# Generating True Random Numbers From Vacuum Fluctuations in a Quantum System

**Love Arreborn**
lovar063@student.liu.se

**Nadim Lakrouz**
nadla777@student.liu.se

## 1 INTRODUCTION

In cryptography, there are many applications for randomly generated numbers. However, the process of producing these random numbers tends to be pseudo-random, e.g. utilizing the current states of various modules [2]. These numbers do not generate true randomness, and in order to heighten security other methods of generating these are required. Current random number generators (*RNG*) are usually implemented in code, using certain states of the host machine as a starting point before running a predetermined algorithm [2]. This pseudo-random number generation (*PRNG*) comes with the drawback that the result is always deterministic, provided that the initial state is known.

True random numbers, then, cannot be produced solely through code. These systems require some input that is neither replicable nor reproducible. One proposed solution for this is quantum random number generation (*QRNG*) [3]. By reading quantum fluctuations from any given source, for instance an optical signal, the inherent natural unpredictability of said source can be harnessed in order to produce a random number from a state that is nigh impossible to reproduce accurately.

In this project, we will be writing firmware for one such solution, which reads quantum variations from an optical signal. Further details about how this signal is produced will be introduced in section 2 and builds on the work of Clason [1]. This optical signal will be converted to a stream of random, raw bits via an Analog to Digital Converter (*ADC*). In turn, these random bits will be processed via Toeplitz-hashing [10] in order to process these bits into random numbers. Some processing has to be done on the microcontrollers themselves in order to ensure that the data is workable, and Toeplitz-hashing is a tried and tested method to accomplish this. These random numbers will then be output from the microcontroller to the host computer via USB. This thesis will aim to answer one key research question: How can sampled vacuum fluctuations be processed efficiently in order to output QRNG?

In producing this firmware, several key considerations have to be made in order for this system to be usable in a production environment. The vision for the end product is a simple USB-stick that can be connected to a host, and produce true random numbers from ambient quantum fluctuations. While this system could very well be implemented on physically larger hardware, thus avoiding the constraints that limited hardware introduces, Clason proposes a simpler and cheaper way to achieve QRNG [1]. In keeping with this, utilizing microcontrollers for processing the raw bits extracted further helps to keep the costs low and the solution reasonably complex. Further details regarding the microcontrollers used in our work will be outlined in section 3. However, due to this portability constraint, our implementation needs to work quickly and efficiently on resource constrained hardware. As such, our main question is broken down into two concrete research areas:

**Research area 1 (*RA1*)**: How can Toeplitz-hashing be implemented as effectively as possible on resource constrained hardware in order to process raw bits into a workable random number?

Toeplitz-hashing been optimized quite well, and previous research can be utilized for this. However, there are still considerations when implementing the firmware for the microcontroller in order to optimize the code. Our goal is to attempt several implementations in order to find the most optimal implementation with the least amount of CPU-cycles.

**Research area 2 (*RA2*)**: How can we ensure that the output of random numbers is not limited by our firmware, but rather only limited by the USB transfer speed of the microcontroller or the ADC?

There will unequivocally be a bottleneck for the processing speed. For instance, the speed at which the ADC can process the optical signal into raw bits as well as the speed that the USB output can transfer processed random number to the host computer will be limiting factors. Further details on the limitations of the ADC will be outlined in section 3. The slowest of these bottlenecks will inevitably be the limiting factor for any implementation. Our research aims to ensure that our implementation of Toeplitz-hashing does not become the limiting factor, but rather processing data fast enough to match or exceed the speed of the hardware.

Section 2 of this article will introduce the theory that allows for QRNG, and how this will be utilized in our works. Section 3 delves further into the hardware and algorithms our work will use, and section 4 will present our methodology. **More sections to follow as we finish the article**.

## 2 THEORY

A majority of the research around this topic stems from physics, with implementations of the technology frequently being published and studied by physicists. As such, a brief introduction to the concepts used in previous research as well as an introduction to previous implementations of this technology will be presented in this section.

The idea of an optical QRNG (*OQRNG*) is not a novel one. The basis of the theory is that intrinsically random properties of a quantum process. Stefanov et. al. [8] proposes using the random choice of a photon between two output signals to generate a random stream of bits, however the theory behind it can be applied to other quantum processes as well. This particular theorem has been implemented by Wayne et. al. [9] to create a quantum number generator. While this article proves the efficacy of OQRNG, it utilizes a slightly different method.

### *Shot noise quantum fluctuations*

Our work revolves around the measurement of shot noise of vacuum states rather than measuring arrival times of photons. Essentially, this is another quantum process with the same inherently random properties as described by Stefanov et. al., but instead using shot noise. As described by Niemczuk [5], shot noise is minor fluctuations in an elecritcal current, which is inherently random. Reading this property, then, gives us an intrisically random source from which to generate a random output, which in turn can be processed into a random number.

Implementations of this theory exist, however with significant drawbacks. Shen et. al. [6] presents an implementation using a fairly complex setup, in which a continuous-wave fiber laser is the optical source. They conclude that sampling the shot noise is, indeed, suitable for OQRNG. However, the implementation requires expensive and complex hardware, and the sheer size of the system prohibits it from being portable and easily reproducible in small-scale tests.

A more recent implementation of OQRNG in a smaller scale has been presented by Singh et. al. [7]. This particular implementation uses a bespoke circuit board where all components are present on a single board – e.g., this experimental setup contains an integrated ADC, post-processor, entropy controller and entropy generator. While this article cements the viability of OQRNG using shot noise (despite the article not being confirmed as peer reviewed), the bespoke nature of the circuit board makes this experiment difficult to reproduce. As our thesis will use commercially available ADCs and microcontrollers, the only bespoke component is the shot noise generator itself. Furthermore, the Toeplitz-hashing is not run on the microcontroller itself in these experiments – instead, the hashing of these raw bits is done on the receiving computer as this bespoke circuit board featured a relatively weak processor.

In summary, previous research has proven that OQRNG can generate true randomness, and more specifically, Shen et. al. [6] and Singh et. al. [7] both implement OQRNG through readings of shot noise. However, there are limitations in both of these works. Either the system that generates the shot noise is large and complex [6] or the system is built on bespoke hardware with limitations in processing power which prevents a fully integrated system [7]. Furthermore, to the best of our knowledge, most of the work in this field is from the perspective of physicists, and there appears to be little research on this subject in the domain of computer science. Our work aims to bridge this gap by using commercially available hardware (other than the bespoke shot noise generator [1]) and focuses on implementing Toeplitz-hashing directly on the microcontroller. Rather than focusing on the intricacies of quantum fluctuations, we will instead approach this problem from a computer science perspective.

## 3 BACKGROUND

Our work is a practical continuation of the work of Clason [1]. In this work, quantum shot noise originating from photodiodes was studies, and in so doing a prototype device was constructed. This device read from an optical source, outputting analog voltage from the data "seen" by the diodes. A prototype was constructed, in which an LED is read by a photodiode soldered millimeters apart. In this section, we introduce the hardware used for our implementations as well as the considerations taken in order to shift the focus from physics to computer science.

### *Optical RNG module*

The one bespoke piece of hardware used in this study is the prototype designed by Clason [1] as a part of his masters thesis. This device produces the optical shot noise which will be the source of randomness in our work implementing the digization scheme discussed in section 5.1 of this article. Moving forward in this article, we will refer to this as the OQRNG-device.

As described in Clason's work [@Clason2023}, the OQRNG-device is an electro-optical system which measures optical shot noise, generating quantum randomness. The device has an LED and a photodiode positioned a few millimeters apart, ensuring efficient light coupling. The photodiode detects light from the LED, and converts th elight into a current signal, which is sent to a transimpedance amplifier to convert it into measurable voltage. In order to minimize disruptions by other external lights, the system is enclosed in a shielded measurment box.

Whereas the exact quantum mechanisms that ensure that this system ensures randomness is better derived directly from Clasons work [1] directly, the end result as if correlates to our study is an inherently random, analog voltage current.

### *ADC converter*

This analog current isn't suitable to operate on without further processing. As mentioned in section 1, the signal needs to pass through an ADC and converted into raw bits in order for it to be useable. In his thesis, Clason [1] suggests a discrete ADC chip capable of analyzing frequencies higher than 25 MHz, as studied in his work. The Nyquist-Shannon theorem poses that – in order to accurately reconstruct the signal, the sampling rate must be at least twice the highest frequency component present in the signal, $f_s \geq 2f_{\max}$. **CITATION NEEDED**

**This section will be filled with more information as soon as the exact ADC we will be using has been selected. The exact requirements needs to be discussed with the project owner before a choice can be made!**

*Microcontroller*
*Toeplitz-hashing*
*Summary*

## 4 METHODOLOGY

With the consideration that our work revolves around optimizing Toeplitz-hashing in order to quickly process random bits into a random number, we will take an iterative approach. For our initial tests, we will use a pre-defined stream of raw bits which is loaded into memory on the microcontroller, and run several different implementations of Toeplitz-hashing to produce numbers. As we always use a pre-defined bitstream, the result will at this stage be deterministic, giving us a clear indication whether the algorithm works as intended.

However, in order to ensure the results work with varying data, we cannot limit ourselves to simply one stream of bits. The main point of the algorithm is to remove patterns in the bitstream that may lead to less randomized results. As such, we will manually produce several bitstreams to use for our tests – each with varying degrees of repeated patterns that should be eliminated by the algorithm. Furthermore, in order to see how well our implementation will work with realistic data from the OQRNG generator, we will sample streams of bits directly from the source.

### *Evaluating optimization efforts*

As discussed in section 3, the hardware used will impose a clear bound on how quickly our implementation needs to process the bits in order to match the speed of the ADC, as well as the output speed of the USB-port, both in $MB/s$. Hyncica et. al. [4] propose that measuring execution time of algorithms directly via the microcontrollers internal timers (while subtracting the interrupt overhead) provides adequate measurements. An additional advantage is that the same code can be used to measure execution speed on several different microcontrollers, rather than relying on counting CPU cycles (as the process for this may vary greatly between controllers). As we will use fixed-size bitstrings for evaluation, we can then derive the throughput of the algorithm in $MB/s$ as follows:

$$Throughput_{MB/s} = \frac{DataSize_{bits}}{ExecutionTime_{ms}} \times \frac{1}{8} \times \frac{1000}{10^6} \quad (1)$$

This measurement allows us to place the throughput of our algorithm soundly in the bounds imposed on us by the hardware.

### *Iterative approach*

Implementation of Toeplitz-hashing will begin by a naive implementation not optimized for speed, but rather for accuracy. This implementation will be executed on a separate computer in order to produce the correct random number for every provided bitstring. These will be used as our baseline for accuracy for future iterations.

This naive implementation will then be flashed to our microcontrollers, beginning with Teensy 4.1 as this is the more capable of the microcontrollers used for this experiment. Code to measure the execution speed in milliseconds will be implemented and tested before we load the naive implementation on said microcontroller. We expect that several implementations ma y be too resource intensive or have a memory complexity far greater than our cheaper, less capable microcontroller are able to handle, and as such these may not be able to be tested until a few iterations of optimization has occurred.

Each iteration will consist of incremental improvements to the algorithm. Our initial investigation has shown several avenues for improving the throughput of the algorithm, and each planned iteration will be discussed briefly. Depending on the empirical results of these iterations, there may be a need for further optimization iterations other than those listed.

**Iteration 1 - Naive implementation**: The naive implementation consists of a Toeplitz matrix $T$ and key $k$, acquired from fixed slice of the bitstream. Than matrix-vector multiplication will be performed using $T$ and $k$, thus performing Toeplitz extraction. Which finally will eliminate jitter and produce random number. Code implementation will look much more simple, using double loop to go through all values and multiply them. We expect that the naive implementation will be far away from optimized iterations and resulting in $O(n^2)$ complexity.

**Iteration 2 - Efficient data structures**: The naive implementation will not utilize efficient data structures for maximum speed, rather it will likely use fixed-size arrays which need to be iterated over for a time complexity of $O(n^2)$. Our hypothesis is that more efficient data structures may allow data to be processed in a far more efficient manner – however, this scenario poses a risk for greater memory complexity which may not be suitable for use on microcontrollers.

**Iteration 3 - Bitshifting**: We expect that Toeplitz extraction can be significantly improved with bitwise XOR operations, thus reducing overhead and memory usage. Instead of explicitly constructing the Toeplitz matrix, a right-shift operation can be used to dunamically reconstruct matrix rows. Furthermore, matrix-vector multiplication can be optimized by utilizing XOR operations to extract relevant bits more effectivly, thus eleminating unnecessary computations. Additionally, the iterative update of the extracted hash can be improved using a left-shift operations, thus allowing for continuos entropy accumulation without requiring full recomputations.

**Scenario 4 - Batching**: Finally, we consider the concept of batching larger amounts of bits for processing. Consider that an input buffer of bits is read from the ADC and stored, waiting for processing. Rather than taking 64 bits, and shifting them one by one, we can take two batches of 64 bits and multiply them directly using XOR bitshifting – eliminating the need to process these bit-by-bit. As we will have a constant stream of bits from the ADC, we theorize that this method will allow for better performance in real-time processing over reading individual bits.

**Iteration 5 - ARM Hardware instructions**: Rather than performing bitshifting operations in the code itself, certain microcontrollers come equipped with a separate processor specifically for bitshift-operations. Offloading the shifting to these processors rather than running them on the main CPU may allow faster processing of the data than performing the shifting in the code itself. However, this operation isn't natively supported by Arduino, and will potentially lead to extreme rewrites of the code which may prove too time consuming to do for two microcontrollers (as the code will not be reusable between controllers). As such, this optimization may only be done for one controller or left for future work.

Each scenario will first be executed on a single thread, and (**if time allows**) multiple threads may be executed concurrently to further optimize the data. The feasability of this highly depends on the performance of every individual implementation.

## 5 LIMITATIONS

## REFERENCES


[1] Martin Clason. 2023. *Development of a QRNG front-end for shot noise measurement: analysis of quantum shot noise originating from photodiodes*. Independent thesis Advanced level (degree of Master (Two Years)). Linköping University, Department of Electrical Engineering, Information Coding, Linköping, Sweden. Available from: 2023-12-22.

[2] R. Gennaro. 2006. Randomness in cryptography. *IEEE Security Privacy* 4, 2 (2006), 64–67. DOI: http://dx.doi.org/10.1109/MSP.2006.49

[3] Miguel Herrero-Collantes and Juan Carlos Garcia-Escartin. 2017. Quantum random number generators. *Rev. Mod. Phys.* 89, Article 015004 (Feb 2017), 48 pages. Issue 1. DOI: http://dx.doi.org/10.1103/RevModPhys.89.015004

[4] Ondrej Hyncica, Pavel Kucera, Petr Honzik, and Petr Fiedler. 2011. Performance evaluation of symmetric cryptography in embedded systems. In *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*, Vol. 1. 277–282. DOI: http://dx.doi.org/10.1109/IDAACS.2011.6072756

[5] Jakub Niemczuk. 2020. Shot noise-based quantum random number generator. In *Quantum Technologies 2020*, Eleni Diamanti, Sara Ducci, Nicolas Treps, and Shannon Whitlock (Eds.), Vol. 11347. International Society for Optics and Photonics, SPIE, France, Article 1134717, 6 pages. DOI: http://dx.doi.org/10.1117/12.2554898

[6] Yong Shen, Liang Tian, and Hongxin Zou. 2010. Practical quantum random number generator based on measuring the shot noise of vacuum states. *Phys. Rev. A* 81, Article 063814 (Jun 2010), 5 pages. Issue 6. DOI: http://dx.doi.org/10.1103/PhysRevA.81.063814

[7] Jaideep Singh, Rodrigo Piera, Yury Kurochkin, and James A. Grieve. 2024. A Compact Quantum Random Number Generator Based on Balanced Detection of Shot Noise. (2024). https://arxiv.org/abs/2409.20515

[8] André Stefanov, Nicolas Gisin, Olivier Guinnard, Laurent Guinnard, and Hugo Zbinden. 2000. Optical quantum random number generator. *Journal of Modern Optics* 47, 4 (2000), 595–598. DOI: http://dx.doi.org/10.1080/09500340008233380

[9] Michael A. Wayne, Evan R. Jeffrey, Gleb M. Akselrod, and Paul G. Kwiat. 2009. Photon arrival time quantum random number generation. *Journal of Modern Optics* 56, 4 (2009), 516–522. DOI: http://dx.doi.org/10.1080/09500340802553244

[10] Xiaoguang Zhang, You-Qi Nie, Hao Liang, and Jun Zhang. 2016. FPGA implementation of Toeplitz hashing extractor for real time post-processing of raw random numbers. In *2016 IEEE-NPSS Real Time Conference (RT)*. IEEE, USA, 1–5. DOI: http://dx.doi.org/10.1109/RTC.2016.7543094