate, dephasing, decoherence, and readout errors have hindered QRNGs in the past (Ash-Saki et al., 2019).

Random number generation has various applications from sampling to simulation. For the purposes of cryptography, a flawed random number generator leaves data vulnerable to attack. While Hadamard initializing and measuring qubits in quantum computers should be capable of generating random numbers in theory, this method applied to the Rigetti and IBM quantum computers cannot currently be used as a reliable source for cryptographic random number generation.

Noise impacts every quantum system and contributes to the correlated and biased numbers generated by quantum computers (Resch and Karpuzcu, 2021). Conditions for each computer also change over time and these tests could be performed to narrow down the factors contributing to the statistical correlation of generated numbers. Future studies could explore these experiments as newer quantum computers increase their fidelity and decrease noise.

## Appendix 1. Glossary

*CSPRNG*: Cryptographically Secure Pseudorandom Number Generator

*Hadamard Gate:* A quantum logic gate that acts on a single qubit, creating a superposition. The gate maps the qubit basis states  and  to two superposition states with equal weight of the computational basis states (Brylinski et al., 2019).

The Hadamard gate maps the state  to Shape

Description automatically generated with low confidence and  to Shape

Description automatically generated with low confidence

The Hadamard Matrix: Shape

Description automatically generated with medium confidence

*HRNG*: Hardware Random Number Generators, a device that generates random numbers from a physical process. Also called True Random Number Generators.

*NIST*: National Institute of Standards and Technology, a United States government agency that publishes cryptographic standards and guidelines.

*NIST STS*: The National Institute of Standards and Technology Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications

*PRNG*: Pseudo-Random Number Generator, a process that generates numbers that looks random but are deterministic.

*QRNG*: Quantum Random Number Generator, a HRNG that utilizes the fundamental randomness of quantum mechanics to produce random numbers.

*Qubit:* A quantum analogue of a bit encoding quantum information. A qubit is a two-state quantum mechanical system allowing the bit to be in the two classical states or represented in superposition of both states simultaneously.

*Randomness Extractor:* A function applied to the output of a weak entropy source to generate a uniform, highly random output. Also known as a unbiasing algorithm.

*RNG*: Random Number Generator, a process that generates random numbers that cannot be predicted better than random chance.

*Seed:* a number used to initialize a PRNG. The result of a PRNG is determined by the seed; if the same seed is used, the PRNG will produce the same number.

*Shots:* Number of times the algorithm is run by the quantum computer.

## References

Ash-Saki, A., Alam, M., & Ghosh, S. (2019). True Random Number Generator using Superconducting Qubits. *2019 Device Research Conference (DRC)*.

Barbour, Holst, L., & Janson, S. (1992). *Poisson approximation*. Clarendon Press; Oxford University Press.

Barker, E. (2020a). Guideline for using cryptographic standards in the federal government: *NIST Special Publication*.

Barker, E. (2020b). Recommendation for key management: *NIST Special Publication*.

Barker, E. B., & Kelsey, J. M. (2015). Recommendation for Random Number Generation Using Deterministic Random Bit Generators. *NIST Special Publication*.

Baron, & Rukhin, A. L. (1999). Distribution of the number of visits of a random walk. *Communications in Statistics. Stochastic Models*, 15(3), 593–597.

Bassham, L. E., Rukhin, A. L., Soto, J., Nechvatal, J. R., Smid, M. E., Barker, E. B., Leigh, S. D., Levenson, M., Vangel, M., Banks, D. L., Heckert, N. A., Dray, J. F., & Vo, S. (2010). A statistical test suite for random and pseudorandom number generators for cryptographic applications. *NIST Special Publication.*

Bracewell, R. N., & Bracewell, R. N. (1986). The Fourier transform and its applications (Vol. 31999, pp. 127-134). New York: McGraw-Hill. Brylinski, R. K., & Chen, G. (2019). *Mathematics of Quantum Computation (Computational Mathematics)* (1st ed.). Chapman and Hall/CRC.

Chrysaphinou, & Papastavridis, S. (1988). A limit theorem on the number of overlapping appearances of a pattern in a sequence of independent trials. *Probability Theory and Related Fields*, 79(1), 129–143.

Chung, K.L. (1978). Elementary Probability Theory with Stochastic Processes. Springer New York.

David, F. N., & Barton, D. E. (1962). Combinatorial chance. *Economica*, 29(115), 332.

Deshpande, D. S., Nirala, A. K., & Salau, A. O. (2020). Implications of Quantum Superposition in Cryptography: A True Random Number Generation Algorithm. *Information and Communication Technology for Intelligent Systems*, 419–431.

Gerlach, W., & Stern, O. (1922). Der Experimentelle Nachweis der Richtungsquantelung im Magnetfeld. Zeitschrift fur Physik, 9(1), 349–352.

Godbole, & Papastavridis, S. G. (1994). *Runs and patterns in probability* (Vol. 283.). Kluwer Academic.

Good, I. J. (1953, April). The serial test for sampling numbers and other tests for randomness. In Mathematical Proceedings of the Cambridge Philosophical Society (Vol. 49, No. 2, pp. 276-284). Cambridge University Press.

Haw, J., Assad, S., Lance, A., Ng, N., Sharma, V., Lam, P., & Symul, T. (2015). Maximization of Extractable Randomness in a Quantum Random-Number Generator. *Physical Review Applied*, *3*(5), 1.

Heisenberg, W., Blum, W., Dürr H.-P, &amp; Rechenberg, H. (1984). Gesammelte Werke. Springer. Ma, X., Yuan, X., Cao, Z., Qi, B., & Zhang, Z. (2016). Quantum random number generation. *Npj Quantum Information*, *2*(1).

Kelsey, J., Schneier, B., Wagner, D., &amp; Hall, C. (1998). Cryptanalytic attacks on pseudorandom number generators. *Fast Software Encryption*, 168–188.

Kovalenko. (1973). Distribution of the Linear Rank of a Random Matrix*. Theory of Probability and Its Applications*, 17(2), 342–346.

Maclaren. (2005). Cryptographic pseudo-random numbers in simulation. In *Fast Software Encryption* (pp. 185–190). Springer Berlin Heidelberg.

Maurer, U. M. (1992). A universal statistical test for random bit generators. Journal of cryptology, 5(2), 89-105.

Menezes, van Oorschot, P. C., & Vanstone, S. A. (1997). *Handbook of Applied Cryptography* (1st ed.). CRC Press.

Noll, L. C. N., Mende, R. G. M., & Sisodiya, S. S. (1998). *Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of a chaotic system* (U.S. Patent 5,732,138). U.S. Patent and Trademark Office.

Resch, & Karpuzcu, U. R. (2021). Benchmarking Quantum Computers and the Impact of Quantum Noise. *ACM Computing Surveys*, 54(7), 1–35.

Revesz, P. (1990). *Random walk in random non-random environments / Pal Revesz*. World Scientific.

Rukhin, A. L. (2000). Approximate entropy for testing randomness. Journal of Applied Probability, 37(1), 88-100.

Spitzer, F. (1964). The classification of random walk. In Principles of random walk (pp. 1-53). Springer, New York, NY.

Symul, T., Assad, S. M., &amp; Lam, P. K. (2011). Real time demonstration of high bitrate quantum random number generation with coherent laser light. *Applied Physics Letters*, 98(23), 231103.

Sýs, M., & Říha, Z. (2014). Faster Randomness Testing with the NIST Statistical Test Suite. *Security, Privacy, and Applied Cryptography Engineering*, 272–284.

Tamura, K., & Shikano, Y. (2020). Quantum Random Number Generation with the Superconducting Quantum Computer IBM 20Q Tokyo. *IACR Cryptol. ePrint Arch., 2020*, 78.

Trevisan, L., & Vadhan, S. (2000). Extracting randomness from sampleable distributions. *Proceedings 41st Annual Symposium on Foundations of Computer Science*.