Proposal for a Thesis

in the Field of Software Engineering

in Partial Fulfillment of the Requirements for the

Master of Liberal Arts Degre

Harvard University

Extension School

May 5th, 2021

Andrew Pham

Cambridge, MA

I.

Tentative Title

“Cloud Quantum Computer Random Number Generation for Cryptographic Purposes”

II.

Abstract

Cryptographic techniques used in computers today rely on a key seeded by random number generators with cryptographic strength depending on the randomness of these numbers. Algorithmic methods have been used to generate pseudorandom numbers but unpredictable sensor data from hardware random number generators has been the preferred method of generating random numbers for cryptography. Quantum processes have theoretical non-deterministic fundamental unpredictability possibly making them the ultimate iteration of hardware random number generators. This paper presents a method for utilizing cloud quantum computers newly made available to the public to generate random numbers. Random sequences generated from these computers will be run through a statistical test suite and show that this method [produces / does not produce] cryptographically strong random bits.

III.

Definition of Terms

*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 actually 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 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.

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

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

IV.

Background and Significance

Random numbers have a wide variety of applications from simulations to sampling but one of the most impactful and every day of uses is encryption. The foundation of modern cryptographic security relies on random number generation. In cryptography, a key is used to encrypt and decrypt information (E. Barker, 2020a). It works analogously to the combination of a lock. On a computer, keys look like a string of random characters. Guessing a key will allow the attacker to decrypt information. Keys must be random to be secure because if the attacker has information on which keys are more or less likely, it makes guessing the key easier (E. Barker, 2020b). Bad RNGs can and have been the demise of many encrypted systems.

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. This is why 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 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).

An ideal HRNG cannot be controlled, calculated, or predicted. The only processes known to be fundamentally random are those of quantum measurements and observation. The generation of genuine randomness is generally considered impossible with only classical means. Multiple measurements made on quantum processes in identical 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. Today, many companies sell QRNG hardware that utilizes quantum phenomena; however, secure QRNG hardware is not widespread for the average consumer.

The release of quantum computers available in the cloud could bring this technology to everyday users. To test if these new methods are suitable for cryptographic purposes, we will run them through the NIST Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. If the numbers generated pass the tests in the suite, we deem the technique ready for widespread application and allow for the adoption of quantum RNG on a universal basis making random number generation one of the first real world applications of quantum computers.

Other attempts at creating cryptographically secure random numbers with quantum computers have failed due to noise skewing the output, or lack of sample size. See prior works in section IX for more details. Now with the release of D-Wave's 5000 qubit quantum computer, we hope to overcome these obstacles.

V.

Prior Works

Implications of Quantum Superposition in Cryptography: A True Random Number Generation Algorithm by Dhananjay S. Deshpande, Aman Kumar Nirala, and Ayodeji Olalekan Salau (Deshpande et al., 2020, p. 421): Used QUISKET to create a QRNG on IBM’s cloud quantum computer, IBM-Q experience. Paper also explains the physical and mathematical aspects of superposition and RNG. The paper shows in theory that RNG for cryptography is possible on quantum computers. The paper utilizes the same algorithm as proposed in this thesis for RNG by applying the Hadamard gate on a qubit. However, the paper does not run statistical testing on the output and the IBM-Q computer was a 15 qubit system, limiting their output.

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.

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.

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.

VI.

Research Methods

Random numbers will be created inside of a quantum computer by measuring a qubit in superposition. The quantum computer will be from Amazon Braket, a platform from Amazon Web Services providing on demand cloud computing. The specific computer for the implementation will be D-Wave's Advantage system. 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.

The random numbers will be generated inside of the quantum computer by measuring qubits after Hadamard initialization. Quantum computers encode information with qubits, basic units of quantum information. Using qubits we can utilize quantum properties such as entanglement and superposition. We are most interested in the property of superposition which puts the qubits into a probabilistic state. To put a qubit into superposition we operate on them with a Hadamard gate. This is commonly called Hadamard initialization. The Hadamard gate puts the qubit into equal superposition between the 0 state and the 1 state (Brylinski & Chen, 2019). In other words, when measured the qubit has an equal chance of collapsing into the 0 state and the 1 state.

The Amazon Braket SDK has built in Hadamard circuits that we can apply to each numbered qubit. Once each qubit is in superposition, we measure the state, collapsing the state into either 0 or 1. These measurements will be stored in sequences for our testing.

Each sequence will be stored as an ASCII sequence of 0s and 1s. We will store 1,000 sequences of 1,000,000 random bits to meet our criteria for significance level and testing requirements described later. Specifically, these sequences will be stored in the Amazon Bucket service once the task is completed by the quantum computer.

The sequences will be evaluated via the NIST Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. It is the current standard for statistical testing of new CSPRNG designs. The test suite includes 15 tests:

|  |  |
| --- | --- |
| Test | Purpose |
| Frequency (Monobits) 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. |
| Test for Frequency within a Block | To determine whether the frequency of ones is an M-bit block is approximately M/2. |
| Runs Test | To determine whether the oscillation between such substrings is too fast or too slow. |
| 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. |
| Random Binary Matrix Rank Test | To check for linear dependence among fixed length substrings of the original sequence. |
| 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. |
| Non-overlapping Template Matching | To reject sequences that exhibit too many occurrences of a given non-periodic (aperiodic) pattern. |
| Overlapping Template Matching Test | To reject sequences that show deviations from the expected number of runs of ones of a given length. |
| Maurer's Universal Statistical Test | To detect whether or not the sequence can be significantly compressed without loss of information. An overly compressible sequence is considered to be non-random. |
| Lempel-Ziv Complexity Test | To determine how far the tested sequence can be compressed. The sequence is considered to be non-random if it can be significantly compressed |
| Linear Complexity Test | To determine whether or not the sequence is complex enough to be considered random. |
| 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. |
| 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. |
| 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. |
| Random Excursions Test | To determine if the number of visits to a state within a random walk exceeds what one would expect for a random sequence. |
| Random Excursions Variant Test | To detect deviations from the expected number of occurrences of various states in the random walk. |

The details and descriptions of these tests can be referenced in NIST Special Publication 800-22 Revision 1a, hereby referred to as SP 800-22.

The significance level for these tests will be set to 0.001, 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 1000 sequences in our sample size. This means each test will be run 1000 times on 1000 different sequences. In order for our RNG to be suitable for cryptographic applications, the random sequences must pass the entire suite with an approximately 96% pass rate.

Each test in SP 800-22 dictates a minimum number of bits required for evaluation. The overlapping template matching test requires the most bits, 1,000,000 at minimum. Since each sequence should have uniform length, each of our sequences will be 1,000,000 random bits.

With 5760 qubits available in the D-Wave advantage system we can generate the same number of random numbers at a time. Each qubit will be Hadamard initialized and measured. The program will be re-run on the quantum computer by dictating the number of “shots” required. To create 100 sequences of 1,000,000 random bits each, we need to run 17,362 shots, recording the binary state of each shot.

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 are able to generate random numbers for cryptographic applications.

If any test rejects the null hypothesis, the thesis will investigate the causes. If the sequences fail the tests, it will likely be from errors in the quantum computer states and measurements. These error boundaries are published by the quantum computer providers and an analysis of how the errors impact the test will be included.

VII.

Experimental Tools

NIST Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications: A testing suite created by the National Institute of Standards and Technology for determining whether or not a random number generator is suitable for cryptographic applications (Bassham et al., 2010).

Faster Randomness Testing: A project from the Faculty of Informatics at Masaryk University optimizing the NIST statistical tests for randomness. Often the NIST test suite can run for hours; this tool can reduce the processing time to minutes (Sýs & Říha, 2014).

Amazon Braket: A cloud quantum computing service from Amazon Web Services offering compute time on multiple quantum computers. This thesis proposes using the D-Wave Advantage System 1.1 quantum computer for its large number of available qubits, 5760. Since the process of generating a random number only requires measuring one qubit as described above, we want to use the system with the most qubits possible.

Python and Braket SDK: The random number generation will be implemented in the python programming language. Amazon Web Services has created a Braket SDK for use with their machines which can be referenced on GitHub at aws/amazon-braket-sdk-python. Quantum gates and circuits are defined in the SDK’s braket.circuits class.

VIII.

Expected Results

Theoretically, the numbers extracted from Hadamard initialization and measurement should be random from the laws of quantum mechanics. However, there are errors in state initialization, gates, and measurements that possibly could lead to a biased random output. The input fed into the tests will be purely from the output of the quantum computer. It will not be fed through a randomness extractor first.

The research will be focused on one method of random number generation, Hadamard initialization. There are other methods for generating random numbers using quantum effects. Many of these techniques do not involve quantum computers at all preferring to measure only quantum effects such as electromagnetic fields in a vacuum. Hadamard initialization is likely not the most efficient way of quantum random number generation. However, many of the other proposed methods for quantum random number generation require randomness extraction. For example, the Australian National University Quantum Random Numbers Generator utilizes the cryptographic hash technique AES-128 as a randomness extractor (Haw et al., 2015).

The research will use one cloud quantum computer, D-Wave Advantage System 1.1. Hadamard initialization was chosen as the simplest way to produce random numbers in a quantum computer. D-Wave Advantage System 1.1 was chosen for its high qubit count, ease of access, ease of programming, and cost efficiency. There are multiple quantum computers available on Amazon Braket and multiple cloud quantum services beyond Amazon Braket. Other cloud quantum services that could be considered for later testing include IBM Q Experience, Microsoft Azure Quantum, and Google Quantum Computing.

IX.

Tentative Schedule

|  |  |
| --- | --- |
| Development environment setup complete on Amazon Braket  Codebase written and finalized  Quantum random data generated  Statistical testing completed | 1 month  1 month  1 month  1 month |
| Design of new experiments (if needed)  Further experiments and testing (if needed) | 1 month  1 month |
| Thesis Draft Completed  Graduation | 1 month  May 2022 |

X.

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