FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme / specialisation: Computer Technology: Reliable and Secure Systems	The *spring* semester, *2023* Open / ~~Confidential~~
Author: Oddvar Nordbø Øksendal	
Supervisor at UiS: Ferhat Özgur Catak	
Thesis title: 5G RF Spectrum-based Cryptographic Pseudo Random Number Generation for IoT Security	
Credits (ECTS): 30	
Keywords: Internet of Things Random Number Generation 5G Security	Pages: 65+ appendix: Stavanger, June 15, 2023

Abstract

This thesis presents a novel approach for generating truly random numbers in 5G wireless communication systems using the radio frequency (RF) spectrum. The proposed method leverages variations in the RF spectrum to create entropy, which is then used to generate truly random numbers. This approach is based on channel state information (CSI) measured at the receiver in 5G systems and utilize the variability of the CSI to extract entropy for random number generation. The proposed method has several advantages over traditional random number generators, including the use of a natural source of entropy in 5G wireless communication systems, minimal hardware and computational resource requirements, and a high level of security due to the use of physical characteristics of the wireless channel that are difficult for attackers to predict or manipulate. Simulation results demonstrate that the proposed method generates high-entropy random numbers, passes statistical randomness tests, and outperforms traditional random number generators regarding energy consumption and computational complexity. This approach has the potential to improve the security of cryptographic protocols in 5G networks.

Acknowledgements

I would like to thank my supervisor Ferhat Özgur Catak for all the help and guidance he has provided during my thesis, and for the opportunity to provide and support a promising new direction to this field of research.

In addition I would like to thank my family for the possibility to spend time working on this thesis.

Terminology

- Random number generator/generation RNG
- Random bit generator RBG
- Deterministic Random Bit Generator DRBG
- Non-deterministic Random Bit Generator NRBG
- Pseudo random number generator PRNG
- Cryptographically secure pseudo random number generator CSPRNG
- Channel State Information CSI
- Internet of Things IoT
- National Institute of Standards Statistical Test Suite NIST STS

Contents

Chapter 1

Introduction

The proliferation of the IoT has led to an unprecedented increase in connected devices, thus creating a need for secure communication channels. With the development of fifth-generation (5G) wireless communication systems, the demand for secure communication channels has increased even further. 5G technology offers higher bandwidth, lower latency, and higher data rates, and it is expected to revolutionize how we interact with the internet. However, deploying 5G networks in large, heterogeneous, distributed environments presents a significant security challenge. To address this challenge, secure communication protocols are necessary to safeguard communication between nodes and defend against malicious attacks. One of the critical challenges in 5G security is generation of secure random numbers.

Random number generation (RNG) is essential for ensuring cryptographic security in various applications, including digital signatures, encryption protocols, password generation, game development, and data transfer. Cryptographic applications require a source of high-quality randomness to generate keys and other secret parameters. Therefore, generating truly random numbers is essential to ensuring the security of these applications. However, in 5G networks, the randomness of generated numbers is often unreliable due to many devices with low entropy sources that are susceptible to attack or malfunction. The quality of random numbers plays a crucial role in the security of cryptographic protocols, including the Elliptic Curve Diffie-Hellman (ECDH) key exchange algorithms and Rijndael encryption

algorithms, which are commonly used in 5G networks. Numerous cryptographic protocols have been compromised due to the use of low-quality random numbers. Therefore, developing a reliable RNG solution is vital for ensuring higher security for communication protocols in 5G networks.

In the computer world, several ways of generating random numbers exist. Depending on what the numbers are used for, there are different types of generators. A random number generator (RNG) is the broad category that all of these generators belong to. This broad definition of an RNG, also known as a random bit generator (RBG), was introduced by the U.S. National Institute of Standards and Technology (NIST). NIST has developed a hierarchy of RNGs based on their level of security that classifies them into two categories: Non-Deterministic Random Bit Generators (NRBGs) and Deterministic Random Bit Generators (DRBGs). NRBGs are also known as True Random Number Generators (TRNGs) and DRBGs are also known as Pseudo-Random Number Generators (PRNGs). NRBGs are considered most secure and within the category of NRBGs there are two main methods of generating bits: Using physical processes (Considered most secure) as a noise source, or using other sources of randomness such as user input. DRBGs differ in the way that they generate random bits using mathematical algorithms. These may also rely on a seed value in order to generate bits, which can be predictable if the seed is known.

In the context of 5G security, generating truly random numbers is essential for ensuring a communication channels' confidentiality, integrity, and availability. However, more than traditional RNGs may be required for the requirements of 5G systems, as they may not be able to generate enough entropy or may be vulnerable to attacks. Therefore, there is a need for new and more robust RNGs that can provide the required level of security.

This thesis introduces a new and innovative technique for generating truly random numbers in 5G wireless communication systems using the radio frequency (RF) spectrum. The proposed method utilizes variations in the RF spectrum to generate entropy, which is then used to create truly random numbers. The proposed approach is based on channel state information (CSI), which measures the quality of the wireless channel. In 5G systems, CSI is measured at the receiver and sent to the transmitter to facilitate signal processing and beamforming techniques. The proposed method leverages the variability of the CSI to generate random numbers. This novel

approach represents a significant advancement in the field of random number generation and has the potential to address the challenge of unreliable random number generation in 5G networks. These advances are: first, it uses a natural source of entropy in 5G wireless communication systems. Second, it requires minimal hardware and computational resources, reducing energy consumption and cost. Third, it provides a high level of security since the randomness is generated based on the physical characteristics of the wireless channel that are difficult for an attacker to predict or manipulate.

In this thesis, existing literature on RNGs and different approaches for generating random numbers will be reviewed. Then, the proposed approach for generating random numbers using the RF spectrum in 5G wireless communication systems will be presented. To evaluate the performance of the proposed method simulations were conducted using a 5G system model, metrics used include entropy, security, and energy consumption. Results indicate that the proposed method generates high entropy random numbers and passes statistical randomness tests.

1.1 Motivation

Generating random numbers is a fundamental trait for most computer security applications. Some NRGBs rely on extra pieces of hardware in order to be used. This reduces the scalability in general and in particular for the Internet of Things (IoT). The need for secure and easy implementable NRBGs are a necessity.

1.2 Random number generation

Random number generation is a fundamental aspect of computer security. Cryptographic algorithms, such as encryption, digital signatures, and secure key exchange, rely on generating high-quality random numbers. The quality of the generated random numbers is crucial to the security of these applications. If the random numbers are predictable or biased, the entire system's security can be compromised. Therefore, generating truly random

numbers is essential to ensure the security of cryptographic applications.

Creating random numbers considered secure can be done in a multitude of ways. These differ depending on what the numbers intended usage are, and therefore there exists several types of generators. A random number generator is the broad category that all of these generators belong to.

> "A process used to generate an unpredictable series of numbers. Also called a Random bit generator (RBG)" [23].

This broad definition of an RNG aka RBG from the U.S. National Institute of Standards and Technology (NIST) could be attained to the top category in Figure 1.1.

1.2.1 Methods for random bit generation

There are several methods for random bit generation, such as hardware-based, software-based and hybrid solutions. Hardware-based methods rely on physical processes such as radioactive decay or thermal noise to generate randomness. These methods have been used for many years and are known for their ability to generate random numbers with high entropy. However, they are expensive and require specialized hardware, which gives them less scalability.

Software-based approaches rely on mathematical algorithms to generate random bits. These methods are less secure than hardware-based methods, as they may be vulnerable to attacks, but they are more cost-effective and easier to implement. The generated bits may be predictable if the algorithm is weak or if the seed value is known.

Hybrid solutions combine hardware-based and software-based solutions to generate random numbers. They normally rely on a hardware-based entropy source which in turn is used as a seed for a software-based PRNG. The hybrid approach provide a good middle road providing both high entropy and scalability.

1.2.2 The National Institute of Standards guidelines

Cryptographic standards and guidelines have been developed to ensure the security of random number generation, such as the NIST Special Publication 800-90A. This publication defines two classes of random number generators: Non-Deterministic Random Bit Generators (NRBGs) and Deterministic Random Bit Generators (DRBGs).

NRBGs that use physical processes, such as radioactive decay or thermal noise, to generate random numbers are considered the most secure, as they generate truly random numbers. However, they are also expensive and may not be practical for some applications.

NRBGs may use other sources of randomness, such as user input, hard drive activity etc. DRBGs or PRNGs, on the other hand, generate random numbers using mathematical algorithms. DRBGs that rely on a seed value to generate a sequence of numbers are the least secure, which can be predictable if the seed value is known.

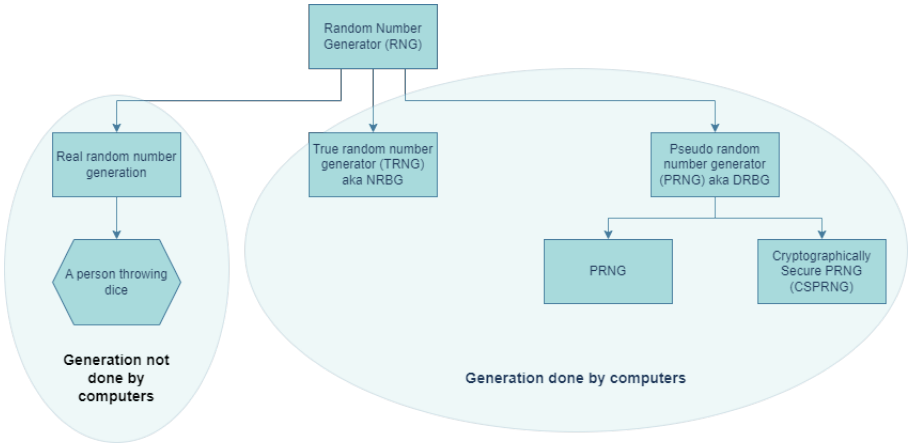


Figure 1.1: Tree overview of different RNG types

1.2.3 Methods explored

Figure 1.1 displays two main categories: Numbers generated by computers and numbers not generated by computers. In this thesis the focus lies solely

on the right side of the tree, numbers generated by computers. NRBGs and DRBGs are Generators that fall into these two categories and are defined as such by the NIST:

> "A device or algorithm that outputs a random sequence that is effectively indistinguishable from statistically independent and unbiased bits. An RBG is classified as either a deterministic RBG (DRBG) or an non-deterministic RBG (NRBG)." [7]

Pseudo randomness

Pseudo randomness is stated in the Merriam-Webster dictionary:

> "being apparently rather than actually as stated : SHAM, SPURIOUS" [18]

It is randomness that may appear to be random at a first glance, but if analyzed, it reveals that it is not necessarily so.

To generate random numbers, computers use algorithms or processes known as pseudo random number generators (PRNGs). These methods ensure that the numbers appear random but are deterministic, meaning they will produce the same output if given the same seed. A DRBG starts generating values based on a seed, which is essential for statistical applications but not cryptographic ones. If attackers know the seed, they could predict future numbers and determine which ones have already been produced, making it unsafe for cryptographic purposes.

Cryptographically Secure PRNG

A CSPRNG is a type of PRNG that is suitable for cryptographic applications. To meet the requirements of being a CSPRNG, it must pass the next-bit test and be able to withstand a state compromise extension. This definition is commonly used but could also be defined as passing only the

Next-bit test and being indistinguishable from a truly random source. In this case, two subgroups would need to be specified, one with backtracking resistance and one without.

- Next-bit test

  > "We say that a sequence of bits passes the next bit test for at any position i in the sequence, if any attacker who knows the i first bits (but not the seed) cannot predict the $(i + 1)$ with reasonable computational power." [34] [31]

- Backtracking resistance

  > "Backward security/Break-in recovery. Future output of the generator looks random, even to an observer with knowledge of the current state, provided that the generator is refreshed with data of sufficient entropy." [3]

According to these definitions, the output of a CSPRNG should be impossible to differentiate from a truly random output, even if someone knows the internal state. The NIST online glossary does not have an entry for CSRPNG, so it's debatable whether a CSPRNG can be classified as NRBG or DRBG. This thesis suggests categorizing certain CSRPNGs as NRBG as long as they have ongoing access to an entropy source, as stated in quote 1.2.3.

Non-Deterministic Random Bit Generator

Where a PRNG use algorithms to appear random a TRNG take a physical phenomena and extract it for use in a computer. Imagine it as having a die connected to your computer. In reality what is used is easier to connect to a computer than a regular die. Generating random numbers using physical phenomena make them non-deterministic. NIST defines a NRBG or TRNG as this:

> "An RBG that always has access to an entropy source and (when working properly) produces output bitstrings that have full en-

> tropy. Often called a True Random Number (or Bit) Generator. (Contrast with a deterministic random bit generator)." [7]

1.2.4 Metrics for evaluation

Various metrics can be used to determine the quality of randomly generated numbers, including entropy, statistical properties, and cryptographic strength. Entropy measures the level of randomness in the generated numbers and is based on their uncertainty. Higher entropy indicates more unpredictable numbers. Statistical properties, such as mean, variance, and distribution, assess the uniformity and randomness of the generated numbers. Cryptographic strength evaluates the ability of the generated numbers to resist attacks.

> "The strength of a CSPRNG is directly proportional to the source of entropy used for seeding it (and re-seeding it). We can safely conclude that the security of a crypto-system depends on configuring the highest level of entropy for seeding a CSPRNG algorithm." [28]

Talking about entropy there are two distinct definitions. That is entropy, and entropy source. They are defined as following by NIST:

- **Entropy:** "A measure of the disorder or randomness in a closed system. The entropy of uncertainty of a random variable X with probabilities $p_i, \ \ldots, \ p_n$ is defined to be $H(X) = -\sum_{i=1}^{n} p_i \ log \ p_i$. " [20]
- **Entropy source:** "A physical source of information whose output either appears to be random in itself or by applying some filtering/distillation process. This output is used as input to either a RNG or PRNG." [20]

1.2.5 Why random numbers in computers?

Random numbers are crucial in computer systems as they play a vital role in cryptography. If the numbers produced can be easily calculated in a short amount of time, then the security of the system is at risk. To emphasize the importance of secure random numbers, potential attacks on the system are discussed[9].

- **Direct cryptanalysis attack**

  When an attacker obtains a significant amount of numbers from the RNG, they can use this to analyze it and figure out how it differs from a truly random stream of numbers.

- **Input-based attacks**

  When an attacker use the input of an RNG to cryptanalyze it.

- **State compromise extension attacks**

  The attacker has in some way learned about the internal state of the RNG and then this can be used in several ways to further compromise the RNG.

A real world example of an attack is when the website Hacker News was hacked. [6] In this white hat attack, they managed to take control of login IDs of other users and was able to impersonate them. This was possible because of a low entropy in the seed for the RNG used. This made it possible for the attacker to find the internal state where the seed was created and replicate it.

1.3 Thesis

Lately, more people have been exploring using the RF spectrum to create random numbers. The RF spectrum refers to the frequencies used for wireless communication, and its signal strength and noise variations can create entropy for generating truly random numbers. Compared to traditional

RNGs, RF-based RNGs offer greater entropy, enhanced security, and reduced energy consumption. However, challenges like calibration and attack vulnerability must be addressed for successful implementation.

This thesis presents a new method of generating random numbers in 5G wireless communication systems by utilizing variations in the RF spectrum. This approach generates entropy and produces truly random numbers, potentially surpassing the limitations of traditional RNGs. By evaluating the performance of this approach using metrics such as entropy, security, and energy consumption, its effectiveness can be determined in providing a more secure and efficient way to generate random numbers in 5G systems.

The feasibility of methods proposed will be tested using tests from NIST 800-22 Rev. 1 and NIST SP 800-90B.

The test data was generated through a simulation package in Matlab [17]. To make it more user-friendly, images of the spectrum were created at a specific point. The data from that point on the image was then processed through bit-wise operations for conditioning. This resulted in entropy for a NumPy package RNG in Python. The numbers were generated using an RBG from the same NumPy package in Python.

To find a frame from the image with the high entropy a NSGA2 optimization algorithm has been applied. Results indicate that it may be a promising entropy source for IoT devices.

1.4 Background

In the background section, we will explain the technologies that were chosen. Furthermore, we will also provide information on related works that focus on entropy sources and the generation of random numbers through radio frequency.

1.4.1 Programming language

Choosing which programming language to use were based on these ranked criteria:

1. Time to learn
2. Third party packages
3. Execution time

The chosen criteria were ranked based on the importance of time in completing the work efficiently. Having third-party packages or code repositories that provide the necessary methods and tests for the thesis is crucial, as it saves time that would have been spent learning a programming language. While execution time is a valuable metric, it is less critical during this research phase, as more work must be done before the novel approach can be used.

After careful consideration, Python was chosen as the most suitable programming language for the thesis project. This decision was made based on the time-saving advantage of not having to create something that already exists. Other languages, such as Golang and C++, were also considered but ultimately ruled out. Golang required the use of third-party packages for specific solutions. In contrast, C++ requires a significant amount of time to learn and could cause more harm to the system due to its low-level capabilities. However, if a real system were created, C++ could provide better execution times.

1.4.2 NIST SP 800-22 Rev. 1a

NIST special publication 800-22 is a statistical test suite for RNG and PRNGs for cryptographic applications. In this thesis this test suite were chosen because it is a widely used and has good implementations already available for free use. It is also a standard that has a well known and trustworthy source. Further work is also being done to improve on the

tests, but have not yet been included in the test suite. The newer work consists among others of tests specific for an entropy source in NIST SP 800-90B. These tests will also be used in this thesis to supplement the standard tests. That it is widely used gives many other publications and results from tests done to compare our results with. These comparisons grants a better foundation for conclusions and whether using the 5G radio spectrum is viable.

There are multiple other test suites for verifying the randomness of an RNG or PRNG where the diehard test suite is a valid option. It has not been chosen as the implementation of the diehard test in python does not have a working editon, and it is not used as an extra test suite at this stage. Adding an extra test suite, may give more results to base a decision, but depending on the implementation it may include having to format the data specifically to the tests more. The extra work necessary for running another battery of tests is considered greater than the rewards. Both the diehard and the NIST test suite contain a few tests that are equal, by my count that is 5. (diehard consist of 12 tests and NIST SP 800-22 Rev. 1a 15.)

Tests in NIST SP 800-22 Rev. 1a that are run is the following: [20]

1: Frequency (Monobits) Checks whether the proportion of ones and zeroes for the sequence is approximately equal to that of a truly random sequence. Assesses how close the fraction of ones are to a half (0.5).

2: Frequency within a Block Focuses on the proportion of ones in a M-bit blocks to be approximately M/2.

3: Runs Looks at the oscillation between ones and zeroes in a sequence, and determines whether it is too fast or too low.

4: Longest Run of Ones in a Block Determines whether the length of longest run of ones within M-bit blocks of a sequence is as expected from a random sequence.

5: Binary Matrix Rank Checks whether there is a linear dependence between sub-matrices and the original sequence.

6: Discrete Fourier Transform (Spectral) Detects periodic patterns in peak heights of the Discrete Fourier Transform of the se-

quence.

7: Non-overlapping Template Matching Counts number of occurrences of pre-specified strings. For this and the Overlapping Template Test an m-bit sliding window is used to search for the pre-specified strings.

8: Overlapping Template Matching Counts number of occurrences of pre-specified strings. (Differs from the non-overlapping template matching in the behavior when a match has been found.) For this and the Non-overlapping Template Test an m-bit sliding window is used to search for the pre-specified strings.

9: Maurer's "Universal Statistical" Can the sequence be significantly compressed without loss of information. (Significantly compressible sequences are considered non-random. [20])

10: Linear Complexity Determines whether or not the sequence is considered complex enough to be considered random using a linear feedback shift register. Random sequences have a longer feedback register.

11: Serial Checks all M-bit sequences and that they are just as likely to appear as any other M-bit sequence.

12: Approximate Entropy Check all adjacent M-bit sequences (M and M+1) that they are as likely to appear as in a random sequence.

13: Cumulative Sums (Cusum) Defines random walks in the sequence and checks if the deviations from the expected random walk is near zero.

14: Random Excursions Contain a total of eight tests that conclude by themselves. The tests looks at deviations from a cumulative sum random walk.

15: Random Excursions Variant Contain a total of eighteen tests that conclude by themselves. The tests looks at deviations from a cumulative sum random walk.

1.4.3 NIST SP 800-90B

NIST Special Publication 800-90B is a recommendation that states design principles and requirements for entropy sources used by RBGs as well as tests for validating entropy sources. The document outlines critical design principles and requirements for entropy sources, including the amount of entropy generated, the source, and the mechanisms used to extract and test the entropy. It also specifies tests that can validate the quality of the entropy source, such as the NIST Statistical Test Suite.

One of the critical goals of the NIST Special Publication 800-90B is to ensure that cryptographic RBGs generate high-quality, unpredictable, and statistically independent data that can withstand attacks by cryptanalytic techniques. The document emphasizes the importance of the three components - entropy source (NIST SP 800-90B), algorithm (NIST SP 800-90A), and method to combine the first two (NIST SP 800-90C) - in creating a secure cryptographic RBG.

By following guidelines in this publication, developers can enhance the security and trustworthiness of cryptographic RBGs, and ensure that they meet the requirements of various applications in which randomness is critical.

In summary NIST Special Publication 800-90B is an essential resource for anyone involved in designing, implementing, and validating entropy sources for cryptographic RBGs.

In validating an entropy source using the NIST Special Publication 800-90B suite, two crucial steps are involved: track determination and entropy estimation of the given data sequence. The determination of the track requires a minimum of 10^6 samples of binary data, which are then classified as IID (Independent and Identically Distributed) or Non-IID. In the case of IID data, the entropy estimate is obtained using the MCV (Most Common Value) estimator. Conversely, if the data is classified as Non-IID, ten (10) estimators are applied, and the minimum result of all the estimators is returned as the sequence assessment. After the track is determined, a further test is conducted to verify the given estimate. These tests ensure that the generated random numbers meet the required level of randomness and security for cryptographic applications.

Entropy estimation strategy and data collection is displayed in figure 2 in the NIST SP800-90B. [21]

Entropy Estimation strategy

Min-entropy is a conservative measure calculated based on the probability of the most likely outcome. It is commonly used in cryptographic applications to ensure that the level of uncertainty in the generated random numbers is sufficient for their intended purpose.

$$min - entropy = -log_2(pmax) \tag{1.1}$$

Data collection

Data collection for the tests from SP 800-90B has to be done in a specified order to be valid. Data used for our entropy source includes a non-vetted conditioning component that is the chaotization algorithm. All these steps have been ensued.

In Figure 1.2 in the NIST SP 800-90B a description of where the test suite is to be tested and where the conditioning is applied is displayed.

1. A sequential dataset of at least 1 000 000 samples obtained directly from the noise source.
2. If the entropy source includes a conditioning component not listed, a conditioned sequential dataset of at least 1 000 00 samples needs to be collected.
3. For the restart tests the entropy source must be restarted a 1000 times, and for each restart 1000 consecutive samples shall be collected directly from the noise source.

Determining the track

In order to determine whether the dataset is to follow the IID track or the non-IID track four required steps need to be satisfied.

1. Submitter must make an IID claim based on an analysis of the design. And provide rationale for the IID claim.
2. The raw sequential dataset must pass the IID statistical tests.
3. The restart dataset must pass the IID statistical tests.
4. If the data contains a non-vetted conditioning component, the conditioned dataset must pass the IID statistical tests.

Restart tests

The restart tests are designed to ensure three aspects:

1. Noise source outputs generated are drawn from the same distribution for every restart.
2. The distribution of samples are independent of the start position in the sequence.
3. Knowledge of other restarts does not offer an advantage in predicting the next.

To restart the noise source for data collection, a different seed was used for a PRNG. This PRNG selected one of the 2016 images and determined the frame size to use.

According to SP 800-90B these data need to simulate the restart process expected in a real-world use. In a real-world use this could be done in at least three ways and all of them would start by restarting the 5G unit itself. In list 3 these three ways to simulate the restart process are described.

1. Choosing the frame size used to extract entropy from the 5G spectrum at random.
2. Using the previous frame size that has been found using optimization to extract entropy from the 5G spectrum.
3. Running the optimization algorithm to find a new optimal frame size to extract entropy from the 5G spectrum.

Out of the three methods for restarting in list 3 we have simulated the first one. We have a limited amount of samples (2016), and we do not have access to a physical device. Knowing that this may not be the desired way to generate the data it may still provide information whether using this noise source is worth looking into. Seeing that if we had a physical tests rig, we would have chosen method three as it is harder for an attacker to determine the output of the optimization algorithm than it is to keep an already discovered value and reusing it.

1.4.4 Numpy

In this thesis we needed to do some array calculations and also have RNGs that can be seeded from the 5G entropy source. Numpy is a wrapper of a library implemented in C which makes the runtimes of methods and functions faster. All the different RNGs implemented, from Numpy [10] are listed and explained briefly below: All the generators provides doubles and unsigned 32 and 64-bit integers that must be consumed by a Generator object.

PCG64 Is a 128-bit implementation of O'Neill's permutation congruential generator.

PCG64XSDM Is a 128-bit implementation of O'Neill's permutation congruential generator. Uses a stronger DXSM output function.

MT19937 Mersenne Twister. Not cryptographically secure if one observes a sufficient amount of iterations. (624) If that is done further operations can be predicted.

Philox A 64-bit PRNG that uses a counter based design. It is based on weaker and faster versions of cryptographic functions.

SFC64 Is a 256-bit implementation of Chris Doty-Humphrey's Small Fast Chaotic PRNG. Distinct seeds will not run into each other for at least 2^{64} iterations.

These PRNGs are not necessarily cryptographically secure, but if they are reseeded and input an entropy source often enough, predicting further outcomes becomes improbable. "An RBG that always has access to an entropy source and (when working properly) produces output bitstrings that have full entropy. Often called a True Random Number (or Bit) Generator. (Contrast with a deterministic random bit generator)." [7] This definition can be stated as such, because many RNGs that in itself are not secure. May only be misused if enough random numbers are generated with the same seed. Thus if we always reseed the RNG with a source that has high entropy. To cryptanalyze the data, or find a way to replicate or predict future numbers would require an unlikely amount of resources. Additionally if the method for producing the seed was discovered by an attacker, it would be improbable for the attacker to discover the seed value.

Even though using PRNGs with an always available entropy source, may make them secure enough. Tests were run on a different set of PRNGs that are cryptographically-based. Although they are cryptographically based it is only stated that SPECK128 is suitable for usage in encryption. These are all implemented in RandomGen [27] and the RandomGen package synergies well with Numpy.

AESCounter A 64-bit PRNG that use a counter-based design based on AES-128.

ChaCha A 64-bit PRNG that use a counter-based design based on the ChaCha cipher.

HC128 Developed by Hongjun Wu. [32] Produces a keystream suitable for encryption. States that it is the fastest software-only encryption quality bit generator.

SPECK128 A 64-bit PRNG that use a counter-based design based on the SPECK-128 cryptographic function.

ThreeFry A 32 or 64-bit PRNG that uses a counter based design, based on cryptographic functions.

1.4.5 Multi-objective optimization and NSGA2

Multi-objective optimization (MOO) is a method used to find the best possible solution to problems with multiple objectives. It is a part of multi-criteria decision-making. In this study, MOO was used to find the optimal entropy source in a digital image. The optimal solution was based on all fifteen tests from the NIST test suite. However, decision-making becomes challenging when multiple essential objectives need to be optimized. MOO may result in complex non-polynomial problems, so an approximation of the solution is necessary.

Solutions from a multi-objective optimization are called nondominated or Pareto optimal. A pareto optimal solution is a solution where the values of any single objective cannot get better without the value from another objective getting worse.

There are several options for implementing MOO and NSGA2 s.a: Pymoo, Gurobi, Pyomo, PulP, etc. Out of these Pymoo was found to be the best fit. Using Gurobi requires a license, they do provide academic licenses for a single user and single computer, but that reduces the scalability. Pyomo and PulP was not chosen for the reason that it requires a more detailed knowledge to implement NSGA2. In pymoo NSGA2 is already implemented, all that needs to be defined is the specific problem that is to be solved.

For this thesis pymoo [4] was chosen to implement multi objective optimization.

NSGA2 (Non-dominated Sorting Genetic Algorithm 2) is a widely used algorithm for solving multi-objective optimization problems. NSGA2 maintains a population of solutions and uses a non-dominated sorting approach to evaluate the quality of the solutions. The algorithm uses a crowding distance operator to support diversity in the population and avoid premature convergence to a local optimum. NSGA2 effectively solves many real-world problems, such as multi-objective engineering design and financial portfolio optimization.

In NSGA2, the optimization problem is represented as follows:

$$\min_x F(x) = (f_1(x), f_2(x), ..., f_k(x)), \tag{1.2}$$

where x is the decision variable vector, and $F(x)$ is the vector of the objective functions $f_1(x), f_2(x), ..., f_k(x)$. The objective functions are subject to constraints $g(x) \leq 0$, and $h(x) = 0$, where $g(x)$ and $h(x)$ are the inequality and equality constraints, respectively.

NSGA2 uses a non-dominated sorting approach to evaluate the quality of the solutions. The non-dominated sorting approach divides the population into different fronts, where the first front contains the non-dominated solutions. The crowding distance operator is used to maintain diversity in the population and avoid premature convergence to a local optimum. The crowding distance operator measures the density of the solutions in a front, and it is used to select the solutions for the next generation.

Crowding distance in NSGA2 is the Manhattan distance in the objective space. Manhattan distance is the shortest distance between two points that is possible to follow.

Figure 1.2: © Kartverket [13]: Section of a map to illustrate the manhattan distance.

As can be seen in figure 1.2 the black line is the shortest, but the red line would be the Manhattan distance between point A and B.

NSGA2 has several advantages over other algorithms for multi-objective

optimization. It is easy to implement and has a fast convergence rate. It also provides a good balance between exploration and exploitation, which is essential for finding a diverse set of solutions.

1.4.6 Related works

Generating random numbers is an essential aspect of securing communication protocols. Various methods have been suggested in the literature to produce random numbers, such as hardware-based, software-based, and hybrid solutions.

Hardware-based methods are called random number generation techniques based on physical phenomena like thermal noise [15], quantum tunnelling [30], and radioactive decay [24]. These methods have been used for many years and are known for their ability to generate random numbers with high entropy. However, they are expensive and require specialized hardware, which gives them less scalability.

Software-based random number generation techniques rely on algorithms to generate random numbers. These algorithms utilize pseudo-random number generators (PRNGs) to generate random numbers based on a seed value. While these techniques are cost-effective and scalable, their randomness may be predictable and vulnerable to cryptographic attacks.

Hybrid solutions combine hardware-based and software-based solutions to generate random numbers. These solutions typically use hardware-based sources of entropy to seed software-based PRNGs, which generate random numbers. The hybrid approach provides the best of both worlds, ensuring high entropy generation capabilities and scalability.

Rădoi et al. [26] propose a reconfigurable hardware approach for generating random numbers in wireless sensor nodes, which is crucial for ensuring secure communication. The authors perform a randomness analysis of the generated data using the The National Institute of Standards and Technology (NIST) methodology to validate their proposed design. They show that using a Field Programmable Gate Array (FPGA) design that utilizes data collected from onboard sensors can lead to faster and more versatile random number generation, despite consuming more energy than traditional

microcontrollers.

Xu et al. [33] proposed an efficient authentication mechanism for smart, collaborative networking on high-speed trains (SCN-R), which relies on a novel chaotic, RNG design based on two logistic maps. The proposed RNG is used to generate and validate one-time passwords (OTPs) of different lengths to support different authentication applications. The authors address the authentication vulnerabilities caused by fast-moving objects by introducing this efficient authentication mechanism that demonstrates feasibility and effectiveness under real-world conditions.

Hossain et al. [11] proposed a randomized pulse-based data encoding scheme for secure and high-speed wireless communication. They encode 4-bit data as a symbol using 16 distinct orthogonal pulses and modulate the bit stream randomly to ensure security. The pulses are randomized using Ferroelectric Random Access Memory (FRAM)-based high-quality random numbers. The proposed scheme provides physical-layer security and supports $M = 2^k * k$ bits data by N $p = 2^k$ different orthogonal pulses. Using MATLAB simulation, the authors verified the power requirements of individual pulses defined by the Federal Communication Commission (FCC). The results demonstrate the feasibility and effectiveness of the proposed scheme for secure wireless data communication.

A number of TRNGs today are DRAM based, Khaled Humood et al. [12] propose a novel approach based on controlling the word line voltage supply part of the DRAM chip during random number mode. The results are considered a milestone towards low cost and high effecient secure hardware for IoT applications. Their method provide random numbers with no post processing that passes the complete NIST STS.

Bikram Paul et al. [25] proposes two PRNGs: BluXor and MPCG. Both made by combining/modifiying Blum-Blum-Shub(BBS) with Xorshift and Permuted Congruential PRNGs respectively. These are tested against most of the tests in the NIST STS and passes the tests run.

Sanu K. Mathew et al. [16] has created μRNG, a TRNG that is created for IoT and wearable platforms. It compines the entropy of multiple independent sources to generate an output bitstream. Three independent self-calibrating all-digital entropy sources, coupled with XOR feedback shift-

register, with low energy consumption and high throughput of numbers. This passes all tests in the NIST STS, and has a lower bound min-entropy greater than 0.99.

Yingnan Sun et al. [29] proposes using outputs of Inertial Measurement Units (IMU) worn by users as an entropy source. Their method divides these signals into gait cycles and generete bits by comparing energy differences between sensor signals in a gait cycle and the averaged IMU signals in multiple gait cycles. The numbers generated was testen on the NIST-STS and passed the tests run.

The requirement for secure and low-cost encryption techniques is essential in the Internet of Things (IoT) devices. Ansari et al. [2] a low-cost True Random Number Generator (TRNG) circuit that can generate random keys for encryption using non-deterministic signals from sensors. The proposed TRNG employs a simple hardware setup consisting of an Arduino Uno board, LDR sensor, and sound sensor. The processing algorithm uses a modular equation and EXOR logic. The generated random numbers pass all the randomness tests given in the NIST-STS test. The proposed TRNG can be used in applications such as cryptocurrency wallets, video games, and other IoT devices requiring true random numbers. This work provides a simple and cost-effective solution for generating secure random numbers. Another study proposed a light-weight implementation of a previously proposed latch-based TRNG using FPGA while maintaining the quality of the generated random numbers [8]. The proposed TRNG with only 16 latches passes the NIST SP 800-22 test suite by accumulating the generated random numbers fifteen times with an XOR operation for each output word. In contrast, the original TRNG required 248 latches. The paper highlights the improvement in the quality of the random sequence by XOR-ing temporally interleaved series of bits.

Researchers have recently investigated various methods of generating random numbers using wireless communication channels, such as Wi-Fi and cellular networks. These techniques utilize the inherent variability of wireless communication channels to generate random numbers. One such approach is using channel state information (CSI) to generate random numbers.

1.5 Outline

A brief outline of main chapters in the thesis and contributions of the thesis.

The contributions of the thesis summarized are the following:

- Giving an example of how 5G RF signals can be used to improve security of IoT devices
- Providing data, and test results as well as an analysis of the data.
- Show that usage of an optimization algorithm for finding an optimal entropy source provides good results.
- Displays a list of several runs of the results with different RNGs to be analyzed side by side, as well as compared to results from other relevant works.

Chapter 1: Introduction Explains the general ideas behind the thesis and background for why certain technologies have been chosen.
Chapter 2: System model and architecture Describes the system architecture and model with tables and figures.
Chapter 3: Results All results from experiments done in the thesis are displayed using tables, plots and figures.
Chapter 4: Discussion The results from experiments in the results chapter will be discussed with pros and cons. In addition an outline of a theoretical solution to how the results can be used in a real system will be discussed.
Chapter 5: Conclusion A conclusion to wrap up the report as a whole as well as a section on future works.

Chapter 2

System model and methodology

Figure 2.1 illustrates the proposed system model. Pilot signals are collected from the wireless communication channel and are used to compute the magnitude of the selected frequency band. The magnitude values are normalized and used to generate a spectrogram, from which the entropy is estimated. The entropy values are then used as a seed to generate a sequence of random numbers.

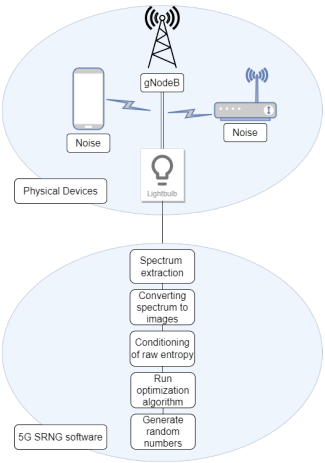


Figure 2.1: Architecture of 5G SRNG

Once a spectrogram is generated, the next step is to extract entropy from the entropy source. This involves measuring the unpredictability of the magnitude of the spectrum, which is used to create a sequence of random numbers. These numbers must pass multiple randomness tests to ensure they are high quality and suitable for cryptographic protocols. The proposed algorithm 1 provides a secure and efficient way of generating random numbers in IoT devices by utilizing the pilot signals already available in wireless communication systems. This approach can significantly enhance the security of cryptographic protocols by providing high-quality random numbers to generate secret keys.

Figure 2.1 portrays a IoT light bulb as a IoT device that is to implement the 5G spectrum extraction method. In a live system this could have been any 5G enabled IoT device. Noise sources that can be a multitude of things that affect the 5G Radio spectrum that is to be extracted such as; people with 5G cellphones walking by, weather conditions and other wireless devices within the same spectrum.

As all physical devices were emulated using the 5G toolbox in Matlab, the architecture model has been modified accordingly. Figure 2.2 depicts the complete architecture with the changes made from the original architecture shown in Figure 2.1.

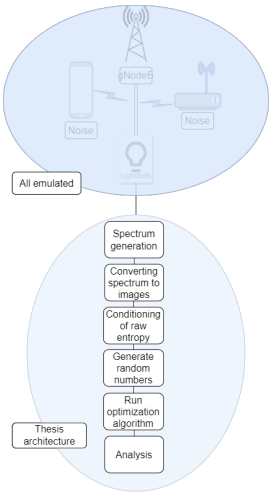


Figure 2.2: Architecture of the system for the thesis.

2.0.1 Computer specifications

The computer used for the experiments is a 5 years old laptop with the following specifications:

Component	Make/Model
Operating system	64-bit Windows 10 Home
Processor	Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
Memory	16 GB RAM
Graphics	NVIDIA GeForce GTX 1050
Storage	HGST HTS721010A9E630: 1 TB 7200RPM

Table 2.1: Specifications for the computer used to run experiments

2.1 Methodology

In summary it can be explained that signals from the 5G Radio spectrum has been generated, and then converted to images. These images are further conditioned to increase the randomness. The conditioned images is used to seed a random number generator that is used in a optimization using NSGA2 to optimize the results from a NIST test suite. A brief analysis of the results are done using plots and algorithms supplied by Pymoo.

The next subsections show how the thesis have been executed in general, by following the steps from figure 2.2.

Spectrum generation

The dataset used was synthesized using the MATLAB 5G Toolbox. This dataset consisted of frames that were 40 milliseconds long, and each frame was randomly shifted in the frequency domain. It was assumed that the 5G signals were within the specified frequency band range, and the network's performance was evaluated based on the varying random bands. The sampling rate of 61.44 MHz was considered sufficient to process the 5G signals

effectively. To generate the corresponding RGB spectrogram images of size 169×369, the complex baseband signals were transformed using a Fast Fourier Transform (FFT) with a length of 4096.

The parameters for generating all the data is given in tables 2.2, 2.3 and 2.4.

Although there exist other simulation software for 5G, this package was chosen as it consisted of all the data necessary and it was ready for use. It has also been used in a wide range of other papers. A search on scholar.google.com with the query: *5g simulation "matlab"* gives over 2000 results for articles published after 2023 (59 000 for anytime).

Creating all of the data for the thesis could also have been done using a physical setup, although it was not a feasible approach at the current state of research.

5G NR Parameter	Value	Units
Bandwidth	[10 15 20 25 30 40 50]	MHz
Sub-Carrier Spacing (SCS)	[15 30]	kHz
SSB Period	[20]	ms

Table 2.2: 5G Near Radio Parameters

Channel Parameter	Value]	Units
SNR	[40 50 100]	dB
Doppler	[0 10 500]	Hz

Table 2.3: Channel parameters for 5G

Generation start data	Value	Units
Number of Frames	2048	Integer
Image size	612 x 14	integer x integer
Number of subframes	40	correspond to ms
Sample rate	61.44	MHz

Table 2.4: Parameters for the generation of data

Converting spectrum data to images

The spectrum data was converted into images using the matplotlib package in Python. The data was split into train and validation datasets, with 2016/32 data points allocated to each. The train data was used for all further tests as they relied on most of the generated images. To ensure that the resulting images did not contain large white spots, the data was plotted and zoomed in before conversion to images, which were then saved in PNG format. The resulting images had a size of 169×369 pixels.

To use these images as an entropy source, they must be loaded into an array, from which a selected frame size is used for conditioning the data to create the entropy source. Figure 3.1 shows example images.

Conditioning entropy based on image

The proposed 5G-SRNG algorithm by Catak et al. previous work [5] is utilized to condition the extracted entropy from the spectrogram using a specified frame size.

Algorithm 1 Proposed 5G-SRNG [35]

Require: $5G\ Spectrogram : \mathcal{D} \in \mathbb{R}^{m\,x\,n}, framesize : c,k$

$x_{start}, y_{start} \leftarrow random(0, m-c), random(0, n-k)$
$x_{end} = min(x_{start} + c)$
$x_{end} = min(y_{start} + k)$
$frame \leftarrow \mathcal{D}[x_{start} : x_{end},\ y_{start} : y_{end}].flatten()$
$seed \leftarrow 0$ ▷ Initialize *seed* with 32-bit float representation of 0
for $t \in frame$ **do**
    $seed \leftarrow seed \oplus t$ ▷ XOR operator
    $seed \leftarrow seed \oplus seed << 13$ ▷ Shift-left the previous *seed* value 13 bits, then perform XOR operator
    $seed \leftarrow seed \oplus seed >> 17$ ▷ Right-right the previous *seed* value 17 bits, then perform XOR operator
    $seed \leftarrow seed \oplus seed << 5$ ▷ Shift-left the previous *seed* value 5 bits, then perform XOR operator
end for
return *seed*

Algorithm 1 shows the pseudocode of the proposed 5G-SRNG. Here m was defined as the size of the 5G Spectrogram's ($\mathcal{D}$) rows, n as the size of the columns and $frame$ as a smaller $\mathcal{D}$. Both x_{start}, y_{start} was defined as two starting points to select the random part (i.e. convolution) of $\mathcal{D}$. Both x_{end}, y_{end} was defined as two endpoints of the selected $frame$. x_{start}, x_{end}, y_{start} and y_{end} are random entries of $\mathcal{D}$. Here $frame$ was defined as a smaller fixed-size $\mathcal{D}$ from 5G Spectrogram. $frame$ contains $c \times k$ elements (i.e. pixels) from $\mathcal{D}$ with $c \geq 1$ and $k \geq 1$. $frame_{i,j}$ is the substantial single element (i.e. pixel) of $frame$. *seed* is the main contribution of 5G-SRNG. The iterations was defined (t) as the loop through the selected $frame$'s pixels. $\oplus$ is a simple operator. $<<$ and $>>$ represent the left and right shifts, respectively. *seed* contains the total value of the XORed $frame$'s pixels. c and k are the variables we can control the randomness of the output by changing the size of c and k. Increasing the size of c and k will increase the number of elements in $frame$, possibly increasing the output's entropy. The entropy of the output can be measured using standard randomness tests, such as the NIST Statistical Test Suite. If the output passes these tests, it can be used as a reliable source of random numbers in cryptographic protocols.

The proposed 5G-SRNG algorithm could be implemented on IoT devices with minimal hardware requirements. The algorithm uses only basic arithmetic and logical operations, which can be efficiently implemented using hardware or software. The algorithm also requires minimal memory, as it only needs to store a single value (i.e., *seed*) during its execution.

Given a spectrogram $\mathcal{D}$ with dimensions $m \times n$, a frame size c, k, and a randomly selected frame $frame$ of size $c \times k$, we can represent the starting position (x_{start}, y_{start}) as two random variables X and Y, respectively, with uniform distributions over the ranges $[0, m - c]$ and $[0, n - k]$, respectively.

Then, we can represent the ending positions x_{end} and y_{end} as:

$$\begin{aligned} x_{end} &= x_{start} + c \\ y_{end} &= y_{start} + k \end{aligned} \tag{2.1}$$

Next, we can represent the selected frame $frame$ as a one-dimensional array of size $c \times k$, which can be flattened into a vector $\mathbf{f}$ of length ck. Then, we

can represent the seed value as a 32-bit float variable S, which is initialized to zero. Finally, we can represent the iteration over the frame as a loop that iterates through the elements of the vector **f**. For each element t in **f**, we can update the seed value S as follows:

$S \leftarrow S \oplus t$ $S \leftarrow S \oplus (S << 13)$ $S \leftarrow S \oplus (S >> 17)$ $S \leftarrow S \oplus (S << 5)$ After the loop completes, the final value of the seed S is returned as the generated random number. Note: $\oplus$ denotes the bitwise XOR operation, $<<$ denotes the left-shift operation, and $>>$ denotes the right-shift operation.

The complexity of the algorithm is $\mathcal{O}(c \times k)$, as it performs one iteration for each element in the flattened $frame$. The memory requirements are minimal, as only a single value (the seed) needs to be stored during the execution of the algorithm.

The execution environment is shown in Table 2.1.

Generate Random numbers

In this thesis, a total of ten types of RNGs and CSPRNGs were employed to generate random numbers using a conditioned seed. Specifically, we used `PCG64`, `PCG64XSDM`, `MT19937`, `Philox`, `SFC64`, `ChaCha`, `AESCounter`, `HC128`, `SPECK128`, and `ThreeFry`.

The algorithm for generating random numbers is presented in Algorithm 2, which ensures that the generated data fits the test requirements and is optimized for use with the `pymoo` optimization framework. The *numberOfElements* and *size* is chosen to satisfy the minimum requirement of 1'000'000 numbers for the Universal statistical test in the NIST test suite. Specifically it is is the Universal statistical test that requires at least 1'000'000 numbers for the test to run. The generated numbers were tested using the NIST SP 800-22 test suite with a total of ten distinct PRNGs and CSPRNGs, ensuring a comprehensive evaluation of the randomness of the generated data. Including Line 9 in Algorithm 2 is necessary to meet the test requirements and optimize the generated data for use with `pymoo`.

Algorithm 2 Random number generation

Require: Generator and *seed*

1: $Generator \leftarrow Generator(SeedSequence(seed))$ ▷ Initialize the chosen generator with the *seed*.
2: $size \leftarrow 2012$
3: $arr \leftarrow [\,]$
4: **for** $i \in size$ **do**
5: $\quad arr.extend(Generator.generate(low, high, numberOfElements))$
6: $bin\_arr \leftarrow [\,]$
7: **for** $i \in arr$ **do**
8: $\quad bin\_arr.append(Binary(String(arr[i])))$
9: bin_data ← "".join(bin_arr)
10: **return** *bin_data*

Optimization

To find a near optimal frame size for the entropy extraction, NSGA2 optimization was used. The NSGA2 algorithm and problem parameters are crucial in defining the optimization problem and conducting the optimization process. For an optimization problem the problem parameters define the variables, objectives, and constraints. Mutation, crossover, and sampling parameters determine how the population evolves, while the termination criterion specifies when the optimization process should stop.

NSGA2 Problem Parameter	Value
Sampling	Integer random sampling
n_var	2
n_obj	16
n_ieq_constr	16
xl	[1,1]
xu	[369,169]

Table 2.5: Parameters for the NSGA2 optimization problem.

Table 2.5 provides the parameters for defining the problem for optimization with pymoo. In this case, the NSGA2 algorithm was used to optimize the frame size of the image, which is a discrete variable problem. The

problem has two variables, x and y, representing the frame size of the image. The constraints were defined based on the requirement that the results be more significant than 0.01 and were reflected in the number of inequality constraints, n_ieq_constr. The goal is to maximize all 16 objectives in the NIST test suite, which were included in the problem as objectives to be maximized. Specifically for pymoo which only has a minimize function objectives needed to be multiplied by -1.

It is essential to set a termination criterion for an optimization algorithm to avoid running indefinitely and unnecessarily consuming computational resources. The termination criterion specifies when the algorithm should stop searching for a better solution based on specific criteria, such as a fixed number of iterations, reaching a certain threshold for the objective function, or when the change in the solution has become small enough. All of the default criteria are displayed in table 2.6.

Criteria	definition	Value
xtol	Movement in the design space	1e-8
cvtol	convergence in the constraint	1e-6
ftol	convergence in the objective space	0.0025
period	the number of generations part of the sliding window	30
n_max_gen	max number of generations	1000
n_max_evals	max number of evaluations	100000

Table 2.6: Default NSGA2 termination criteria

In this case, the number of generations was set as the termination criterion for the NSGA2 optimization algorithm. This means the algorithm stopped after generating a fixed number of candidate solutions (three in this case).

Parameters used for the NSGA2 algorithm and problem definition were carefully selected based on the requirements of the problem and the example provided by Pymoo. The termination criterion was set to three generations as a test criterion, and the population size and number of offspring were determined based on the variables included in each generation. The mutation, crossover, and sampling parameters were also determined based on the example provided by Pymoo for discrete variable problems.

To ensure the feasibility and effectiveness of the optimization, the objective

NSGA2 Algorithm Parameter	Value
Population size	3
Number of offspring	3
Number of generations	3

Table 2.7: Parameters for the NSGA2 optimization algorithm.

function values were constrained within the range of the image size produced by the method, which was 169×369. The lower and upper bounds for the variables were determined by the variables xl and xu, respectively. Notably, the optimization required that the first x value in the array had a boundary of 369, while the second x value had a boundary of 169 to enable the optimization to function accurately.

Furthermore, specifically for the random excursion test from the NIST 800-22 test suite, the inequality constraints was defined based on the average results of all subtests, which were required to be greater than 0.01.

NIST Test suite

The NIST test suite is a widely-used benchmark for evaluating the performance of cryptographic hash functions. This thesis implemented the suite using the code base developed by Steve Ang [1]. The code base provides two distinct methods for executing the tests: one involving a graphical user interface (GUI) and the other requiring a command-line interface. To carry out the optimization, the latter method was employed, and only the tests were imported into the objective functions, while other features of the code base were not utilized. This approach enabled the researcher to streamline the optimization process and focus solely on the objectives of interest without any unnecessary overhead. Using the NIST STS in this thesis ensured that the optimized image size would have improved cryptographic properties, thereby contributing to the development of more secure cryptographic systems.

Preliminary tests

To ensure the effectiveness and accuracy of the optimization process, preliminary tests were conducted to identify suitable optimization parameters and assess the 5G spectrum's suitability as an entropy source. The NumPy package's default random number generator, PCG64, was utilized for these tests, with entropy obtained from the 5G spectrum serving as the seed. A total of 2012 images were processed during the tests, with each image producing 2048 random numbers. In total, $4,120,576$ random numbers were generated and evaluated, providing a robust parameter optimization and testing dataset.

Furthermore it follows a few steps:

1. Create numbers for eight (8) different frame sizes. $(10 \times 10, 20 \times 20, \cdots, 80 \times 80)$
2. Same numbers in a separate file for each frame size.
3. Run the NIST test suite on each file, and save results.
4. Compare results from each single test with all other frame sizes.
    - Save the top result in an array
5. Create an array of the total average of all results for each frame size.
6. Return an array of the top results, and the total averages.

Analysis

The optimization results were analyzed using the open-source optimization library `pymoo` [4]. The solutions obtained through the optimization process represent a set of non-dominated solutions, adhering to the principle that no objective can be improved without sacrificing another objective within the solution set. Pareto optimality was employed to evaluate these non-dominated solutions generated by the optimization.

Pareto fronts, which depict the optimal non-dominated solutions, are often utilized to identify the most favourable solutions. However, since the exact Pareto front for the specific problem under investigation is currently unknown, it must be incorporated into the graphs presented in this thesis.

To further explore and evaluate the generated solutions, two methods were employed based on the getting started guide from the Pymoo library:

ASF: Augmented Scalarization Function [4]

- A decomposition method: transforms multi objective problems into many single objective optimization problems.

Pseudo weights. [4]

- A method that calculates the normalized distance to the worst solution regarding each objective.

The augmented ϵ-constraint method is a multi-objective optimization technique that involves fixing one of the objectives as the primary objective and treating the rest as constraints. The ϵ value represents the amount by which the objective function can be violated for the constraint to be satisfied. This method was used to analyze the non-dominated solutions obtained from the NSGA2 algorithm by fixing each of the 16 objectives as the primary objective and treating the rest as constraints.

On the other hand, the Pseudo-weights approach is a multi-objective optimization technique that involves assigning weights to each objective and transforming the problem into a single-objective optimization problem. The transformation is achieved by minimizing a weighted sum of the objectives, where the weights assigned to each objective reflect their relative importance.

This thesis used the Pseudo-weights approach to obtain a single optimized solution that best balances all 16 objectives. Equal weights were assigned to all objectives in this thesis since there was no prior knowledge or indication of any objective being more or less important than the others.

2.2 Design of the thesis

This thesis has been sectioned into four distinct sections. The first section focuses on the data generation process. The second section involves the application of a previously developed implementation to condition the data. In the third section, the conditioned data is input to an optimization algorithm to determine the optimal frame size. Finally, the fourth section presents a thorough analysis of both the data and the results obtained from the optimization tests.

Code for the thesis is in a github repository. [1]

[1] All the code for this thesis can be found in a github repository in Appendix A

Chapter 3

Results

In this chapter, we will present all the results obtained from the optimization process, preliminary experiments, and their comparisons.

3.1 Pre-optimization

Pre-optimization results were conducted as a preliminary evaluation of the optimization problem. The main goal was verify the correctness of the implementation and to explore possible alternative solutions. Moreover, the pre-optimization results helped select appropriate parameters and implementations for the optimization algorithm. These preliminary tests provided valuable insight into the suitability of the 5G spectrum as an entropy source and the ability to generate random numbers from it. The results from the pre-optimization tests served as a baseline for comparing and evaluating the optimization results.

3.1.1 Comparison of NIST test suites

As part of the preliminary test, a test run was conducted using two distinct implementations of the NIST 800-22 test suite. The aim of running this

test was done in order to provide insight into which implementation to use for optimization, hence execution time was the main metric.

The test set consisted of eight *.txt* files consisting of approximately 4.1 million 9 bit numbers. The test numbers were generated using frame sizes in the range of: $10x10 \rightarrow 80x80$.

In list 3.1.1 names of the packages and execution time results are relayed.

- NistRng by Luca Pasqualini - SAILab - University of Siena [22]
  - Execution time: 8 minutes and 54.8 seconds.
- NIST Randomness Testsuit by Steven Kho Ang [1]
  - Execution time: 12 hours, 55 minutes.

Conducting the same operations using these distinct implementations resulted in execution times with a large difference. To ensure that differences in execution times were heavily anchored in the implementations itself all timings related to preliminaries were exempted.

A curiosity for the comparison of these test results is that when the package by Steven Kho Ang ran only a single of the 8 test files fail, but when the package by Luca Pasqualini ran all 8 test files failed.

Preliminary tests

During the preliminary tests, all eight test files from the NIST STS was evaluated using an average over all tests and comparing the performance of different frame sizes for each test. It was aimed to verify the code's correctness and explore possible alternative solutions to the problem. Additionally, these results served as a basis for selecting suitable parameters and implementations for the optimization algorithm. It was found that only the tests with a frame size of 70×70 failed, and specifically, these failures were observed in the Random Excursion Variant test. Table 3.1 summarizes the experimental results. The results on display for each row is: tests number from the NIST SP 800-22 Rev 1a test suite, the frame size with the highest value results (0 - 1) and the value.

NIST test number	Frame Size	Value
1	60×60	0.978
2	10×10	0.950
3	10×10	0.935
4	70×70	0.991
5	30×30	0.831
6	60×60	0.819
7	50×50	0.965
8	70×70	0.989
9	40×40	0.998
10	80×80	0.894
11	60×60	0.919
12	40×40	0.807
13	20×20	0.816
14	10×10	0.462
15	50×50	0.845

Table 3.1: Results for each test from the preliminary test run. Test numbers can be found in the list of the NIST test suite in chapter 1.4.2.

The highest average over all the test for a single frame size was $60x60$ with the average at: 0.6404969122334114.

3.1.2 Shannon's entropy

Shannon's entropy is a measure of uncertainty or randomness of a random variable and was first introduced in information theory. In this thesis, Shannon's entropy was employed in order to evaluate the quality of the proposed entropy source. The efficiency of the entropy source was investigated by testing four distinct RNGs with and without seeding from the 5G-SRNG. For each RNG, 200 numbers were generated in the range of 0 to 32. The frequency distribution of each number was then recorded in tables, and Shannon's entropy was calculated based on the proportion of each number's count to the total count. The maximum value of Shannon's entropy in this setup is $log_2(32) = 5$, which is the upper limit for the expected entropy.

The experimental results for the four different RNGs are presented in Table 3.2. Here it is shown that the Shannon entropy with a seed from the suggested method using 5G is akin to results from known secure random number generators such as windows SystemRandom.

RNG and parameters	Value
Numpy random (No seed)	4.926832205583327
Windows SystemRandom	4.8836097014479884
ChaCha and 5G SRNG seed	4.919273194563815
PCG64 and 5G SRNG seed	4.8494091755251345

Table 3.2: Results from calculating Shannons entropy using different RNGs and seeds.

3.2 Execution times of the Experiments

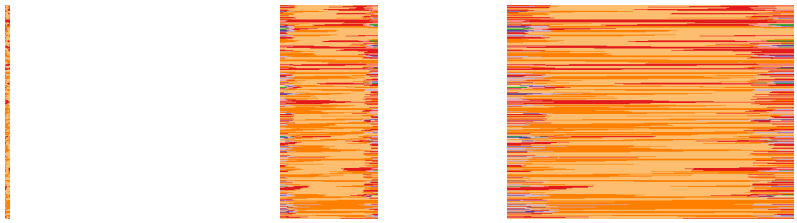
All results for execution times were recorded and are displayed in this section.

3.2.1 Image generation

Creating 2016 images using method provided in the github repository in appendix A took 15 hours 11 minutes and 23.4 seconds. The creation of a single image took approximately 27 seconds.

All images was generated with an aspect of 0.05. Setting the aspect ratio to 0.05 in this context for the matplotlib package in Python translates to 1 unit of y-data being 0.05 times the displayed size of 1 unit of x-data. A initially square pixel would end up being 1 pixel high and approximately 20 pixels wide.

An example of the differences of these three aspects are presented in figure 3.1. In figure 3.1b and figure 3.1c pixels that originally also appear in figure 3.1a have been expanded in a single direction (width). They have been expanded by a factor of 0.05 and a factor of approximately $\frac{1}{62}$. This expanding may alter the amount of entropy an image provides.



(a) Aspect set to square. Image size: $8 \times 369 (W \times H)$

(b) Aspect set to 0.05. Image size: $169 \times 369 (W \times H)$

(c) Aspect set to auto. Image size: $496 \times 369 (W \times H)$

Figure 3.1: Images with different aspects

3.2.2 Entropy generation

Some experiments using a version of the code to get the entropy source was run to create a large number of entropies. Running this on a standard free bundle on google Colab indicated that using matplotlib for reading the images would result in approximately 100 it/s and using cv2 it resulted in 130 it/s.

To generate a dataset for the NIST SP800-90B test suite at least $1,000,000$

samples was required. That would lead to a runtime of over two (2) hours for generating all samples using this approach. Thus tests were conducted to find the limits of the current conditioning algorithm. The tests used twenty (20) distinct images, and ran in two instances, one with images with a minimum frame size (1,1) and the other with a maximum frame size (369,169). The execution times are presented in the list below:

- Execution times are for an experiment with conditioning 20 different images.
- Best case $(c, k) = (1, 1) : 0 - 0.5$ seconds
- Worst case $(c, k) = (369, 169) : 24$ minutes and 12.4 seconds

3.2.3 RNG execution times

The total execution times for NSGA2 optimization with parameters from table 2.7 and table 2.5 are presented in table 3.3. At the left the ten (10) distinct RNGs are named and the execution time for each are presented. The latter five (5) RNGs in table 3.3 are all RNGs based on cryptographic algorithms.

All the non cryptographically based algorithms had a runtime of approximately 5 hours and 15 minutes, whereas the cryptographically based took approximately 7 hours and 30 minutes.

RNG	Execution time (hh:mm:ss)
PCG64	05:16:05
PCG64XSDM	05:14:34
MT19937	05:14:47
Philox	05:22:11
SFC64	05:09:56
ChaCha	07:34:10
AESCounter	07:33:58
HC128	07:50:37
SPECK128	07:29:33
ThreeFry	07:28:41

Table 3.3: Execution times for the optimization using different RNGs.

3.3 NIST SP 800-22 Rev. 1a results

The NIST SP 800-22 Rev 1a test suite and optimization algorithms were employed to evaluate the performance of the RNGs in this thesis. The tests were conducted for a single test (Spectral) and the entire test suite. Execution times for each RNG using the complete test suite are presented in Table 3.3. A total of 60 runs using the optimization algorithm were conducted, and all tests were passed in each run. During the preliminary tests, only one out of the eight tests failed. Therefore, the overall pass rate for the entire NIST test suite was 67/68, indicating high reliability and robustness in the RNGs under consideration.

3.3.1 Comparison

This section presents the combined results of the optimization and preliminary tests, along with comparing the best results obtained using a PRNG and a CSPRNG with those reported in other research papers. Table 3.4 provides an overview of these results. It is worth noting that the actual value of one test for SF64 was 0.99687395, but it is presented as 1.00 in the table.

3.3 NIST SP 800-22 Rev. 1a results

Test No.	DTRNG [12]	BluXor [25]	MPCG [29]	μRNG [16]	RNG IMU [29]	SFC64	Philox*	ThreeFry	SPECK128*
1	0,58	0,51	0,11	0,59	0,02	0,58	0,91	0,84	0,86
2	0,06	0,48	0,04	0,94	0,3	0,81	0,3	0,58	0,98
3	0,59	0,49	0,29	0,27	0,2	0,92	0,73	0,94	0,75
4	0,89	0,44	0,06	0,68	0,78	0,68	0,74	0,87	0,58
5	0,47	0,44	0,15	0,29	0,88	0,8	0,87	0,87	0,94
6	0,85	0,46	0,42	0,59	0,12	1	0,82	0,9	0,76
7	0,59	0,8	0,01	0,65	0,54	0,98	0,8	0,68	0,25
8	0,56	0,55	0,32	0,09	0,85	0,32	0,16	0,33	0,38
9	0,28	0,45	0,24	0,11	0,74	0,94	0,29	0,97	0,63
10	0,66	0,54	0,22	0,36	0,38	0,63	0,79	0,74	0,18
11	0,25	0,5	0,03	0,07	0,68	0,5	0,44	0,68	0,9
12	0,46	-	-	0,27	0,42	0,67	0,78	0,21	0,94
13(F)	0,05	0,55	0,07	0,96	0,88	0,7	0,73	0,55	0,88
13(B)	0,17	-	-	0,71	0,66	0,72	0,83	0,74	0,72
15	0,61	0,28	0,42	0,35	-	0,42	0,46	0,45	0,55
16	0,79	0,4	0,39	0,48	-	0,52	0,4	0,4	0,42
Execution time:	-	-	-	-	-	05:09:56	05:34:04	07:28:41	04:56:35

Table 3.4: Comparison of the two best results from tests run compared to results from various research papers. Results have been rounded to contain 2 significant figures. $*$ uses a modified conditioning method.

Preliminary test vs Optimization

A single run and frame size are selected to compare the performance of preliminary tests against optimization runs. The results with the best total average are chosen for both approaches. The optimization approach yields the highest total average of $\approx 0,699$, while the preliminary tests yield the highest total average of ≈ 0.641.

Table 3.5 presents the results for each of the 15 tests conducted in both the preliminary test and optimization approaches. The results show that, in general, the optimization approach yields higher scores compared to the preliminary test approach. Specifically, the highest scores achieved by the optimization approach are 0.9969 (for test 6), 0.9789 (for test 7), and 0.9393 (for test 9), while the highest scores achieved by the preliminary test approach are 0.9777 (for test 1), 0.9189 (for test 11), and 0.8185 (for test 6). Overall, the optimization approach yields a higher total average of approximately 0.699 compared to the preliminary test approach, which yields a total average of approximately 0.641. According to table 3.5 and comparing the two best results against each other we get the following result:

- Optimization: 8 / 15
- Preliminary test: 7 / 15

NIST test number	Optimization results	Preliminary test results
1	0,57550922	0,97766215
2	0,80669584	0,79911961
3	0,92277767	0,44486678
4	0,68061125	0,78530007
5	0,80478141	0,15645192
6	0,99687395	0,81854581
7	0,97891873	0,48349724
8	0,32135038	0,71522095
9	0,93929601	0,30765173
10	0,63298786	0,45322080
11	0,5046433	0,91889769
12	0,66691474	0,80322810
13	0,70854764	0,69609688
14	0,41786676	0,56532833
15	0,51913499	0,62676566

Table 3.5: Results for each test from the preliminary test run. Test numbers can be found in the list of the NIST test suite in chapter 1.4.2.

3.4 NIST SP 800-90B Results

Table 3.6 displays the results of various NIST SP 800-90B entropy source tests, including IID permutation tests, chi-square tests, LRS tests, and restart tests. These tests evaluate the entropy source's ability to produce truly random and unpredictable data that can be utilized for cryptographic key generation. From the table we can see that all tests have passed, and moreover that the concluding min-entropy is approximately 0.2136.

Test name	Raw entropy	Conditioned entropy
IID Permutation tests	Pass	Pass
Chi square tests	Pass	Pass
LRS test	Pass	Pass
Restart Test	Pass	Pass
Min-Entropy Estimate	0.214516375	0.37823625
Conditioning test h_out	0.2136096580726153692825	

Table 3.6: NIST SP 800-90B Entropy source test results

3.4.1 Comparison of entropy results

A primitive comparison of the results from the NIST SP 800-90B a compression of the data files using CMIX [14] was conducted. Table 3.7 exhibits the results from both a conditioned dataset and a dataset consisting of raw data from the entropy source. Results are presented using min-entropy estimation from both CMIX and NIST SP 800-90B.

Dataset	SP800-90B	CMIX
Conditioned	0.37823625	0.553125
Raw	0.214516375	0.299875

Table 3.7: Comparison of reported min-entropy for NIST SP 800-90B vs CMIX compression tool

3.5 Generated data

A selection of three sequential images that have been created based on the 5G RF spectrum data is displayed in presented in figure 3.2. These three images were chosen to emphasize two noticeable appearances. Where the first appearance is negative in terms of entropy whereas the second is positive. The first appearance are visualized by comparing the left image and the middle image. These two are quite similar in both color and where the colors appear. Second appearance is how dissimilar both the middle and left image are compared to the right image. It is these images that we take a frame of and use in further methods before it is used as a seed to different RNGs.

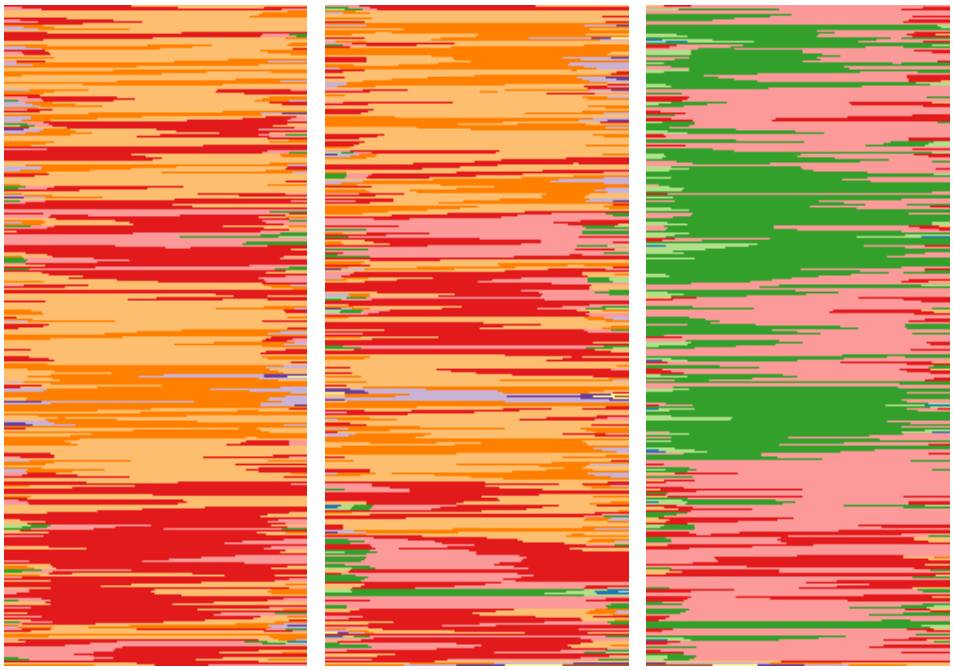


Figure 3.2: Images created from the generated data, these images are what is further used as an entropy source.

3.6 Optimization

In the optimization section, the results for a simple Pareto-front and the results based on ASF and pseudo weights are presented. To evaluate whether an optimal frame size was found, pymoo code was used to analyze all the results. Pymoo provided a set of non-dominated solutions, which cannot be improved in one criterion without worsening another criterion. The documentation from pymoo's getting started guide provided the basis for creating plots and performing multi-criteria decision-making. All results presented here are present in the excel-sheet in the github repository for this thesis in appendix A.

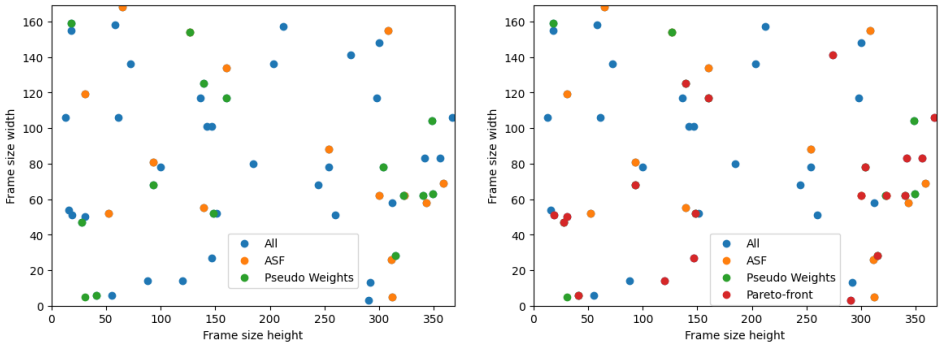
3.6.1 ASF, Pareto-front and pseudo weights

The results depicted in Figure 3.3 illustrate the frame sizes obtained from 60 optimization runs. Figure 3.3a presents three different colors: blue, indicating the frame sizes that were not selected by either ASF or pseudo weights; orange indicating the ones chosen by ASF; and green, indicating the ones specified by pseudo weights. In Figure 3.3b, a red color is added to represent the frame sizes that are the best according to the Pareto-front. Results in figure 3.3 display all of the frame sizes returned from 60 optimizations runs.

The layers in the plot have the following order where a higher number is more to the front of the image. (Lies on top of the other layers):

1. Blue: All frame sizes
2. Orange: ASF
3. Green: Pseudo weights
4. Red: Pareto-front

X values range from $0 \rightarrow 369$ and Y values $0 \rightarrow 169$. These represent the pixels in the images used in optimization.



(a) All solutions with ASF in orange and pseudo Weights in green.

(b) Same plot with Pareto-fronts in red added to see differences.

Figure 3.3: Scatter plots of frame sizes.

NSGA2 no termination criteria

Running a single test from the test suite took approximately 8 minutes with 3 generations and with no termination criteria set it spent 55 generations, 3 hours 35 minutes and 48 seconds.

Chapter 4

Discussion

Results that have been described and displayed in chapter 3 will be reviewed and analyzed in this chapter. Whether the results are good, and whether tests have been performed in good environments will be be a part of the discussion. This chapter means to discuss advantages and disadvantages, and be truthful to whether the implementations done in this thesis are acceptable, or if they have improvements that need to be addressed.

Results will be looked at in mostly the same order as they appear in chapter 3.

4.1 Pre-optimization

For the pre-optimization results, they show that different choices could have been made and different metrics could have presented a change in the conclusion. Thoughts on what is good about the way things are done, and what could have been made better will be described.

4.1.1 Comparison of NIST test suites

The two implementations of NIST's test suite was very different in their implementation. Thus several parameters could have been changed in order to better the execution time for either of the two implementations. That change could have been to optimize the data for either of the tests in some way. As the data are now, they are most likely better suited for Steven Kho Ang's implementation.

Since the execution times between the two implementations had such a large difference, the choice as to which to use was clear. Although the execution times could be different with another setup for the computer and data. It is not very likely that that change would lead to a big enough change in terms of execution times to alter the conclusion.

Failed tests in Luca Pasqualini's implementation, may be because the data was not modified in order to work optimally with this implementation. In the documentation it states that there are utility functions to pack a provided sequence of data in 8-bits. Since the data is 9-bits that may have provided an issue for passing the tests using this implementation. Still the main reason for not choosing this implemenation was the difference in execution times.

4.1.2 Preliminary tests

In table 3.1 all the 8 different frame sizes are represented. A single frame size is represented at most three times and at least once. What can be deduced from these results?

One deduction from the results is that there seems to be no frame size that is optimal. Does that mean that it is impossible to find an approximation to an optimal frame size? Or is the tests executed in a way that makes improbable to find an optimal frame size? These questions will be answered in later sections where results from the optimization have been discussed.

Can the average be used as a metric to find the optimal frame size be better. Is this a metric that has support from the NIST? In SP 800-22 Rev. 1a [20],

they give no such number as to say how one can sum up the results from all tests to compare them to another. Thus it has not been a major inclusion to the discussion, although it may be an interresting aspect to analyze.

In my opinion the average could be used as a metric, but not alone. If it were to be combined with several other metrics like the median, the lowest result, top three and bottom three. Then I think it could provide a simple guide as to compare one result with another. Why then have not all these been included in the thesis? Mostly because to compare the results, other reports or papers would have needed to use the same metrics. Although all these metrics can be gathered from a table of results if they are provided in a paper, this has not been prioritized.

4.1.3 Shannon's entropy

In table 3.2 the results indicate that using an entropy source produced from the 5G RF is good. Which RNG is used may not matter in order to get a higher value. Although choosing a different RNG matters for other metrics, such as whether it is to be used for cryptographic applications or not.

Results for Shannon's entropy and the average of the results from the NIST's test suite are both results that in itself provides little value. Still both of them provide a larger basis for any conclusion to whether using the 5G radio spectrum as an entropy source is valid or not.

4.2 Execution times

Whether providing and registering execution times are important, and what they can be used as a further input to will be discussed in this section.

4.2.1 Images

The documentation for matplotlib states that setting the aspect correct is relevant for images, since it determines whether a pixel is square or not. In

figure 3.1 three different aspects were presented and in both 3.1c and 3.1b pixels are not square. How this would change the entropy result relies in different aspects.

In terms of execution time for the conditioning, it would make a difference with the algorithm presented, because the execution time is O($n \times m$) a change in the image size would also be reflected in higher execution times.

Generating images with the method in this thesis took approximately 27 seconds per image. This process most likely has several ways of being optimized, because in the data generation some images are also generated and this may be done in a faster and more elegant way. Making the image generation process faster is most likely beneficial, at least as long as the quality of generated images are not reduced.

Changing the image generation process would most likely affect the results from the NIST STS, but how it would change is hard to determine. Most likely it would lead to a change in the entropy provided by a single image. Because if a single point of data has been stretched over several pixels the increase in entropy is most likely equal to zero since no new data has been introduced. The only difference lies in that a single point of data in the x direction now exists in approximately 20 more pixels. Hence a change in the aspect would be an interesting subject to alter and verify if it is a positive modification. In order to verify this tests run with a new set of images would be necessary.

4.2.2 Conditioning

Based on the results from chapter 3.2.2, there are improvements that could be done in terms of execution time. The execution time of the algorithm could be changed in order to get a execution time that is closer to O(n) instead of O($n \times m$). That would lead to a faster entropy extraction and reduce time spent waiting for a new entropy source.

What then would a change in the algorithm to reduce the execution time do to the entropy? If it would lead to the entropy being not good enough to pass the tests, then a decrease in execution times would not be beneficial. Is it possible that a change that reduces execution times also increases or

at least not reduces the entropy such that it is not useful anymore? If we imagine a Rubik's cube that is completed as a start, and that there are no known algorithms for solving it. Whether ten or a thousand permutations are done to it before you start solving it is not essential to the entropy.

All this would only be possible if the starting point is actually a known position, which is unlikely when using the 5G RF spectrum because these images can be extracted with very low intervals (40 ms for this thesis), and each image contains slight variations. In other terms the entropy of these images are high.

So then a possible improvement, that needs to be examined is a change to how the images are conditioned.

New variant of the algorithm

Based on the image generation results, a faster variant of algorithm 1 was proposed and tested in order to run the NIST SP 800-90B test suite. The changes to the algorithm was to set a explicit end to the for loop in line 6. Now the start and end of the for loop is based on a modulo of the shape of the current spectrogram. In this way it reduces the amount of iterations in the for-loop to a maximum of 50 iterations. This significantly reduces the execution time.

This was done to reduce the execution time even further and determine whether less conditioning would reduce the entropy gained from conditioning.

Changes done was mostly used for generating the data from the entropy source to test in the NIST SP 800-90B. To verify whether the conditioning done here was sufficient the NIST STS was also ran on data using this conditioning method. These results from these data are included in the results section and will be discussed in the optimization section.

Algorithm 3 shows the pseudo code for the variant. Lines 6-8 are all the lines that reflect the changes. All other lines are as they were in the original.

Algorithm 3 Variant of proposed 5G-SRNG

Require: $5G\ Spectrogram : \mathcal{D} \in \mathbb{R}^{m\,x\,n}, framesize : c,k$

$x_{start}, y_{start} \leftarrow random(0, m-c), random(0, n-k)$
$x_{end} = min(x_{start} + c)$
$x_{end} = min(y_{start} + k)$
$frame \leftarrow \mathcal{D}[x_{start} : x_{end},\ y_{start} : y_{end}].flatten()$
$seed \leftarrow 0$ ▷ Initialize *seed* with 32-bit float representation of 0
$loop_{start} = (frame \mod 3) + 1$
$loop_{end} = frame \mod 51$
for $t \in (loop_{start}, loop_{end})$ **do**
    $seed \leftarrow seed \oplus t$ ▷ XOR operator
    $seed \leftarrow seed \oplus seed << 13$ ▷ Shift-left the previous *seed* value 13 bits, then perform XOR operator
    $seed \leftarrow seed \oplus seed >> 17$ ▷ Right-right the previous *seed* value 17 bits, then perform XOR operator
    $seed \leftarrow seed \oplus seed << 5$ ▷ Shift-left the previous *seed* value 5 bits, then perform XOR operator
end for
return *seed*

4.2.3 RNGs

Table 3.3 gives all the execution times with different RNGs. In this table the difference between a PRNG and a cryptographically based PRNG is not more than at most close to three hours. This difference gives a good basis for choosing which RNG to use since both take hours to complete. As will be presented in chapter 4.7 whether this takes five or eight hours is most likely not an issue.

4.3 NIST SP 800-90B

Notably, all tests in table 3.6 have been successfully passed, indicating a high quality of the generated data.

Moreover, the table also presents the min-entropy estimate, which quantifies the amount of entropy that can be extracted from the data. The

conditioning process has increased the conditioned entropy estimate, which is desirable for cryptographic applications. The raw entropy estimate is 0.2145 bits per bit, whereas the conditioned entropy estimate is 0.3782 bits per bit.

Furthermore, the conditioning test *h_out* is included in the table, providing insight into the conditioning process's effectiveness. With 0.2136 bits per bit, the conditioning process has not significantly decreased the data's entropy.

These results demonstrate that the entropy source generates high-quality random and unpredictable data suitable for cryptographic applications. However, the author notes that further testing is necessary to assess the optimization method's performance on a real system-generated dataset.

4.3.1 Comparison

Based on the data in table 3.7 we can see that the min-entropy estimate in SP800-90B gives a more conservative result than the results based on the compression. In some online forums and websites the results from NIST SP 800-90B entropy estimation results have been questioned, and it has been stated that they give very conservative results. Based on the data presented here this also seems to confirm this, but whether this is a better way to determine the entropy is not likely as the results from NISTs tests rely on a larger amount of data and tests.

4.4 Optimization

The presented thesis evaluated the performance of several RNGs using the complete NIST test suite and a single test (Spectral). The results demonstrated the high reliability and robustness of the RNGs. These results suggest that the evaluated RNGs are suitable for generating cryptographic keys with high entropy and can withstand attacks by cryptanalytic techniques. The results indicate that all RNGs passed the NIST test suite with a high overall pass rate of 67/68. The optimization algorithm further improved the

performance of the RNGs, as evidenced by the reduced execution times and improved pass rates for the NIST test suite. Moreover, the results obtained using the cryptographically based PRNG were better than those obtained using the PRNG, which is expected due to the cryptographic properties.

Comparing the results with those reported in other research papers, we observe that the optimized SF64 and the optimized RNGs outperformed the RNGs proposed in other studies. This indicates that the optimization method can be used to improve the performance of existing RNGs, including those with already high quality. The results also suggest that the proposed optimization method is effective and efficient in generating high-quality random numbers for cryptographic applications.

Comparing results generated using the shorter conditioning component with the other papers and the original conditioning. These results outperform those from other studies, but they do not perform better than the original version of the conditioning. Since the optimization method has been applied on these results as well, it provides a good comparison to whether the conditioning done is good enough. As they all pass the tests with top results.

Results from chapter 3.1.1 present the timing it took to complete run of the NIST test suite. If this timing is somewhat similar to what is being spent during an optimization we can derive that from a total of 5 hours for an optimization, a total of 108 minutes is spent running the test suite. That gives a proportion of $108/300 = 0.36 = 36\%$. These numbers are all approximated. As such a large amount of time is spent only to get the values used in the optimization, a large amount of function evaluations would be more time consuming. All this time spent leads to a lower number of evaluations, and therefore most likely a worse result. From the test with no termination criteria set it took 26 times longer. If we are to use these numbers for the complete test set, it would then take $5hours * 26 = 130hours \approx 5.5days$. If this optimization were to be run only once in a device in production, it could be argued that this is an acceptable time of completion. These difficulties all needs to be addressed in the theoretical solution to how this could be used in a live system.

For test done in this thesis, a seed parameter has not been set. For the purpose of reproducibility of the results, that could have been done. The

reason it has not been done is to generate more diverse results as a basis for seeing if the methods presented are indeed viable for using in a live system.

4.4.1 NIST SP800-22 Rev. 1a valid?

In a recent decision to revise NIST SP 800-22 Rev. 1a the NIST state that they want to clarify and reject the use of these tests to assess RNGs. [19] Does this affect the results and usage of these tests in this paper? It may, but as this paper not necessarily proposes a new RNG, but an entropy source and how to find the best frame size for using that entropy source. The results from the test still hold and they are all useful as to gaining knowledge about whether the entropy source are good or not. Still it does raise the question on how to increase the validity of results in this thesis, but it is also compared and based on the current usage of this test suite. Which is included in several recent relevant research papers, and thus it may provide starting point for further works on this.

4.4.2 NSGA2 parameters

The settings from table 2.7 are all low values, but what significance does this have for tests run in with the specific problem in this thesis? Having the parameters set to this gave a indication based on the results that even though we only ran twelve evaluations all evaluations passed. What this can tell us is that even though a higher amount of evaluations would be preferred to find the all around best solution. Using as few evaluations as done still gave results that passed all tests in the NIST test suite.

What would it take to find the optimal values for parameters to the problem in NSGA2, and is there an easy way to figure out what would be the optimal parameters for these values for the usage and objectives in this thesis? Based on the no free lunch theorem, finding the best parameters is simply put not easy. It is something that requires an effort, and if one are to rely on the results from the parameters used. They all pass the tests, and the results are better than those of other related works.

In general for population size, number of offspring and number of genera-

tions. These things can be said:

- Set to too large values: The epoch time would be long, this restricts the chances of an individual to explore it's neighborhood.
- Set too small values: The coverage of the values looked at is bad.

4.4.3 ASF, Pareto-front and Pseudo weights

From the results on these numbers, it seems that there is a similarity between the pareto-front and pseudo weights. That is from the observable results these two often coincide with which is the best result. Since part of what NSGA2 does is approximating the pareto-front, and pseudo weights with a convex pareto-front are an indicator to a solution. Since the results from observing the data and the pseudo weights calculation coincide it seems like the pareto-front for this problem is convex based on this information.

Apart from the pareto-front and pseudo weights agreeing on which result is the best, other usages of this data have not been found. ASF results sometimes coincide with the pareto-front but it is rare that all three coincide in the gathered data.

Looking at figure 3.3, there are more green dots that are overlaid by the red dots of the pareto-front. Still not all the dots from ASF and pseudo weights coincide with the dots from pareto.

4.4.4 Optimal frame size

In order to figure out what the optimal frame size for the entropy source is, looking at both the results from all results and their corresponding frame sizes, as well as the comparison of simple test versus the optimization results will be observed.

To start with the simple tests against results from the optimization in table 3.5. These observations suggest that the size of the frame significantly impacts the performance of the NIST tests, and larger frame sizes generally

lead to better results. It also gives sense of that there are very likely many frame sizes that produce good results. That is to the extent that in all the 68 runs in total (optimization and simple tests) only a single test from the simple test did not pass. These results are quite promising in concern of using the 5G radio frequency spectrum as an entropy source.

Looking at the results from the plot in figure 3.3 we see no apparent clustering to a range of frame sizes. Instead we see that the frame sizes are scattered all over the plot. This does indicate that finding an optimal frame size is not straightforward. It may be that with many more evaluations in an optimization the results could have been more clustered. Although as it is there are no clear tendency towards a optimal frame size with the current data and parameters for the optimization.

Why then does the data not lean towards a range in which an optimal frame size exist? It may be because when the data has been created there may be differences in the noise which causes the images to appear random. If there is no common link between the images, finding an optimal frame size would also be difficult. As what is optimal for one image is likely not optimal for another, because they are different. In addition the conditioning of the data from the images, may also cause two almost equal input data to be unequal such as the avalanche effect in a hash function. Although since only 9 bits are used, the conditioning is not collision resistant. Meaning that two different values may end up with the same en value.

So an optimal frame size for the data in this thesis may not have been found, but could it have been found with a physical setup where there may have been less fluctuations in the spectrum signal? It may be possible, but I think that a rather more important question is whether finding a optimal frame size is important or not? From the data we do see that there are a multitude of frame sizes that have been reported as the optimal, and out of all the ones gathered using the optimization all passes the tests. So can the optimization instead be used to weed out the bad frame sizes, and return a set of good frame sizes instead? This may be a promising use of the optimization for a device that is stationary that is in the need of finding a good entropy source.

4.5 Energy consumption

The energy consumption on a device using the proposed method for generating random bits, would as opposed to most other TRNG devices, not require an additional component. In addition the complexity of the code used for RBG using this method is low. These are main metric when looking at the energy consumption of a device using the 5G RF spectrum to generate random bits. As stated in the related works there does exist other methods that also use already in-built hardware to generate random bits. The difference from most of them is that they are not using such a general hardware as the network connection.

4.6 Generated data

The images in figure 3.2 display that there may be not that large a difference between images, but still on the ones that at a first glance look most similar. It is hard to find a frame that could provide a similarly promising entropy. On the other hand the image to the right that contains colors from a different spectrum, displays the entropy of the images. Based on this the data from the 5G RF spectrum seem to have a high entropy.

4.7 Theoretical solution

Based on the promising results obtained from the optimization process and preliminary experiments conducted on the NIST test suite, the aim is to investigate the effectiveness of this approach on a dataset generated by a real system. Specifically, the idea is to use the optimization method to determine the optimal frame size for an entropy source implemented on a practical device, such as a hardware TRNG, and evaluate the quality of the generated entropy using established metrics and statistical tests. This investigation will allow to assess the practicality and effectiveness of this approach in real-world applications and provide insights into the optimization of entropy sources for cryptographic purposes.

4.7 Theoretical solution

To make the solutions in this thesis more tangible a theoretical version of how it can be used in physical system will be described.

Imagine that we have a some 5G enabled IoT device, let's say a temperature sensor. This temperature sensor is to monitor some important data for a company that does research in gardening. In order to ensure that this data is not compromised they want to find a good solution for entropy that is energy efficient, fast and provides high entropy. To get this they install a version of software described in this thesis. After it has been installed, it takes roughly 7-8 hours before the software is completely ready to provide seeds with high entropy. After this initialization period, every 40 ms a new entropy can be created, and good random numbers can be generated.

The gathering of an entropy follows the same steps as described in figure 2.1.

1. Spectrum extraction: takes approximately 40 ms. Where it takes a snapshot of the 5G RF at that moment.
2. Image generation
3. Conditioning entropy source
4. entropy source ready for use

Ultimately, the aim is to enhance the security and trustworthiness of RBGs by ensuring that they generate high-quality and unpredictable random bits.

Chapter 5

Conclusion

This thesis introduces a new method for generating random numbers in 5G wireless communication systems. It utilizes the radio frequency spectrum as an entropy source and extracts entropy from the variability of channel state information. The result is high-quality random numbers with a high level of entropy. Simulation testing of the proposed method in a 5G system model showed that it outperforms traditional RNGs regarding energy consumption and computational complexity.

The proposed method represents a significant advancement in the random number generation field and can potentially address the challenge of unreliable random number generation in 5G networks. Moreover, the proposed method's performance could be compared with other state-of-the-art RNG methods to validate its effectiveness further. Other potential areas of future research include the development of hardware implementations of the proposed method and exploring how the proposed method can be integrated into existing cryptographic protocols to enhance their security.

Using an optimization algorithm to find a near optimal solution to the frame size used as an entropy source also looks promising since all results pass the tests in this thesis.

Future research could focus on optimizing 5G as an entropy source and enhancing the results from NIST SP 800-90B tests to improve the proposed

method's security and reliability. There is potential to investigate methods for reducing the impact of channel impairments and noise on the randomness of generated numbers and exploring ways to mitigate potential biases. Evaluating the proposed method in different environments could also determine its effectiveness in various wireless communication systems.

Bibliography

[1] Steven Kho Ang. Nist randomness testsuit. https://github.com/stevenang/randomness_testsuite.

[2] Uzma Ansari, Akhilesh Kumar Chaudhary, and Sudhanshu Verma. True random number generator (trng) using sensors for low cost iot applications. In *2022 International Conference on Communication, Computing and Internet of Things (IC3IoT)*, pages 1–6, 2022.

[3] Boaz Barak and Shai Halevi. A model and architecture for pseudo-random generation with applications to /dev/random. Cryptology ePrint Archive, Paper 2005/029, 2005. `https://eprint.iacr.org/2005/029`.

[4] J. Blank and K. Deb. pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509, 2020.

[5] Ferhat Ozgur Catak, Evren Catak, and Ogerta Elezaj. 5G-SRNG: 5G Spectrogram-based Random Number Generation for Devices with Low Entropy Sources. *arXiv e-prints*, page arXiv:2304.09591, April 2023.

[6] dfranke. How i hacked hacker news (with arc security advisory). https://news.ycombinator.com/item?id=639976, 2009.

[7] John Kelsey (NIST) Elaine Barker (NIST). Recommendation for random number generation using deterministic random bit generators. Technical Report SP 800-90A Rev. 1, U.S. Department of Commerce, Washington, D.C., 2015.

[8] Naoki Fujieda, Hitomi Kishibe, and Shuichi Ichikawa. A light-weight implementation of latch-based true random number generator. In *2019*

15th International Wireless Communications & Mobile Computing Conference (IWCMC), pages 901–906, 2019.

[9] John Kelsey; Bruce Schneier; David Wagner; Chris Hall. Cryptanalytic attacks on pseudorandom number generators. Technical report, Counterpane Systems, University of California Berkeley, 2017.

[10] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[11] Md. K. Hossain, Mohammad I. Rashid, Mohammad R. Haider, and Md. T. Rahman. Randomized pulse-based encoding for secure wireless data communications. In *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 289–292, 2020.

[12] Khaled Humood, Baker Mohammad, and Heba Abunahla. Dtrng: Low cost and robust true random number generator using dram weak write scheme. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[13] kartverket. Vilkår for bruk av kartverkets opne data. https://www.kartverket.no/api-og-data/vilkar-for-bruk, 2021.

[14] Byron Knoll. Cmix. https://www.byronknoll.com/cmix.html.

[15] Nicoleta Cucu Laurenciu and Sorin D. Cotofana. Low cost and energy, thermal noise driven, probability modulated random number generator. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2724–2727, 2015.

[16] Sanu K. Mathew, David Johnston, Sudhir Satpathy, Vikram Suresh, Paul Newman, Mark A. Anders, Himanshu Kaul, Amit Agarwal, Steven K. Hsu, Gregory Chen, and Ram K. Krishnamurthy. μ rng: A 300–950 mv, 323 gbps/w all-digital full-entropy true random number generator in 14 nm finfet cmos. *IEEE Journal of Solid-State Circuits*, 51(7):1695–1704, 2016.

[17] Inc Matlab, The MathWorks. Spectrum sensing with deep learning to identify 5g and lte signals. https://se.mathworks.com/help/comm/ug/spectrum-sensing-with-deep-learning-to-identify-5g-and-lte-signals.html, 1994-2023.

[18] Merriam-Webster. "pseudo.". https://www.merriam-webster.com/dictionary/pseudo, 2022. Accessed 28 Feb. 2023.

[19] NIST. Decision to revise nist sp 800-22 rev. 1a. https://csrc.nist.gov/News/2022/decision-to-revise-nist-sp-800-22-rev-1a, 2022.

[20] Andrew Rukhin (NIST); Juan Soto (NIST); James Nechvatal (NIST); Miles Smid (NIST); Elaine Barker (NIST); Stefan Leigh (NIST); Mark Levenson (NIST); Mark Vangel (NIST); David Banks (NIST); N. Heckert (NIST); James Dray (NIST); San Vo (NIST); Lawrence Bassham (NIST). A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical Report SP 800-22 Rev. 1a, U.S. Department of Commerce, Washington, D.C., 2010.

[21] Meltem Sönmez Turan (NIST); Elaine Barker (NIST); John Kelsey (NIST); Kerry McKay (NIST); Mary Baish (NSA); Michael Boyle (NSA). Recommendation for the entropy sources used for random bit generation. Technical Report SP 800-90B, U.S. Department of Commerce, Washington, D.C., 2018.

[22] Luca Pasqualini SAILab University of Siena. Nistrng. https://github.com/InsaneMonster/NistRng.

[23] Elaine Barker National Institute of Standards and Technology. Security requirements for cryptographic modules. Technical Report SP 800-57 Part 1 Rev. 5, U.S. Department of Commerce, Washington, D.C., 2020.

[24] Jungmin Park, Seongjoon Cho, Taejin Lim, Swarup Bhunia, and Mark Tehranipoor. Scr-qrng: Side-channel resistant design using quantum random number generator. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019.

[25] Bikram Paul, Gaurav Trivedi, Pidanič Jan, and Zdeněk Němec. Efficient prng design and implementation for various high throughput cryptographic and low power security applications. In *2019 29th International Conference Radioelektronika (RADIOELEKTRONIKA)*, pages 1–6, 2019.

[26] Ionuț Rădoi, Lidia Dobrescu, and Cosmin Rusea. Random number generation in hardware reconfigurable wireless sensor nodes. In *2023 13th International Symposium on Advanced Topics in Electrical Engineering (ATEE)*, pages 1–4, 2023.

[27] Kevin Sheppard. Additional bit generators and distribution for numpy's generator. https://bashtage.github.io/randomgen/index.html, 2018.

[28] Mansi Sheth. Cryptographically secure pseudo-random number generator (csprng). https://www.veracode.com/blog/research/cryptographically-secure-pseudo-random-number-generator-csprng, 2017.

[29] Yingnan Sun and Benny Lo. Random number generation using inertial measurement unit signals for on-body iot devices. In *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, pages 1–9, 2018.

[30] P. V. Vezeteu, I. I. Popescu, and D. I. Nastac. The generation of random numbers using the quantum tunnel effect in transistors. In *2019 IEEE 25th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, pages 379–382, 2019.

[31] Wikipedia. Next-bit test. https://en.wikipedia.org/wiki/Next-bit_test, 2022.

[32] Hongjun Wu. The stream cipher hc-128. Technical report, Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium, 2008.

[33] Tong Xu, Deyun Gao, Ping Dong, Chuan Heng Foh, Hongke Zhang, and Victor C. M. Leung. Improving the security of wireless communications on high-speed trains by efficient authentication in scn-r. *IEEE Transactions on Vehicular Technology*, 68(8):7283–7295, 2019.

[34] Andrew C. Yao. Theory and applications of trapdoor functions. Technical report, University of California, Berkeley, California 94720, 1982.

[35] Ferhat Özgur Catak. Ferhat ozgur catak. https://www.ozgurcatak.net/, 2022.

Appendix A

Program Listings

This appendix contains a link to a github repository where all the code for the thesis can be found:

https://github.com/Oddvar-N-O/5G-RF-Spectrum-based-Cryptographic-Pseudo-Random-Number-Generation-for-IoT-Security

Appendix B

Experimental results

This appendix contains all the results from test run using the NSGA2 algorithm, as well as a comparison of them with other paper

RNG function: >	PCG64			
Frame Size:	19x51	244x68	16x54	147x101
Frequency (Monobit)	0,90224241	0,98165872	0,08562330	0,81485801
Frequency Test within a Block	0,42793168	0,09267947	0,16013742	0,08394309
Runs	0,79478180	0,14651681	0,19675123	0,88561337
Longest Run of Ones in a Block	0,62072306	0,51991420	0,09470547	0,58033974
Binary Matrix Rank	0,21471682	0,47158560	0,19209492	0,79751453
Discrete Fourier Transform (Spectral)	0,83479622	0,53907632	0,61835756	0,59539815
Nonoverlapping Template Matching	0,13021566	0,72718812	0,64875986	0,87134967
Overlapping Template Matching	0,29464899	0,03568127	0,16565386	0,04836684
Maurer's "Universal Statistical"	0,90660581	0,77583802	0,21031280	0,28910408
Linear Complexity	0,12515734	0,02890599	0,15752195	0,03396834
Serial	0,33603040	0,49978380	0,77875709	0,59474656
Approximate Entropy	0,67443508	0,06507357	0,39306963	0,34529449
Cumulative Sums (Forward)	0,49414989	0,69905751	0,04443542	0,83554867
Cumulative Sums (Backward)	0,40152243	0,67752069	0,12150867	0,96946949
Random Excursions	0,20817368	0,54854788	0,25403466	0,32978018
Random Excursions Variant	0,42946957	0,74723993	0,43836264	0,41239949
Total Average:	**0,487225**	**0,472267**	**0,285005**	**0,530481**

RNG function: >	PCG64			
Frame Size:	359x69	349x63	342x83	300x62
Frequency (Monobit)	0,35934149	0,82277089	0,42198211	0,38520668
Frequency Test within a Block	0,82490063	0,65188204	0,34916247	0,26444538
Runs	0,54098255	0,49765799	0,83770629	0,41000643
Longest Run of Ones in a Block	0,61808815	0,80473646	0,44186487	0,81870238
Binary Matrix Rank	0,91525797	0,48728657	0,65465297	0,24095752
Discrete Fourier Transform (Spectral)	0,35125115	0,62154718	0,30593711	0,63652552
Nonoverlapping Template Matching	0,49854035	0,79511296	0,2189735	0,17592645
Overlapping Template Matching	0,72447952	0,02015279	0,15433382	0,15471807
Maurer's "Universal Statistical"	0,26792108	0,09865896	0,15627966	0,77344787
Linear Complexity	0,20203581	0,88113936	0,96539416	0,20698018
Serial	0,26964824	0,80202176	0,27279499	0,50605472
Approximate Entropy	0,39424894	0,11320535	0,2052516	0,57703239
Cumulative Sums (Forward)	0,38922256	0,88905115	0,48953006	0,68596954
Cumulative Sums (Backward)	0,50965495	0,6895078	0,67782758	0,16332497
Random Excursions	0,48797744	0,50420175	0,68091082	0,43542758
Random Excursions Variant	0,11865888	0,47422759	0,4330066	0,35339642
Total Average:	**0,467013**	**0,572073**	**0,454101**	**0,424258**
Execution time		**05:16:05**		
Best regarding to:	ASF	Pseudo		ASF

RNG function: >	PCG64			
Frame Size:	52x52	13x106	28x47	254x88
Frequency (Monobit)	0,17053174	0,19195646	0,83044574	0,41086975
Frequency Test within a Block	0,84722223	0,61389309	0,85889897	0,83375626
Runs	0,93144568	0,96040904	0,29298598	0,22652687

Figure B.1: All the tables of results

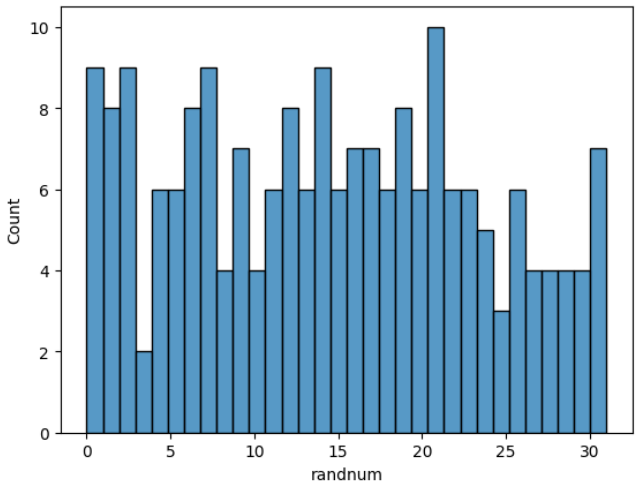


Figure B.2: Base for Shannons entropy for numpy random

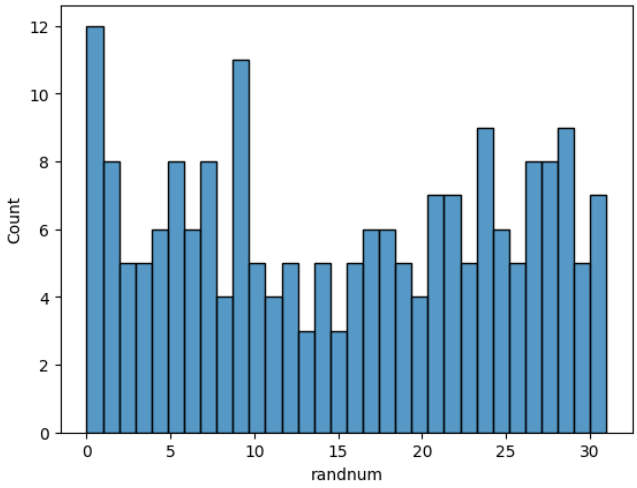


Figure B.3: Base for Shannons entropy for SysRandom

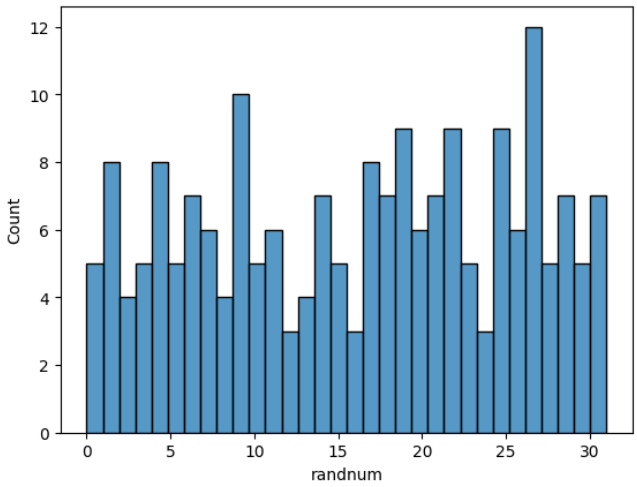


Figure B.4: Base for Shannons entropy for 5GSRNG with ChaCha

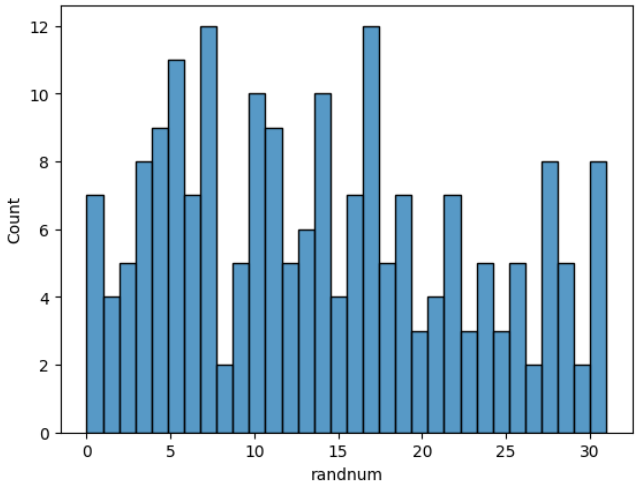


Figure B.5: Base for Shannons entropy for 5GSRNG with PCG64