ELE201 Semester Project

Thobias Høivik

Contents

1 Introduction

In this project we use the STM32 microcontroller along with analog sensors to gather data that we analyze for statistical and informational randomness, with the goal of evaluating how feasible such a setup is as a hardware-based true random number generator (TRNG). The data is sent over a network to a server for processing.

We will combine theory from information theory, probability, signal processing, and networking to assess both the quality of randomness and how such a system could integrate into a distributed architecture.

We will discuss:

- Theoretical background on randomness and entropy.
- Methods for sampling analog noise using microcontrollers.
- Implementation and data transfer.
- Statistical analysis and randomness testing.
- Viability discussion and possible improvements.
- Network design for scaling such systems.

2 Theory

2.1 Randomness and Entropy

Randomness can be defined both in a statistical and algorithmic sense. A sequence is considered random if it is unpredictable and lacks compressible structure.

Definition 2.1: Shannon Entropy

Given a discrete random variable X that takes values in χ with probability distribution $p : \chi \to [0,1]$, its entropy is

$$H(X) = -\sum_{x \in \chi} p(x) \log_2 p(x).$$

Entropy measures the uncertainty of a random variable. For a binary sequence, $H(X) = 1$ indicates perfectly balanced and maximally unpredictable bits.

Intuitively, high entrop should mean unpredictable bits, while low entropy should mean that the bits are structured and predictable. This sounds very good for our purposes, but if we look at a string like "010101010101…", where we alternate 0s and 1s, we get a shannon entropy of 1 which is as high as we can go. However, this string of bits very clearly has a predictable structure, which entropy alone cannot take into account.

Thus we must find more meassures of randomness. Kolmogorov complexity

(https://www.cs.cmu.edu/ venkatg/teaching/15252-sp20/notes/Kolmogorov-Complexity.pdf),

which determines, for a piece of data, the smallest program that can produce that output. Equivalently, it determines how "compressible" data is. Take, for instance, the Mandelbrot set which contains an infinite amount of points (obviously finite in an image of it), but it can be generated by a relatively small program.

Furthermore if we look at strings, the string "0101010101…", which we discussed earlier, gets a very low Kolmogorov complexity as it is just the string 01 however many times we need to get back the string.

This sounds great! Kolmogorov complexity seems a powerful meassure for determining the randomness of our data. However, while the repeating sequence of 01s had a very obvious and easy-to-find pattern, what about a string like "01101011101100101011111010010"? No immediate pattern jumps out like it did in the example above. There probably is a pattern, but what if this string was $100x$ longer with other patterns in it? As it turns out, determining the Kolmogorov complexity of a piece of data is very hard. Not only is it very hard, but it is in fact undecidable in general.

NOTE: Possible useful defns for l8r →

Definition 2.2: Entropy Rate

The entropy rate of a stochastic process (X_n) is defined as

$$h(X) = \lim_{n \to \infty} \frac{1}{n} H(X_1, X_2, \ldots, X_n),$$

which measures the average uncertainty contributed by each new observation.

Definition 2.3: Conditional Entropy

For two random variables X and Y,

$$H(X|Y) = -\sum_{x,y} p(x,y) \log_2 p(x|y),$$

representing the expected uncertainty in X given that Y is known.

Definition 2.4: Min-Entropy

The min-entropy of a random variable X is

$$H_{\min}(X) = -\log_2 \left(\max_x p(x) \right),$$

representing the worst-case predictability of any outcome.

2.2 Sources of Entropy in Hardware

Microcontrollers can extract entropy from physical phenomena such as:

- Thermal noise (Johnson–Nyquist noise) in resistors or sensors.
- Electronic jitter in oscillators and ADCs.
- Fluctuations in analog sensors (light, temperature, sound, etc.).

These sources are often amplified and sampled via the ADC to produce random bits.

2.3 True vs. Pseudo Randomness

A pseudo-random number generator (PRNG) produces deterministic sequences from an initial seed, while a true random number generator (TRNG) relies on physical entropy. For hardware-based RNGs, ensuring unbiased and unpredictable output requires:

- High-quality analog entropy sources.
- Proper sampling rates and whitening.
- Statistical post-processing.

2.4 Statistical Tests of Randomness

We use the NIST SP 800-22 statistical test suite as a theoretical foundation. The most relevant tests include:

- Frequency (Monobit) Test
- Runs Test
- Autocorrelation Test

- Approximate Entropy Test
- Linear Complexity Test

Each of these tests returns a p-value indicating how likely the observed sequence could occur under true randomness.

2.5 Compression and Algorithmic Randomness

Kolmogorov complexity describes how compressible a sequence is. In practice, this can be approximated by compressing the data using a general-purpose algorithm (e.g., gzip) and comparing the compression ratio.

2.6 Signal Processing Perspective

Entropy in analog signals can also be characterized by:

- Autocorrelation function (should approach 0 for nonzero lag).
- Power spectral density (should be approximately flat for white noise).

3 Implementation

3.1 Hardware Setup

We will use the STM32 microcontroller, connected to one or more analog sensors. Candidate sensors include:

- Microphone or sound sensor.
- Light-dependent resistor (LDR).
- Temperature sensor.

The microcontroller samples the analog voltage via the ADC and stores or streams the digital values over a serial or network interface.

3.2 Data Transmission

Data is transmitted from the STM32 to a computer for analysis. This can be done using:

- UART/Serial connection.
- Ethernet or WiFi (via ESP module or similar).

We use a lightweight protocol to ensure each sample is timestamped and properly framed. The receiving server stores the data in a binary file.

3.3 Software Stack

On the analysis computer:

- Python scripts for entropy calculation and statistical tests.
- Optional C or Rust implementation for efficient batch processing.
- Visualization of autocorrelation and frequency distribution.

3.4 Data Preprocessing

Collected data may exhibit bias or patterns due to hardware noise or ADC quantization. We can apply:

- Mean removal and normalization.
- Bit extraction (e.g., least significant bit of ADC samples).
- Von Neumann debiasing if needed.

4 Results and Analysis

4.1 Entropy Estimates

We calculate Shannon entropy for bit sequences of various lengths, expecting values close to 1 for highly random data.

4.2 Frequency and Runs Tests

We evaluate balance between 0s and 1s and the expected number of alternations.

4.3 Autocorrelation and Spectral Tests

We compute autocorrelation functions and Fourier transforms to look for repeating structures or periodic interference.

4.4 Compression Ratio

We compress the data using a lossless compressor to check for compressibility. A ratio near 1 indicates high randomness.

4.5 Discussion of Findings

We discuss:

- Whether the observed randomness is sufficient for cryptographic use.
- How sensor noise compares to other hardware RNGs.
- Effects of sampling rate and environmental conditions.

5 Network Architecture and Subnetting

5.1 System Overview

We consider a distributed network of sensor-based TRNG nodes, each sending entropy samples to a central collector.

5.2 Network Design

A possible setup:

- Each microcontroller is connected to a local router.
- A dedicated subnet for sensor nodes (e.g., 192.168.10.0/24).
- Central analysis server on a separate subnet.

5.3 Subnetting and Addressing

We show how to subnet the network efficiently:

- Example: dividing a /24 into four /26 subnets.
- Assigning IP ranges for sensors, analysis nodes, and administration.

5.4 Data Security and Transmission Integrity

We briefly discuss:

- Packet integrity verification (CRC or checksum).
- Optional encryption for data in transit.
- Synchronization and time-stamping for accurate sampling.

6 Conclusion

We summarize:

- Theoretical feasibility of analog-sensor-based TRNGs.
- Experimental results and limitations.
- Potential for distributed entropy networks.

6.1 Future Work

- Hardware whitening circuits and amplification.
- FPGA or ASIC implementations.
- Scaling to larger networks and entropy pooling.

References

(Include IEEE papers, arXiv articles, and videos cited earlier.)