Welcome to the Harvard   
Extension School Template!

Here are a few pointers to get you started:

1. Be sure autosave (Word for Windows: File > Options > Save; Word for Mac: Word menu > Preferences > Save) is enabled (it is by default). The default autosave interval is 10 minutes, so you may want to adjust that if you prefer more frequent saves.
2. For some large edit moves you may want to go as far as saving your current draft with a version number (1.1, 1.2, and so on) before you make the move in case you change your mind later.
3. You may [want to see the invisible characters](http://clients.wordwizardsinc.com/harvard/paragraph-pane.html#showHidePara)—the symbols that don’t print—to see where your paragraphs end or to view section or page breaks. Select the Home tab in the Ribbon and click the ¶ button in the Paragraph pane.
4. If you would prefer your page numbers are at the upper-right instead of the bottom center of the page, you’ll need to adjust the header and footer for the main body of your text. To do that, scroll down to the bottom of page 2 and double-click near the page number; select and cut (ctrl-x or cmd-x) the page number; paste (ctrl-v or cmd-v) the page number in the header area; delete any extra returns (there should just be one); go to the Home tab and click the Align Right button in the Paragraph pane (); Close the header (it will also close the footer).
5. **Remember to delete this page before submission**. (select the box all the way to, and including, the Section Break (Next Page), then delete).

Below are links to some resources that go into detail about submitting your thesis and how it needs to be formatted. *Good luck!*

[Formatting & Submission Guidelines](http://thesis.extension.harvard.edu/formatting-submission)

[Margins and Pagination](http://thesis.extension.harvard.edu/margins-and-pagination)

[Discipline-specific style guide](http://thesis.extension.harvard.edu/mla-apa-or-chicago-manual-style)

**More help for this template**

[Contents](http://clients.wordwizardsinc.com/harvard/index.html)

[Introduction](http://clients.wordwizardsinc.com/harvard/introduction.html)

[Your styles environment](http://clients.wordwizardsinc.com/harvard/setting-up.html)

[Styles basics](http://clients.wordwizardsinc.com/harvard/styles.html)

[How they work](http://clients.wordwizardsinc.com/harvard/styles.html#howTheyWork)

[The Style pane](http://clients.wordwizardsinc.com/harvard/styles.html#stylePane)

[Quick Style Gallery](http://clients.wordwizardsinc.com/harvard/styles.html#quickStyleGallery)

Paragraph buttons

[Template styles](http://clients.wordwizardsinc.com/harvard/styles.html#templateStyles)

[Starting a document: Template overview](http://www.clients.wordwizardsinc.com/harvard/the-template.html)

[Notes and comments in the template](http://clients.wordwizardsinc.com/harvard/comments.html)

[Hiding comments](http://clients.wordwizardsinc.com/harvard/comments.html#hiding)

[Deleting comments](http://clients.wordwizardsinc.com/harvard/comments.html#deleting)

[Adding a new chapter](http://clients.wordwizardsinc.com.harvard/adding-chapter.html)

[Block quotes](http://clients.wordwizardsinc.com/harvard/block-quotes.html)

[Generating a Table of Contents](http://clients.wordwizardsinc.com/harvard/create-toc.html)

[Printing](http://clients.wordwizardsinc.com/harvard/printing.html)

[Printing without comments](http://clients.wordwizardsinc.com/harvard/printing.html#withoutComments)

# 

# Cloud Quantum Computer Random Number Generation for Cryptographic Purposes

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

To insert Abstract text here, select this text and then either type the text you wish to use or paste text from another document, being sure to keep the text only and not the formatting from the previous document. To keep text only, choose Paste, and then from the drop-down box that will appear, choose the Keep Text Only option on the right, with the icon of a clipboard and the letter A.

## 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 and holds 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, and fellows at Harvard whose teachings have become the foundation of this this thesis.

Table of Contents

Author’s Biographical Sketch iv

Dedication v

Acknowledgments vi

List of Tables ix

List of Figures x

Chapter I. Introduction 1

Heading 1.1, The Random Number Generation Problem in Cryptography 1

Heading 1.2, Quantum Solutions to the RNG Problem 2

Heading 1.3, Purpose of this Study 3

Chapter II. Quantum Computing 4

Heading 2.1, Superposition 4

Heading 2.2, The Hadamard Gate 5

Chapter III. Statistical Testing 6

Chapter IV. Baseline Studies 9

Heading 4.1, IBM QX4 Tenerife 9

Heading 4.2, IBM 20Q Tokyo 10

Chapter V. Experimentation 12

Heading 5.1, Sample Size 12

Heading 5.2, Minimum Input Size of Each Test 12

Heading 5.3, Testing Parameters 17

Chapter VI. Specific Testing Environment 19

Chapter VII. Testing on IBMQ Manila 20

Heading 3, SectionTitle/A-Head 21

Chapter VIII. Testing on the Righetti Quantum Computer 22

Chapter IX. Findings and Discussion 24

Chapter X. Conclusion 26

Appendix 1. Glossary 27

References 29

## List of Tables

[Table 1. NIST Statistical Tests for Randomness. 7](#_Toc94311822)

[Table 2. Maurer’s Universal Statistical Test Variables. 13](#_Toc94311823)

[Table 3. Overall Test Input Size Requirements. 14](#_Toc94311824)

[Table 4. Specific Testing Enviornments. 16](#_Toc94311825)

[Table 1. Table Title. 23](#_Toc94311826)

[Table 2. Table Title 26](#_Toc94311827)

## List of Figures

[Figure 1. Figure Title. 2](#_Toc500510491)

[Figure 2. Figure Title 2](#_Toc500510492)

## 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 create a clear advantage over classical computers in this case (Deshpande et al., 2020).

### Heading 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. A random bit sequence has 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.

TODO INSERT SEED GUESSING EXAMPLE

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. Thus, PRNGs require inputs called seeds which adds unpredictability. The seed itself must be random and unpredictable. Therefore, PRNGs are often seeded with a HRNG in a process called entropy input (Baker and Kelsey, 2015). Since they must be seeded with a random number, it seems to defeat the purpose of using PRNGs since one needs a random number to begin the process. PRNGs have better statistical properties for randomness and produce random numbers faster than pure HRNGs. Therefore, HRNGs are typically used in conjunction with PRNGs with HRNGs providing the seed.

TODO EXAMPLE OF PRNG

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). One can also subvert a HRNG by inducing signals from its supposedly random source, for example shining in a light in our previously mentioned HRNG camera sensor. Other methods of HRNG come from weakly random sources such as keyboard delays, mouse movement, or disk drive timing information. These methods need to be run through a randomness extractor to pass for use in cryptographic standards (Trevisan & Vadhan, 2000). The problem of producing a HRNG that cannot be controlled, calculated, or predicted remains.



Figure 1. LavaRand Entropy Generator at Cloudflare.

Wall of lava lamps recorded for entropy data used in encryption at website security company, Cloudflare. Photo licensed to Andrew Pham © David Edelman

### Heading 1.2, Quantum Solutions to the RNG Problem

Quantum processes have theoretical non-deterministic fundamental unpredictability possibly making them the ultimate iteration of HRNGs and a potential solution to the RNG problem. Multiple measurements made on quantum processes in identical superimposed states will not always give the same result (Ma et al., 2016). The physics behind quantum systems dictates that they are inherently random and unpredictable.

### Heading 1.3, Purpose of this Study

Many companies sell QRNG hardware that utilizes quantum phenomena; however, secure QRNG hardware is not widespread for the average consumer. Those hoping to take advantage of QRNG for encryption typically need to purchase expensive hardware 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 quantum devices product cryptographically strong random bits. Though they theoretically should, other factors such as nose and measurement error of the quantum states interfere with the random data. 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 subjecting the random data to statistical testing.

## Chapter II. Quantum Computing

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.

### Heading 2.1, 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 Righetti, the two manufacturers of quantum devices utilized 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 1. IBM Dilution Refrigerator.

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

### Heading 2.2, 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. 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.

A picture containing schematic

Description automatically generated

Figure 1. Hadamard Gate Visualized.

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

## Chapter III. Statistical Testing

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 has recognized expertise in the field of cryptography and publishes a standard for testing random number generators along with a recommendation of cryptographically secure random number generators for use. These papers are in the public domain and the process, source code, and standards are transparent for public scrutiny.

The NIST Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications (STS) 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, the null hypothesis is that “the sequence tested is random”. Small P-values indicate a significant result telling us that the null hypothesis is wrong, that the sequences is not random.

Table 1. NIST Statistical Tests for Randomness.

|  |  |  |
| --- | --- | --- |
| Test Number | Test | 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. |
| 2 | Test for Frequency within a Block | To determine whether the frequency of ones in an M-bit block is approximately M/2. |
| 3 | Runs Test | To determine whether the oscillation between substrings is too fast or too slow. |
| 4 | Test for the Longest Run of Ones in a Block | 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. |
| 5 | Random Binary Matrix Rank Test | To check for linear dependence among fixed length substrings of the original sequence. |
| 6 | Discrete Fourier Transform (Spectral) Test | 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. |
| 7 | Non-overlapping Template Matching | To reject sequences that exhibit too many occurrences of a given non-periodic (aperiodic) pattern. |
| 8 | Overlapping Template Matching Test | To reject sequences that show deviations from the expected number of runs of ones of a given length. |
| 9 | Maurer's Universal Statistical Test | To detect whether the sequence can be significantly compressed without loss of information. An overly compressible sequence is non-random. |
| 10 | Linear Complexity Test | To determine whether the sequence is complex enough to be considered random. |
| 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. |
| 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. |
| 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. |
| 14 | Random Excursions Test | To determine if the number of visits to a particular state within a cycle deviates from what one would expect for a random sequence. |
| 15 | Random Excursions Variant Test | The purpose of this test is to detect deviations from the expected number of visits to various states in the random walk. |

Purpose of the 15 tests in the NIST STS.

## Chapter IV. 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.

### Heading 4.1, IBM QX4 Tenerife

One of the first studies to evaluate IBM computers for cryptographic random number generation was done by Abdullah Ash- Saki, Mahabubul Alam, and Swaroop Ghosh in “True Random Number Generator using Superconducting Qubits”. In their implementation, they use IBM’s quantum computer QX4, also known as ibmq\_tenerife, and noticed that the ratio of 0s and 1s is severely affected by noise. The frequency test shows a deviation of 35% from the ideal ratio. In an attempt to fix the noise 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 6. 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 Abdullah et al. using Tenerife as the QRNG.

Error data from the Abdullah 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.

### Heading 4.2, 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, considered a leap in quantum computing technology with twice the quantum volume as 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 that passing the tests required both the von Neumann and Samuelson randomness extractors, though the effectiveness of this method is unclear. The primary limitation of this study was a limited sample size which we can overcome with quantum computers realizing more qubits for computation and being able to generate more random numbers per cycle.

Table 6. 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 Abdullah et al. using Tokyo as the QRNG.

## Chapter V. Experimentation

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 random number 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 two experiments, first closely replicating baseline studies on IBM quantum computers available to researchers and second, on Righetti quantum computers available on Amazon Web Services available as a commercial product.

### Heading 5.1, 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 must pass each test in the suite with an approximately 96% pass rate for a sample size of 100 binary sequences (Bassham et al., 2010).

## Chapter VI. Minimum Input Size of Each Test

Each test in SP 800-22 dictates a minimum number of bits recommended 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, understanding the minimum input requirement will reveal viable tests for our sample size.

### Heading 6.1, 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

### Heading 6.2, 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 2. 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.

### Heading 6.3, 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 2. Binary Matrix Rank n Calculation

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

### Heading 6.4, Discrete Fourier Transform

The NIST recommends the Discrete Fourier Transform (Spectral) Test be performed with a minimum of input size of 1000 bits.

### Heading 6.5, 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 = floor(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.

### Heading 6.6, Overlapping Template

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

### Heading 6.7, Maurer’s Universal Statistical

Maurer’s Universal Statistical Test requires a long 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 2. Maurer’s Universal Statistical Test Variables.

|  |  |  |
| --- | --- | --- |
| n | L | Q = 10 \* 2 ^ L |
| ≥ 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

### Heading 6.8, Linear Complexity

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

### Heading 6.9, Serial

The serial test requires that m < floor(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 < floor(log2 (n)) – 2  2 < floor(log2 (n)) – 2  n > 16, solving for n |

Figure 2. Linear Complexity n Calculation

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

### Heading 6.10, Approximate Entropy

The test requires that m < floor(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 < floor(log2 (n)) – 5  2 < floor(log2 (n)) – 5  n > 128, solving for n |

Figure 2. Approximate Entropy n Calculation

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

### Heading 6.11, Cumulative Sums

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

### Heading 6.12, 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.

### Heading 6.13, 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 limited the scope of testing in other studies examined in our baseline testing. In our tests we are going to test the 9 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 3. 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 generatedFigure 2. NIST STS Test Configuration

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

## Chapter VII. 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.

### Heading 7.1, Frequency Test 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 = floor(n/M)  N = floor(10,000/101) = 99  Which satisfies N < 100 |

Figure 2. Frequency Test Within a Block m Calculation

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

### Heading 7.2, 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.

### Heading 7.3, Serial

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

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

Figure 2. Serial m Calculation

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

### Heading 7.4, 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 < floor(log2 n) – 5.

|  |
| --- |
| Choose m and n such that m < floor(log2 n) – 5  m < floor(log2 10000) – 5  m < 8  We will use m = 7 |

Figure 2. Approximate Entropy m Calculation

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

### Heading 7.5, Overall Testing Parameters

Table 3. 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 2. NIST STS Parameter Configuration

Terminal configuration of parameters in our experiment.

## Chapter VI. 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 4. 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 5. 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 VII. 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 7. 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.

### Heading 7.1, 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.

000001001100000000110111011010011110111000110101110110111000110010001000000110101001010110000110011001010100011101000110011011100011010111000110111100001010001000001000010110110111100101011111011000

Figure 2. 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 2. ipynb code used to generate random numbers

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

Table 8. 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 below 96%. The full table is included as an appendix.

### Heading 7.2, 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 that the null hypothesis is wrong, that the sequences is not random

* 12/100 sequences passed the Frequency Test. 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 2. 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. Each column is a P-value of width 0.10, meaning C1 represents P-values 0.00-0.19 and C2 represents 0.2-0.29 etc.

* 85/100 sequences passed the Block Frequency Test. In our test specifically we test the proportion of 1s within 101-bit blocks.

Figure 2. 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. Each column is a P-value of width 0.10, meaning C1 represents P-values 0.00-0.19 and C2 represents 0.2-0.29 etc.

* 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.

Table 8. 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 |

Table description style, legend, brief citation etc.

## Chapter VIII. Testing on the Righetti Quantum Computer

Righetti is much better for this because there is no limit, its publicly available, and integrated into AWS for easy integration as a quantum rng as a cloud service for encryption, ready for commercial use by anyone. TODO UNLIKE IBM

The Rigetti Quantum Computer charges $0.30 per task and $0.00035 per shot. A shot is a single execution of the quantum circuit. We pay to execute a task, our quantum circuit, then for repetitions of that task or shots. Since the task is only factored in once, task cost is negligible compared to the final cost so we will exclude it from our cost calculations. The Rigetti Quantum Computer currently has 32 qubits, meaning we can generate 32 random bits at a time. To calculate our cost per random bit generated we divide the shot cost ($0.00035) by 32, giving us a cost of $0.0000109375 per bit. Generating 1,000,000 bits would cost $10.94. If we wanted to run every test in the test suite for the minimum significance level recommended (0.01), we would need 100 sequences of 1,000,000 bits, which would cost $1,093.75

As mentioned, we will run the 9 tests with the lowest bitstream length requirements. We’ve generated 1,000,000 bits by running 31,250 shots of the circuit. Those 1,000,000 bits are divided into 100 sequences of length 10,000 bits. We will use this sample size for the following tests:

Table 8. Manila NIST STS Results.

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

Table description style, legend, brief citation etc.

## Chapter IX. Findings and Discussion

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. TODO MOST FAILED AND WHY

Table 9. IBM Quantum Computer RNG STS Comparisons.

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

Table description style, legend, brief citation etc.

## Chapter X. Conclusion

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.

## 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.

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.

*Input Based RNG Attack:* modify the input of the RNG to put it in a known state allowing the prediction of future output. This can be done by seeding a PRNG or feeding signal to a HRNG.

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

*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.

*SDK:* Software Development Kit, a collection of tools in a package that facilities the creation of applications.

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

*State Compromise Extension RNG Attack*: when the internal secret state of the RNG is known and can be used to predict future output.

## References

Ash-Saki, A., Alam, M., & Ghosh, S. (2019). True Random Number Generator using Superconducting Qubits. *2019 Device Research Conference (DRC)*. <https://doi.org/10.1109/drc46940.2019.9046456>

Barker, E. (2020a). Guideline for using cryptographic standards in the federal government: *NIST Special Publication*. <https://doi.org/10.6028/nist.sp.800-175br1>

Barker, E. (2020b). Recommendation for key management: *NIST Special Publication*. <https://doi.org/10.6028/nist.sp.800-57pt1r5>

Barker, E. B., & Kelsey, J. M. (2015). Recommendation for Random Number Generation Using Deterministic Random Bit Generators. *NIST Special Publication*. <https://doi.org/10.6028/nist.sp.800-90ar1>

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.* <https://doi.org/10.6028/nist.sp.800-22r1a>

Brylinski, R. K., & Chen, G. (2019). *Mathematics of Quantum Computation (Computational Mathematics)* (1st ed.). Chapman and Hall/CRC.

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. <https://doi.org/10.1007/978-981-15-7062-9_41>

Gerlach, W., & Stern, O. (1922). Der Experimentelle Nachweis der Richtungsquantelung im Magnetfeld. Zeitschrift fur Physik, 9(1), 349–352. https://doi.org/10.1007/bf01326983

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. <https://doi.org/10.1103/physrevapplied.3.054004>

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). <https://doi.org/10.1038/npjqi.2016.21>

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.

Sýs, M., & Říha, Z. (2014). Faster Randomness Testing with the NIST Statistical Test Suite. *Security, Privacy, and Applied Cryptography Engineering*, 272–284. <https://doi.org/10.1007/978-3-319-12060-7_18>

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 samplable distributions. *Proceedings 41st Annual Symposium on Foundations of Computer Science*. <https://doi.org/10.1109/sfcs.2000.892063>