CLOUD QUANTUM COMPUTER

RANDOM NUMBER GENERATION

FOR CRYPTOGRAPHIC PURPOSES

by

Andrew Pham

B.S. Computer Science (California State University, Fullerton) 2019

A thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Liberal Arts

in Extension Studies

Field of Software Engineering

Cambridge, MA

2021

HARVARD UNIVERSITY

ABSTRACT

CLOUD QUANTUM COMPUTER

RANDOM NUMBER GENERATION

FOR CRYPTOGRAPHIC PURPOSES

By Andrew Pham

This paper presents a novel application of quantum computer random number generation using the Rigetti Quantum Computer. Using the NIST Statistical Test Suite the random sequences passed 2/10 tests run. These results are not in agreement with theoretical predictions and show worse performance than previous efforts by Abdullah Ash-Saki, et al. The work presented here has implications for future studies of cryptography and may one day help solve the problem of true random number generation.

Tags: Random Number Generation, Cryptography, Quantum Computers

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

List of Figures ii

List of Tables iii

Definition of Terms iv

Introduction 1

Chapter I: Random Number Generation in Cryptography 2

Statement of Problem 3

Purpose of Study 3

Chapter II: Cloud Quantum RNG Methodology 12

Rigetti Quantum Computer 13

Hadamard Initialization 21

Data 21

Chapter III: Statistical Testing 40

NIST 41

Tools and Testing Environment 41

Sample Size and Parameters 41

Test Results 43

Chapter IV: Findings and Discussion 60

Summary of Findings 63

Possible Causes of Failure 41

Practicality 71

Future Recommendations 71

Glossary 73

Bibliography 75

Appendix A: Raw FTS Output 77

Appendix B: Truncated Righetti Output 78

Appendix C: asdf 79

List of Figures

List of Tables

Definition of Terms

*Bitstream*

*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  and  to 

The Hadamard Matrix: 

*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.

*Seeding*

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

*Shot:*

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

**Chapter 1: 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.

**1.1 Problem Statement: The Random Number Generation Problem in Cryptography**

Cryptographic techniques used in computers today such as RSA rely on a key seeded by random number generators with cryptographic strength depending on the randomness of these numbers. Sensor data from hardware random number generators in combination with randomizing algorithms has been the preferred method of generating random numbers for cryptography. 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.

TODO MAYBE AN EXAMPLE OF RSA ENCRYPTION WHERE SEED IS KNOWN

Therefore, the foundation of modern cryptographic security relies on random number generation. Bad RNGs can and have been the demise of many encrypted systems. Currently, the process of generating random numbers has been divided into two main approaches, PRNGs and HRNGs. PRNGs utilize an algorithm to generate “random” numbers, or simply a series of mathematical formulas. However, by nature of an algorithm, PRNGs are deterministic in nature 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. Many times, PRNGs have better statistical properties for randomness and produce random 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 thermal 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 source (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.

**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. This means that theoretically QRNGs are impervious to known or forced state attacks such as state compromise extension attacks and input-based attacks because the state cannot be used to predict or dictate future behavior. Because of this property, QRNGs have been said to be the final iteration of random number generators.

**1.3 Purpose of this Study**

Today, 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 Righetti machine.

The purpose of this study is to determine whether randomly generated numbers from this machine produces cryptographically strong random bits by subjecting the random data to statistical testing.

**Chapter 2: 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.

**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. 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. 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 though other quantum computers may make use of superposition defined by other methods such as photon polarization or trapped ions.

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

**Chapter 3: Experimental Process Overview**

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 attempt to replicate baseline studies on IBM quantum computers available to researchers and apply this methodology to Righetti quantum computers available on Amazon Web Services as a commercial product.

**Chapter 4: 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 is a US government agency, has strong documentation, and provides tests specifically for cryptographic applications.

Table: NIST Statistical Tests for Randomness

|  |  |  |
| --- | --- | --- |
| Test Number | Test | Purpose |
| 1 | Frequency (Monoids) Test | To determine whether that 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 is an M-bit block is approximately M/2. |
| 3 | Runs Test | To determine whether the oscillation between such 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 |  |
| 15 | Random Excursions Variant Test |  |

**Chapter 5: Testing Parameters**

**5.1 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 the entire suite with an approximately 96% pass rate.

**5.2 Calculating 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. Understanding the minimum input requirement will reveal viable tests for our sample size.

**5.2.1 Frequency 5.2.2 Block Frequency 5.2.3 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.

**5.2.4 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.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Minimum n | M | | 128 | 8 | | 6272 | 128 | | 750,000 | 100,000 |   The minimum input length accommodated  by the NIST being 128 for smallest M preset value, 8. |

**5.2.5 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 has 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. |

**5.2.6 Discrete Fourier Transform**

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

**5.2.7 Non-overlapping Template**

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 requirement that N is less than or equal to 100, M is greater than 0.01, and m is no less than 9 for meaningful results. For the default setting N=8 (number of independent blocks), n=8M (M is block length). 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.

|  |
| --- |
| 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) |

**5.2.8 Overlapping Template**

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

**5.2.9 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: 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 |

**5.2.10 Linear Complexity**

The linear complexity test requires n greater than 1,000,000.

**5.2.11 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, 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 |

**5.2.12 Approximate Entropy**

The serial test requires that m < floor(log2 (n)) – 5 where m is the length of bits in each block. We will choose m = 2, 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 |

**5.2.13 Cumulative Sums**

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

**5.2.14 Random Excursions 5.2.15 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.

Table: 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 |

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 500,000 bits per sequence, and all 15 tests can be run with 1,000,000 bits per sequence. These minimums explain the scope of testing in other studies duplicated 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 on generating more bits; if not, there would be no need to continue with more tests because we can conclude the sequences are not random enough for encryption.

**Chapter 6: Specific 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: Testing Environment

|  |  |
| --- | --- |
| 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 |
| Processor | AMD Ryzen 5 3600 |

**Chapter 7: Baseline Studies**

IBM has made quantum computers available for research scholars…

**7.1 IBM QX4**

Similar testing has been 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 failed 8 of the 15 NIST randomness tests at best.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

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.

**7.2 IBM 20Q Tokyo**

Quantum Random Number Generation with the Superconducting Quantum Computer IBM 20Q Tokyo by Kentaro Tamura and Yutaka Shikano (Tamura et al., 2020): Utilizes the IBM 20Q Tokyo for QRNG with Hadamard initialization then runs the NIST tests to analyze the output, an identical approach to this thesis proposal. Using the 20 qubit quantum computer they 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.

|  |  |
| --- | --- |
| Test | Raw Sample |
| Frequency | 0.0000 |
| Frequency Block | 0.0000 |
| Runs | 0.0000 |
| Runs Block | 0.0000 |
| Matrix Rank | 0.5285 |
| DFT | 1.0000 |

**Chapter 8: Baseline Reproduction 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 this baseline on a similar modern IBM machine, Manila (ibmq\_manila).

Figure: 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 |

The IBM computer is free but ha a max of 5 qubits requiring 200,000 shots to achieve 1,000,000 random bits. The max shot size allowed is 20,000 shots. We will run 20 batches of 20,000 shots giving us a sample size of 2,000,000 bits which we can divide into 200 sequences of 10,000 bits.

Testing Parameters Full Calculations

|  |  |
| --- | --- |
| Test | Parameters |
| Length (n) | 10,000 |
| Bitstreams | 200 |
| Frequency Test within a Block | n ≥ MN  The block size M should be selected such that M ≥ 20, M > .01n and N < 100.  1. M ≥ 20  2. M > .01n = M > 100  Therefore we choose M = 101  N = floor(n/M) = floor(10,000/101) = 99  Which satisfies N < 100 |
| Non-overlapping Template Matching Test | m = 9 or 10 as recommended by NIST |
| Serial Test | Choose m and n such that  m < floor(log2 n) - 2.  m < floor (log2 10,000) – 2  m < 11  We will use m = 10 |
| Approximate Entropy Test | Choose m and n such that  m < floor(log2 n) – 5  m < floor(log2 10000) – 5  m < 8  We will use m = 7 |

Testing Parameters Quick Reference

|  |  |
| --- | --- |
| 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 |

Testing Results Quick Reference

|  |  |
| --- | --- |
| Test | Parameters |
| 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: Baseline Comparisons

**Chapter 9: Novel Commercial Implementation on AWS**

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 before 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: TODO. TODO TESTS THAT REQUIRE SAMPLE SIZE MODIFICATION

Testing Parameters Full Calculations

|  |  |
| --- | --- |
| Test | Parameters |
| Frequency Test within a Block | 0 |
| Non-overlapping Template Matching Test | 0 |
| Serial Test | 0 |
| Approximate Entropy Test | m < floor(log2 n) – 5 |

Testing Parameters Quick Reference

|  |  |
| --- | --- |
| Test | Parameters |
| Frequency Test within a Block | 0 |
| Non-overlapping Template Matching Test | 0 |
| Serial Test | 0 |
| Approximate Entropy Test | m < floor(log2 n) – 5 |

The tests in 800-22 are formulated to test the null hypothesis, “the sequence is random”. The alternate hypothesis is that “the sequence is not random”. If every test accepts the null hypothesis, cloud quantum computers can generate random numbers for cryptographic applications.

**Chapter 10: Findings and Discussion**

The minimum pass rate for each statistical test included here is approximately 96.0150% for a sample size of 100 binary sequences.

Results Table

|  |  |
| --- | --- |
| Test | Proportion of Passing Sequences |
| Frequency Test | 49%\* |
| Frequency Test within a Block | 89%\* |
| Runs Test | 96.25% |
| Test for the Longest Run of Ones in a Block | 95%\* |
| Binary Matrix Rank Test |  |
| Discrete Fourier Transform Test | 92%\* |
| Non-overlapping Template Matching Test | In the 148 templates tested, 108 had a passing proportion of passing sequences (40 failing) |
| Overlapping Template Matching Test |  |
| Maurer’s Universal Statistical Test | x |
| Linear Complexity Test | x |
| Serial Test | 99% |
| Approximate Entropy Test | 94%\* |
| Cumulative Sums Test | 53%\* |
| Random Excursions Test | x |
| Random Excursions Variant Test | x |

\*Failure, proportion of passing sequences under 96.0150%

**CHAPTER DISCUSSION**

Description of Findings

Test name, what does it test for, what operations does the test carry out, what parameters did we use, did it pass or fail, by how much

Frequency test, our findings were similar to…. True Random Number Generator using Superconducting Qubits by Abdullah Ash-Saki, Mahabubul Alam, and Swaroop Ghosh (Ash-Saki et al., 2019). Generated random numbers using superconducting qubits from IBM. Ran statistical testing on the samples, which failed, then gave detailed analysis. Their experiments showed a deviation of 35% from ideal 1/0 ratio due to noise. They also propose techniques to improve the 1/0 ratio and pass the NIST statistical tests.

Further research

Findings matched…

The absence of values on exactly 50% is due to the noise and de-coherence effect which is mentioned in the limitations section of this paper TODO LOOK AT IMPLICATION OF QUANTUM LIMITATIONS SECTION

Summary

Bibliography

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>

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>

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>

As theorized by… Quantum Random Number Generation by Ma, Xiongfeng, Xiao yuan, Zhu Cao, Bing Qu, and Zhen Zhang (Ma et al., 2016): Details general approaches to creating QRNGs incorporating aspects such as trustworthiness and random number generation speed.