Hardware Based True Random Number Generator (TRNG)

By

Maj Usama Zia

Maj Yaruq Shah

Maj Fahad Salahudin

Capt Sharjeel Mansoor

Supervised by:

Col Faisal Akram

Submitted to the faculty of Department of Electrical Engineering,
Military College of Signals, National University of Sciences and Technology, Islamabad,
in partial fulfillment for the requirements of B.E Degree in Electrical (Telecom) Engineering.

June 2023

In the name of ALLAH, the Most benevolent, the Most Courteous

CERTIFICATE OF CORRECTNESS AND APPROVAL

This is to officially state that the thesis work contained in this report

"Hardware based True Random Number Generator"

is carried out by

Maj Usama Zia

Maj Yaruq Shah

Maj Fahad Salahudin

Capt Sharjeel Mansoor

under my supervision and that in my judgement, it is fully ample, in scope and excellence, for the degree of Bachelor of Electrical (Telecom.) Engineering in Military College of Signals, National University of Sciences and Technology (NUST), Islamabad.

Approved by

Supervisor

Col Faisal

Department of EE, MCS

Date: _____

DECLARATION OF ORIGINALITY

We hereby declare that no portion of work presented in this thesis has been submitted in support of another award or qualification in either this institute or anywhere else.

ACKNOWLEDGEMENTS

Allah Subhan'Wa'Tala is the sole guidance in all domains.

Our parents, colleagues and most of all supervisor, ____Col Faisal __ without your guidance.

The group members, who through all adversities worked steadfastly.

Plagiarism Certificate (Turnitin Report)

This thesis has ____ similarity index. Turnitin report endorsed by Supervisor is attached.

_____

Student 1 Name

NUST Serial no

_____

Student 2 Name

NUST Serial no

_____

Student 3 Name

NUST Serial no

_____

Student 4 Name

NUST Serial no

_____

Signature of Supervisor

ABSTRACT

Random numbers are crucial in various fields, like generating cipher keys in cryptography, ensuring fairness and randomness in online gaming platforms, modelling random events and procsses accurately in simulations, and producing random samples for scientific experiments. In recent years, true random number generators (TRNGs) have become increasingly popular due to their ability to produce unpredictable and unbiased random numbers. This project aims to design and implement a TRNG using a combination of a random bit generator version 3 (RBG3) and a deterministic random bit generator (DRBG) based on the Advanced Encryption Standard (AES) 256 algorithm, in accordance with the National Institute of Standards and Technology (NIST) guidelines. RBG3 is a type of TRNG that utilizes the principle of entropy, which refers to the unpredictability of a random event. Entropy is generated by measuring physical phenomena, such as ring oscillator and zener diode breakdown, subsequently converting the analog values into random bits using a microcontroller. DRBGs, on the other hand, generate random numbers based on a deterministic algorithm and a secret seed value. The evaluation of the

TRNG's performance using various health tests will provide insights into its strengths and limitations which will allow further improvements to be made before being deployed.

Table of Contents

Chapter 1: Introduction

Modern cryptographic systems and security protocols rely heavily on random number generation as the cornerstone for guaranteeing the confidentiality, integrity, and autheticity of sensitive data. The National Institute of Standards and Technology (NIST) has published guidelines and standards to ensure that random number generation for cryptographic applications satisfies quality and security requirements. Truly random numbers can now be generated in digital systems with high reliability using hardware-based random number generators (RNGs). These RNGs use physical events to produce high-entropy random bits, making them appropriate for a range of security-critical applications.

The specifications for True Random Number Generators (TRNGs) and Deterministic Random Bit Generators (DRBGs) are provided in the NIST documents NIST SP 800-90A, NIST SP 800-90B and NIST SP 800-90C respectively. While DRBGs are deterministic algorithms that create random bits based on an initial seed and a cryptographic method, TRNGs generate random bits using physical phenomena such as thermal noise, jitter, and other analogue qualities. To guarantee the quality, unpredictability, and cryptographic security of hardware-based RNGs, these publications offer standards for their design, implementation, and testing.

It is impossible to exaggerate the significance of hardware-based RNGs in contemporary technologies. They are crucial for safeguarding a number of applications, including secure key generation, secure communication, secure storage, and secure authentication. For the system to be secure overall and to safeguard sensitive data from hostile attacks, the quality and unpredictable nature of the produced random numbers are essential. Therefore, in contemporary cryptographic systems and security protocols, the creation of hardware-based RNGs that adhere to the NIST

criteria is crucial. In this research work, we present a hardware-based TRNG based on RBG3 and a DRBG based on AES256, both of which adhere to the NIST requirements. We test both systems for compliance and perform statistical tests to assess their performance and security.

1.1 Background and Motivation

A crucial necessity for many disciplines, including encryption, simulations, gaming, and statistical analysis, is the creation of random numbers. To guarantee the secrecy, integrity, and validity of sensitive data, cryptographic methods employ random integers as keys, nonces, initialization vectors, or seeds. Random numbers that are inaccurate or predictable can create security flaws that can endanger sensitive data and important systems. As a result, reliable random number generation is crucial to maintaining the security and dependability of contemporary digital systems.

Pseudorandom number generators (PRNGs), which are commonly used to generate random numbers on computers, are based on deterministic algorithms and do not produce truly random values. They produce numbers that seem random but are actually the result of a predictable, deterministic process if the algorithm and seed are understood. Since the quality and unpredictable nature of random numbers are crucial in cryptographic applications, this makes PRNGs vulnerable to attacks.

Hardware-based True Random Number Generators (TRNGs), which rely on physical events to produce random bits, were created in response to the requirement for really random numbers. TRNGs produce really random bits with high entropy by taking advantage of the inherent unpredictability of physical processes like thermal noise, jitter, and radioactive decay. In comparison to PRNGs, hardware-based TRNGs have a number of benefits, such as more security, greater randomness, and independence from the initial seed or algorithm.

The need for safe and dependable systems is driving the use of hardware-based TRNGs in contemporary technologies. The demand for safe random number generation has increased significantly as a result of the spread of connected devices, cloud computing, and data-intensive applications. TRNGs are essential for safeguarding a variety of applications, including secure key

generation, secure communication, secure storage, and secure authentication. They give cryptographic methods a solid foundation and guarantee the confidentiality, integrity, and validity of sensitive data.

Additionally, NIST standards compliance is necessary to guarantee the reliability and security of random number generation in cryptographic applications. The design, implementation, and testing of TRNGs and DRBGs must adhere to strict NIST standards, such as those in NIST SP 800-90B and NIST SP 800-90A, to guarantee their quality, unpredictability, and cryptographic security. To meet the demanding security criteria of contemporary digital systems and guarantee the general security and dependability of cryptographic applications, TRNGs that adhere tthese standards are required.

1.2 Problem Statement

Traditional RBGs are not suited for contemporary security requirements since they are attackable. The Advanced Encryption Standard (AES)-based Deterministic Random Bit Generators (DRBGs) have been suggested as a more secure alternative. The computational burden and reliance on the security of the underlying block cypher are two drawbacks of DRBGs.

The purpose of this thesis is to develop and build an RBG3 with DRBG AES256 that overcomes the drawbacks of conventional RBGs and DRBGs while still offering high-quality random numbers for cryptographic applications. The DRBG AES256 will be assessed for its randomness, security, and performance, and the RBG3 will be created to adhere to the NIST SP 800-90B and SP 800-22 requirements. Additionally, the thesis will examine the effects of the suggested RBG3 with DRBG AES256 on various cryptographic applications and offer suggestions for additional study in this field. Therefore, the goal of this thesis is to create an RBG3 with DRBG AES256 that is more

efficient and safe and can provide high-quality random numbers for cryptographic applications while still meeting NIST SP 800-90B and SP 800-22 criteria.

1.3 Proposed Solution

The major goal of our proposed solution is to make a hardware based TRNG using multiple ring oscillators and zener diode break down (physical phenomenon) as entropy sources, digitize the analogue signals using a microcontroller and apply health tests on raw data. The digitized output will be conditioned to extract entropy which will be seeded to a determnistic random number generator and and subsequently coupled with its output.

1.4 Research objectives

A. To review the existing literature on RBGs and DRBGs, including their limitations, strengths, and weaknesses, to identify the research gap and scope of the study.
B. To design an RBG3 architecture that incorporates the DRBG AES256, considering the NIST SP 800-90B and SP 800-22 standards.
C. To implement the RBG3 with DRBG AES256 and evaluate its performance in terms of speed, randomness, and security.
D. To validate the RBG3 with DRBG AES256 for cryptographic applications, including key generation, digital signatures, and secure communication, and compare its performance with traditional RBGs and DRBGs.
E. To analyze the impact of the proposed RBG3 with DRBG AES256 on the security of cryptographic systems and provide recommendations for further research in this area.

F. To contribute to the development of a more secure and efficient RBG3 with DRBG AES256 that meets the modern security requirements of various applications.

Chapter 2: Literature Review

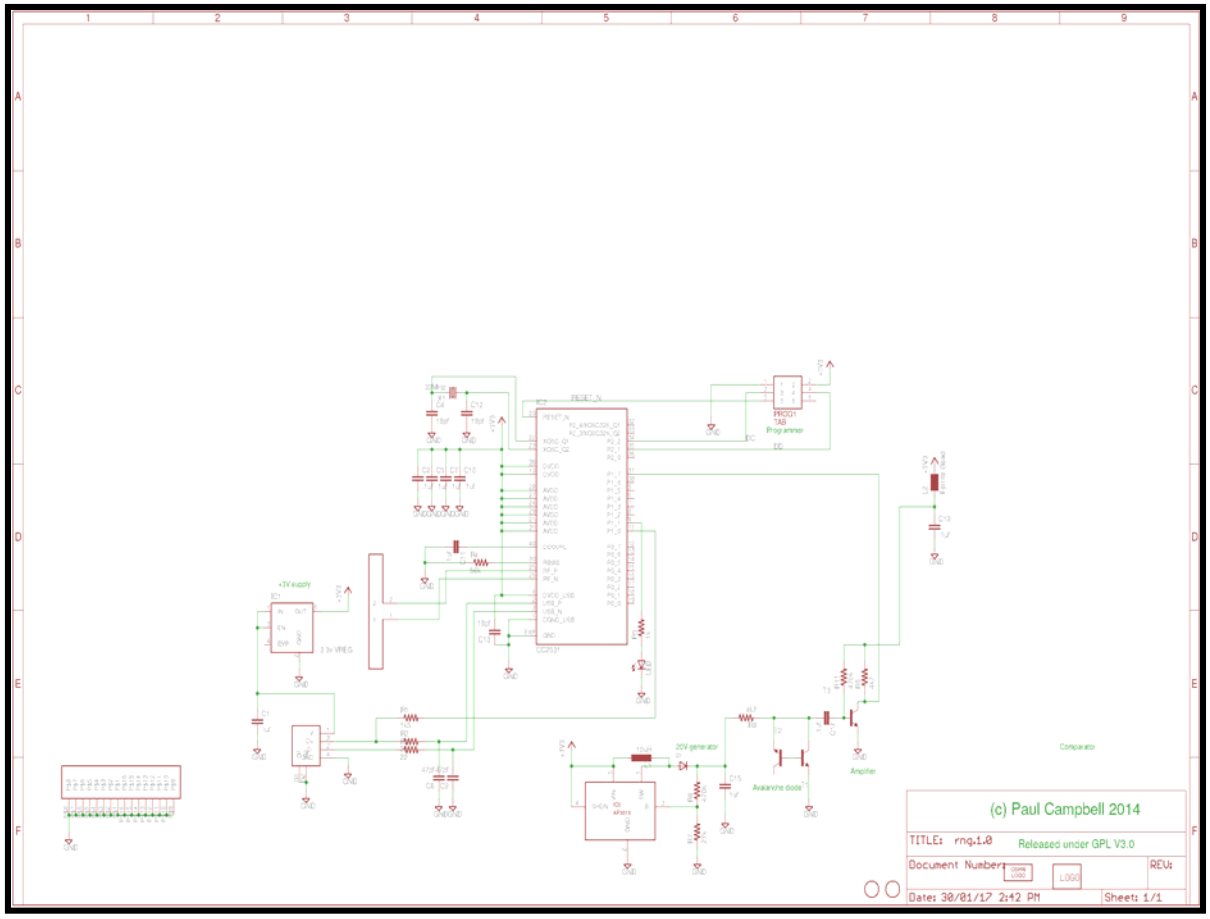
2.1 Review of existing TRNG (OneRNG)

2.1.1 General Description

OneRNG is a well-known open-source hardware project with the goal of offering various applications with a high-quality True Random Number Generator (TRNG). The NIST SP 800-90B and SP 800-22 criteria for randomness and security are met by the OneRNG, which is based on the Avalanche Noise and designed to do so. We will go over the history, conception, and performance assessment of the OneRNG in this review of the literature.

2.1.2 Design

The OneRNG is based on the Avalanche Noise, a scientific phenomenon that causes erratic voltage fluctuations as a result of electron mobility in a semiconductor. A voltage divider, an amplifier, and a Schmitt trigger make up the OneRNG circuit, which converts analogue noise into digital signals. The circuit is made to deliver a high-quality random bit stream at up to 4 Mbps of bit rate.



2.1.3 Performance Assessment

The OneRNG has undergone a thorough randomness, security, and performance assessment. Several statistical tests, such as the NIST Statistical Test Suite, the Dieharder Test Suite, and the TestU01 Test Suite, were used to evaluate the OneRNG's unpredictability. The tests were successful, proving that the OneRNG is reliable and generates random numbers that are up to the NIST SP 800-90B and SP 800-22 requirements for randomness.

2.1.4 Security

By examining the effects of several assaults, such as the bias injection attack, the clock glitch attack, and the power analysis attack, the security of the OneRNG was also assessed. The outcomes

demonstrated the OneRNG's resistance to these assaults, demonstrating that it offers a high level of security for cryptographic applications.

2.1.5 Speed and Power Usage

The OneRNG's performance was assessed in terms of speed, power usage, and hardware complexity. The OneRNG was discovered to be quick, use little power, and have a straightforward hardware design, making it appropriate for a variety of applications.

2.1.6 Conclusion

OneRNG is a top-notch open-source TRNG that offers a reliable and effective way to generate random numbers for a variety of applications. The OneRNG has undergone comprehensive testing for randomness, security, and performance, and the findings demonstrate that it satisfies the current security standards of a number of applications. The OneRNG is easy to integrate into numerous systems thanks to its straightforward and small design. The OneRNG is a promising replacement for TRNGs and is appropriate for many uses that call for high-quality random numbers.

2.2 Overview of NIST standards for random number generation

In the area of random number generation, NIST has published several standards, including SP800-90A, SP800-90B, and SP800-90C.

2.2.1 SP800-90A

Titled "Recommendation for Random Number Generation Using Deterministic Random Bit Generators," provides guidelines for generating random numbers using deterministic random bit generators (DRBGs). The standard specifies the security requirements for DRBGs and provides algorithms for generating random numbers based on cryptographic primitives such as hash

functions and block ciphers. The standard also specifies entropy sources and tests to ensure the quality of the generated random numbers.

2.2.2 SP800-90B

Titled "Recommendation for the Entropy Sources Used for Random Bit Generation," provides guidelines for the evaluation of entropy sources used for generating random numbers. The standard defines the concept of entropy and the requirements for a good entropy source. It also specifies statistical tests for evaluating the quality of the entropy source and provides guidelines for designing tests that can detect and quantify any biases in the entropy source.

2.2.3 SP800-90C

Titled "Recommendation for Random Bit Generator (RBG) Constructions," provides guidelines for the construction of random bit generators (RBGs) using DRBGs and entropy sources. The standard specifies the requirements for the construction of RBGs and provides algorithms for generating random numbers using DRBGs and entropy sources. It also provides security requirements for RBGs and specifies tests for evaluating the quality of the generated random numbers.

2.2.4 Obligation of compliance

Together, these standards provide a comprehensive framework for generating high-quality random numbers and ensuring their security. Compliance with these standards is important for ensuring the security of cryptographic systems that rely on random numbers, such as encryption, digital signatures, and secure communications**.**

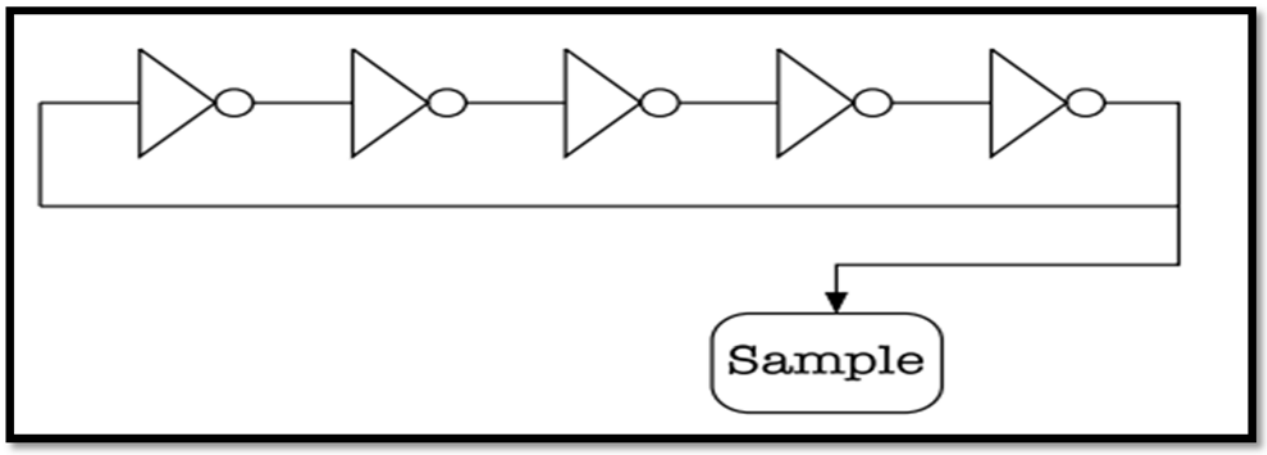
Chapter 3: Proposed TRNG and DRBG Design

3.1 Entropy Sources

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.

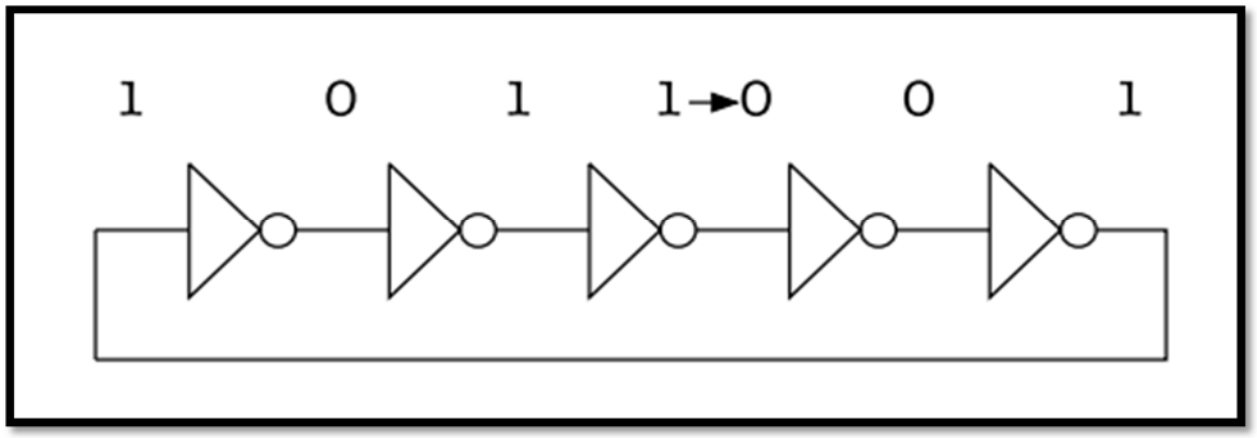
3.1.1 Ring Oscillator

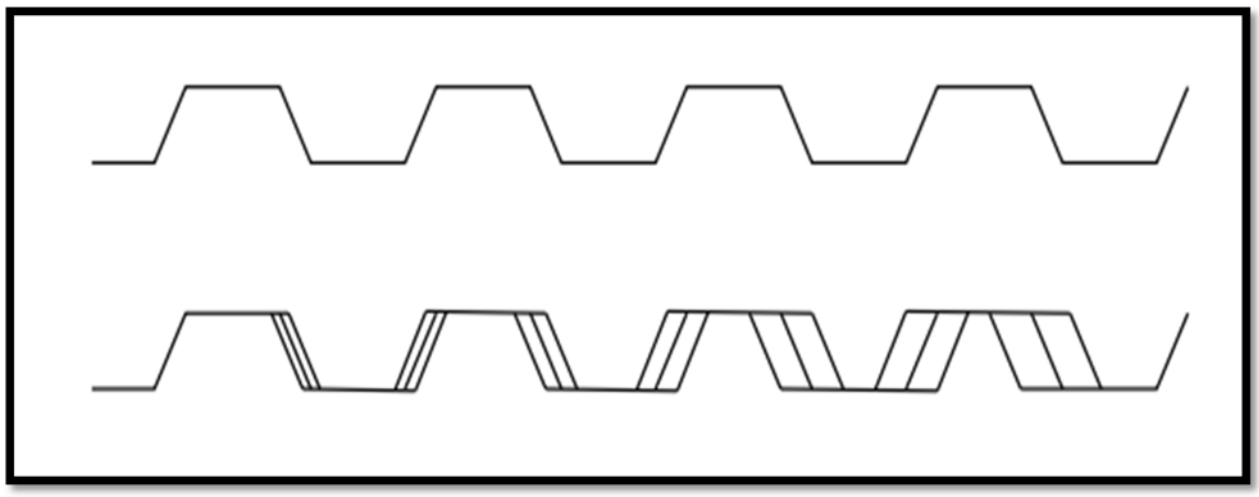
They can be used as an entropy source for a hardware-based True Random Number Generator (TRNG) based on RBG3 to produce random numbers. A ring oscillator is a circuit with a closed loop feedback route made up of an odd number of inverters connected in a loop. To create a ring of inverters that fluctuate between logic high and logic low levels, the output of the final inverter is fed back into the input of the first inverter. Output is sampled from some point in the chain. Sampling rate must be less than the frequency of ring oscillator. Uncertainity in clock periods results in different lengths which subsequently equals to a single bit period.



The binary value alternates from 0 to 1 to 0 as it crosses the inverters. However, since there is an odd number of inverters, there must be a discontinuity point where there is an input value that equals the output value. The output of that inverter gate, therefore, changes to the opposite value. The

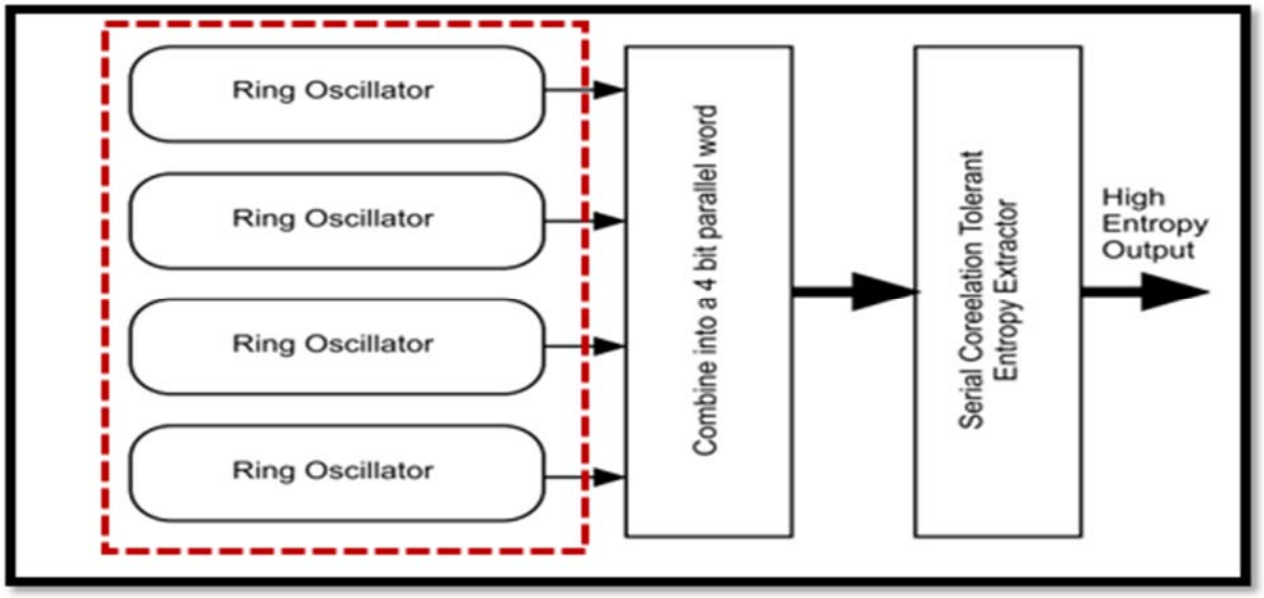
discontinuity, as a result, moves on to the next inverter. This discontinuity carries on cycling around the circuit. If you look at the waveform at any particular point, then you will see that the value oscillates up and down, changing once for each time the discontinuity makes a trip around the circuit.





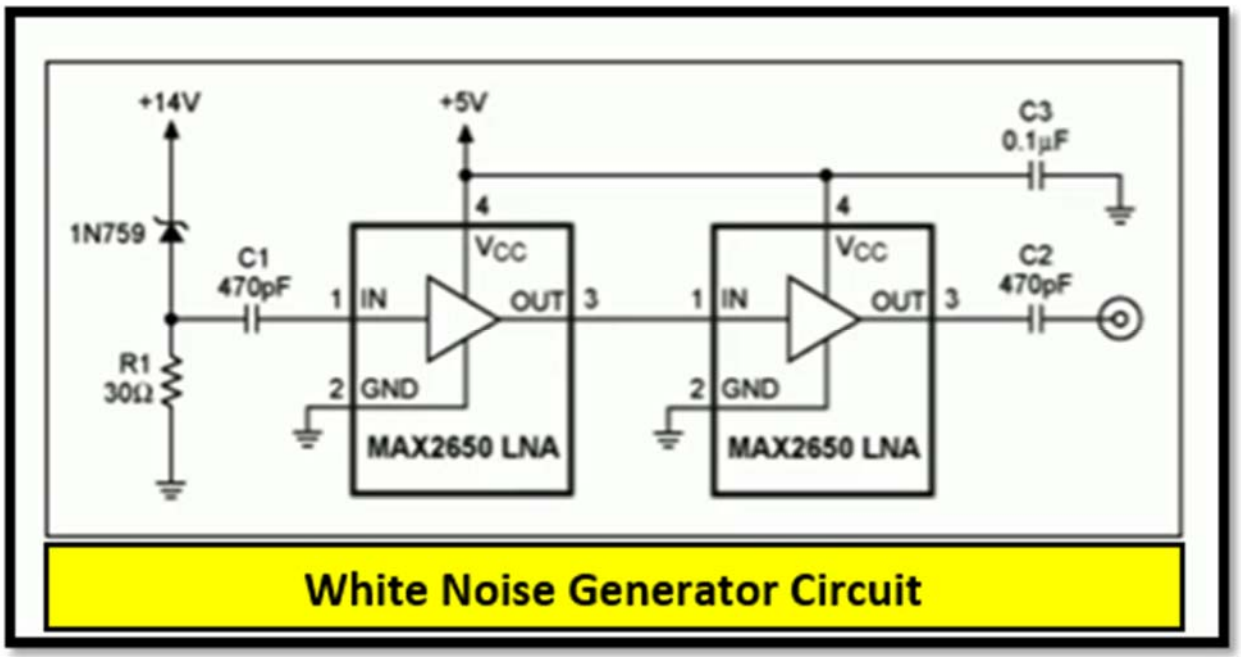
The intrinsic unpredictability of a ring oscillator's oscillation frequency due to process variances, temperature fluctuations, and other external conditions underlies its operation as an entropy source. The output frequency of the ring oscillator becomes unpredictable as a result of these oscillations,

making it a viable entropy source for producing random numbers. This viable source has an inherent flaw of slow output which can be offset by combination of multiple ring oscillators.



3.1.1 Zener diode breakdown Noise

Reverse biassing a diode at the avalanche breakdown point, amplification, and sampling of the resulting electrical noise represent an old method for producing electrical noise in random number generators. On the Internet, there are numerous examples of circuit topologies for avalanche noise-based entropy generators. They often fall into one of two categories: noise sources based on transistor diodes or zener diodes. A general model for a circuit layout that breaks down a transistor diode, amplifies the resulting noise, buffers it, and then transforms it to digital voltage levels is shown in the example below.



Zener breakdown, also known as Zener diode breakdown, can be used as an entropy source for generating random numbers in a True Random Number Generator (TRNG) based on NIST standards. Zener diodes are special types of diodes that operate in the reverse-biased breakdown region, where they exhibit a highly nonlinear voltage-current characteristic. This characteristic can be exploited to generate random numbers with high entropy.

At the breakdown point, the voltage across the Zener diode is very unpredictable and affected by a number of variables, including temperature, manufacturing variability, and electrical noise. The voltage levels at the Zener diode's output are affected by these variables, which can be used as an entropy source to produce random numbers.

A stream of random bits can be produced by further processing the Zener diode's output voltage. As an illustration, the output voltage can be converted to digital form using an analog-to-digital converter (ADC), and the digital values that result can then be processed using statistical tests, cryptographic algorithms, and other methods to extract high-quality random numbers with high entropy.

3.2 RBG3-based TRNG design

RBG3 is an abbreviation for "Random Bit Generator, version 3," which is a particular design methodology for creating TRNGs. A noise source, a conditioning circuit, and a deterministic post-processing algorithm are its three main parts.

3.2.2 Noise source

The primary component that creates random data is the noise source. Inherent randomness and unpredictable behaviour, such as thermal noise, shot noise, or quantum noise, should be present. The noise source employed in an RBG3-based TRNG must be well-characterized, which means that its statistical characteristics, such as entropy rate and bias, must be carefully examined and documented.

3.2.3 Conditioning circuit

The conditioning circuit takes the raw output from the noise source and processes it to improve its randomness and remove any biases. It typically involves amplification, filtering, and/or digitization of the noise signal. The conditioning circuit should be carefully designed to minimize any potential introduction of biases or correlations in the generated random numbers. Its main role is to process the raw output from the noise source to improve the randomness and quality of the generated random numbers. Following are the key roles of the conditioning circuit in an RBG3-based TRNG design as per NIST.

3.2.3.1 Amplification

In order to prepare the raw noise signal for subsequent processing, the conditioning circuit may boost it to an appropriate level. It is crucial to carefully amplify the signal to prevent biases or distortions that can impair the randomness of the generated random numbers.

3.2.3.2 Filtering

To remove any undesirable components or noise beyond the required frequency range, the conditioning circuit may apply filters to the raw noise signal. This aids in isolating the necessary random signal and raises the standard of the randomly generated numbers.

3.2.3.3 Digitization

In order to facilitate subsequent processing, the conditioning circuit may convert the analogue noise signal to a digital representation. In order to preserve the randomness and quality of the noise signal without adding any quantization bias or other artefacts, the digitization procedure should be properly planned.

3.2.3.4 Bias Removal

In order to correct any bias or non-uniformity in the raw noise signal, the conditioning circuit may employ certain strategies. Random numbers that are biassed or correlated due to noise signal bias might jeopardise the security of cryptographic applications. According to NIST recommendations, any bias that the conditioning circuit introduces needs to be thoroughly examined and minimised.

3.2.3.5 Error Correction

To identify and fix any potential faults or alterations in the raw noise signal, the conditioning circuit may use error correction techniques. By guaranteeing that the generated random numbers are free from any unwanted errors or distortions, this helps to increase their dependability and quality.

3.3 DRBG-AES256 design

The phrase "DRBG-AES256" refers to a Deterministic Random Bit Generator (DRBG) design that follows the recommendations made by the National Institute of Standards and Technology (NIST) in the United States and employs the Advanced Encryption Standard (AES) with a 256-bit key length. In many security applications, DRBGs, which are cryptographic algorithms that generate

random numbers from a known seed value, are employed. Here is a summary of the DRBG-AES256 design in accordance with NIST recommendations:

3.3.1 The DRBG-AES256 algorithm

It employs the AES algorithm and a 256-bit key length. AES is a symmetric-key encryption technique that uses keys with lengths of 128, 192, or 256 bits and works with fixed-size data blocks (128 bits). AES-256's adoption offers a high level of security because it has robust encryption capabilities and a vast key space. The DRBG medium functions handle the DRBG's internal state. The DRBG mechanisms in this Recommendation have five separate functions

1. The instantiate function acquires entropy input and may combine it with a nonce and a personalization string to produce a seed from which the original internal state is created.
2. The induce function generates pseudorandom bits upon request, using the current internal state and conceivably fresh input; a new internal state for the coming request is also generated.
3. The reseed function acquires new entropy input and combines it with the current internal state and any fresh input that's handed to produce a new seed and a new internal state.
4. The uninstantiate function zeroizes(i.e., erases) the internal state.
5. The health test function determines that the DRBG medium continues to serve rightly.

The DRBG mechanisms specified in this Recommendation support four security strengths 112, 128, 192 or 256 bits. The security strength for the externalization is requested during DRBG externalization, and the instantiate function obtains the applicable quantum of entropy for the

requested security strength. Each DRBG medium has restrictions on the security strength it can support, grounded on its design(see Section 10).

The factual security strength supported by a given externalization depends on the DRBG perpetration and on the quantum of entropy handed to the instantiate function. Note that the security strength actually supported by a particular externalization could be lower than the maximum security strength possible for that DRBG perpetration(see Table 1). For illustration, a DRBG that's designed to support a maximum security strength of 256 bits could, rather, be expressed to support only a 128- bit security strength if the fresh security handed by the 256- bit

3.3.2 Seed

DRBG-AES256 accepts as input a seed value that acts as the starting point for producing random numbers. To ensure the security of the generated random numbers, the seed value needs to be unpredictable and confidential. The minimal length of the seed depends on the DRBG medium and the security strength needed by the consuming operation, but shall be at least the number of bits of entropy needed.

A nonce may be needed in the construction of a seed during externalization in order to give a security bumper to block certain attacks. The nonce shall be either

A value with at least(security_strength/ 2) bits of entropy, or

A value that's anticipated to repeat no more frequently than a(security_strength/ 2)- bit arbitrary string would be anticipated to repeat.

Each nonce shall be unique to the cryptographic module in which externalization is performed, but need not be secret. When used, the nonce shall be considered to be a critical security parameter.

A nonce may be composed of one(or further) of the following factors(other factors may also be applicable):

1. A random value that is generated anew for each nonce, using an approved random bit generator.
2. A timestamp of sufficient resolution (detail) so that it is different each time it is used.
3. A monotonically increasing sequence number, or
4. A combination of a timestamp and a monotonically increasing sequence number, such that the sequence number is reset when and only when the timestamp changes. For example, a timestamp may show the date but not the time of day, so a sequence number is appended that will not repeat during a particular day.

For case 1 above, the random value could be acquired from the same source and at the same time as the entropy input. In this case, the seed could be considered to be constructed from an "extra strong" entropy input and the optional personalization string, where the entropy for the entropy input is equal to or greater than (3/2 security_strength) bits.

For case 2 above, the timestamp must be trusted. A trusted timestamp is generated and signed by an entity that is trusted to provide accurate time information.

The nonce provides greater assurance that the DRBG provides security_strength bits of security to the consuming application. If a DRBG were instantiated many times without a nonce, a compromise could become more likely. In some consuming applications, a single DRBG compromise could

reveal long-term secrets (e.g., a compromise of the DSA per-message secret could reveal the signing key).

A nonce shall be generated within a cryptographic module boundary. This requirement does not preclude the generation of the nonce within a cryptographic module that is different from the cryptographic boundary containing the DRBG function with which the nonce is used (e.g., the cryptographic module boundary containing an instantiate function). However, in this scenario, there needs to be a secure channel to transport the nonce between the cryptographic-module boundaries.

3.3.3 Personalization string

A personalization string is an optional (but recommended) input to the instantiate function and is used to derive the seed . The personalization string may be obtained from inside or outside a cryptographic module, and may be an empty string. Note that a DRBG does not rely on a personalization string to provide entropy, even though entropy could be provided in the personalization string, and knowledge of the personalization string by an adversary does not degrade the security strength of a DRBG instantiation, as long as the entropy input is unknown. When used within a cryptographic module, a personalization string is not considered to be a critical security parameter.

The personalization string may contain secret information, but shall not include secret information that requires protection at a higher security strength than the DRBG being instantiated will support. For example, a personalization string to be used to instantiate a DRBG at 112 bits of security strength shall not include information requiring 128 bits of protection. A given implementation of a DRBG may support the use of a personalization string, but is not required to do so.

The intent of a personalization string is to introduce additional input into the instantiation of a DRBG. This personalization string might contain values unknown to an attacker, or values that tend to differentiate this DRBG instantiation from all others. Ideally, a personalization string will be set to some bitstring that is as unique as possible. Good sources for the personalization string contents include:

Application identifiers

Special key values for this specific

Device serial numbers

DRBG instantiation

User identification

Protocol version identifiers,

Per-module or per-device values

Network addresses

3.3.4 Prediction Resistance and Backtracking Resistance

Backtracking Resistance: Backtracking resistance is provided relative to time T if there is assurance that an adversary who has knowledge of the internal state of the DRBG at some time subsequent to time T would be unable to distinguish between observations of ideal random bitstrings and (previously unseen) bitstrings that were output by the DRBG prior to time T. This assumes that the adversary is incapable of performing the work required to negate the claimed security strength of the DRBG. Backtracking resistance means that a compromise of the DRBG internal state has no effect on the security of prior outputs. That is, an adversary who is given access to all of the prior output sequence cannot distinguish it from random output with less work than is associated with

the security strength of the instantiation; if the adversary knows only part of the prior output, he cannot determine any bit of that prior output sequence that he has not already seen with better than a 50-50 chance.

For example, suppose that an adversary knows Statex. Backtracking resistance means that:

1. The output bits from State1 to Statex-1 cannot be distinguished from random output
2. The prior internal state values themselves (State1 to Statex-1) cannot be recovered, given knowledge of the secret information in Statex.

Backtracking resistance can be provided by ensuring that the DRBG generation algorithm is a one-way function. All DRBG mechanisms in this Recommendation have been designed to provide backtracking resistance.

Prediction Resistance: Prediction resistance means that a compromise of the DRBG internal state has no effect on the security of future DRBG outputs. That is, an adversary who is given access to all of the output sequence after the compromise cannot distinguish it from random output with less work than is associated with the security strength of the instantiation; if the adversary knows only part of the future output sequence, he cannot predict any bit of that future output sequence that he does not already know (with better than a 50-50 chance).

For example, suppose that an adversary knows Statex. Prediction resistance means that:

1. The output bits from Statex+1 and forward cannot be distinguished from an ideal random bitstring by the adversary, and

2. The future internal state values themselves (Statex+1 and forward) cannot be predicted (with better than a 50-50 chance), given knowledge of Statex.

Prediction resistance is provided relative to time T if there is assurance that an adversary with knowledge of the state of the RBG at some time(s) prior to T (but incapable of performing work that matches the claimed security strength of the RBG) would be unable to distinguish between observations of ideal random bitstrings and (previously unseen) bitstrings output by the RBG at or subsequent to time T. In particular, an RBG whose design allows the adversary to step forward from the initially compromised RBG state(s) to obtain knowledge of subsequent RBG states and the corresponding outputs (including the RBG state and output at time T) would not provide prediction resistance relative to time T.

Prediction resistance can be provided only by ensuring that a DRBG is effectively reseeded with fresh entropy between producing output for consecutive DRBG requests. That is, an amount of entropy that is sufficient to support the security strength of the DRBG being reseeded (i.e., an amount that is at least equal to the security strength) must be provided to the DRBG in a way that ensures that knowledge of the current DRBG internal state does not allow an adversary any useful knowledge about future DRBG internal states or outputs. Prediction resistance can be provided when the randomness source is or has direct or indirect access to an entropy source or an NRBG

For example, suppose that an adversary knows internal Statex-2. If the adversary also knows the DRBG mechanism used, he then has enough information to compute Statex-1 and Statex. If

prediction is then requested for the next bits that are to be output from the DRBG, new entropy bits will be inserted into the DRBG instantiation before Statex+1 is produced that will create a separation between Statex and Statex+1, i.e., the adversary will not be able to compute Statex+1, simply by knowing Statex; the work required will be greatly increased by the entropy inserted during the prediction request.

The introduction of fresh entropy via reseeding will also make the DRBG less susceptible to cryptanalytic attack. Whenever an entropy source is available, it is strongly recommended that DRBGs be requested to provide prediction resistance as often as is practical.

3.3.5 Reseeding

To maintain the randomness and security of the generated random numbers, DRBG-AES256 may include a reseeding mechanism where the seed value is refreshed on a regular basis or when specific criteria are satisfied. Specific requirements for reseeding, such as frequency and procedure quality, are provided by NIST guidelines.

3.3.6 Security Considerations

In accordance with NIST recommendations, the DRBG-AES256 design must take a number of security factors into account. This covers defence against well-known attacks like brute force, statistical, and other cryptographic ones. The generated random numbers must also have enough entropy, or high randomness and unpredictability, according to the design.

3.3.7 Testing and Validation

To make sure that DRBG-AES256 complies with the advised requirements, it should go through a thorough testing and validation process in accordance with NIST rules. In order to evaluate the calibre, security, and dependability of the generated random numbers, this comprises statistical tests, performance tests, and other validation methods.

Chapter 4: Performance and Evaluation of Testing Random numbers

3.1 Known Answer Tests

A test that determines whether deterministic algorithms, such as online health tests, entropy extractors, and PRNGs, are functioning properly. It is a method for demonstrating that an implemented random number generator algorithm complies with a specification. From a starting seed state, a reference implementation written in a high-level language is used to produce a set of random reference random numbers. The testing implementation is put in the same state and given a run to produce random numbers. When the reference numbers and the numbers from the implementation are compared, the numbers will match if the implementation follows the algorithm. NIST has provided answers for SP800-90A-compliant DRBG algorithms. http://csrc.nist.gov/groups/STM/cavp/documents/drbg/DRBGVS.pdf. It describes formats for communicating initial states and a test procedure for each type of DRBG described in SP800-90A. A descriptor indicates the type of the algorithm being tested, in this case the Hash DRBG using SHA-1

3.2 Distinguishability Test

A stream of random bits is examined using this statistical hypothesis testing methodology to determine whether the random data under test can be distinguished from a 100% entropic stream of bits. A pass/fail rating and confidence level are the outcomes. (for eg P=0.999). The test algorithm's determination of whether the data under test "looked" random is indicated by the pass/fail. Confidence provides the probability of the hypothesis (or its inverse). It is a bit strange. in contrast to standard statistical tests. You would typically be searching for a specific hypothesis. This test

seeks to demonstrate the hypothesis that the data contains some flaw, such as a discernible departure from perfect randomness. The test is as follows:

3.2.1 Monobit.

It is a test of bias. whether a sequence of bits was generated with no bias. It is not sensitive to the order of bits (Code avail on page 201)

3.2.2 Frequency Test Within a Block

It is an improvement on the monobit test. (Code avail on page 202)

3.2.3 Discrete Fourier Transform

Computes a spectrograph of the binary in- put data and establishes that there are no frequency components that are outside the bounds of what would be expected for truly random data. It is particularly applicable to directly testing circuits with feedback that may lead to oscillating behaviour. (Code avail on page 207)

3.2.4 Nonoverlapping Template Matching

Count occurrences of a pattern within blocks of the data being tested. (Code avail on page 210)

3.2.5 Overlapping Template Matching

It is like the Nonoverlapping Template Test in that it counts the frequency of the number of matches of a pattern within a number of blocks. The difference with this measurement is that when a match is made, the pattern matching algorithm moves along by one bit position, so it can match the same pattern in an overlapping position. (Code avail on page 215)

3.2.6 Longest Run of Ones in a Block

This test splits the data into blocks, finds the length of longest run of ones in each block, and assembles a frequency table of the lengths. (Code avail on page 219)

3.2.7 Binary Matrix Rank

This test looks for linear dependence between strings of bits in the data. It is most appropriate for simple PRNGs, which might introduce linear dependence between bits where subsequent bits can be expressed as a simple polynomial of previous bits. (Code avail on page 224)

3.2.8 Random Excursion

This test builds a one-dimensional random walk from the data This test is particularly sensitive to mean reverting behaviour in data. (Code avail on page 228)

3.2.9 Random Excursion Variant

It is similar to the Random Excursion Test, except that instead of counting the frequency of occurrence of values within cycles, it treats the sequence as one big cycle and counts the occurrence over a wider range of numbers. (Code avail on page 231)

3.2.10 Maurer's Universal Statistical

It is a compression test. It attempts to establish if the data is compressible. (Code avail on page 236)

3.2.11 Linear Complexity

This test runs the Berlekamp–Massey algorithm over blocks of data from the input data; the algorithm (Berlekamp–Massey) computes the shortest LFSR that can output the block. Over uniform data, the algorithm should not be able to find LFSRs that have a linear complexity significantly shorter than the input data block. (Code avail on page 239)

3.2.12 Serial

He serial test counts the frequency of overlapping bit-patterns within the bit series (Code avail on page 244)

3.2.13 Cumulative Sums

It computes the maximum excursion of a random walk over the cumulative sums over the bits in the sequence. It does this by first running through the data forwards, then again running through the data backwards. (Code avail on page 246)

3.2.14 Approximate Entropy

It counts the frequency of all patterns of a certain bit-length m and all the patterns of bit-length m+1, then performs a $\chi 2$ test on the expected difference between the two counts and the actual difference. (Code avail on page 249)

3.3 PRNG Test Suites

There are several online tests suites available which test PRNG

3.3.1 Dieharder

Dieharder is a software program used for testing the quality and randomness of random number generators (RNGs). It is designed to run a battery of statistical tests on the output of an RNG, in order to determine whether the generator is producing truly random numbers or not. It can be downloaded from http://www.phy.duke.edu/~rgb/General/dieharder.php

3.3.2 NIST SP800-22

Another common test suite is specified in SP800-22 [19] by NIST. a soft-ware implementation of the SP800-22 test suite at http://csrc.nist.gov/groups/ST/toolkit/rng/documentation_software.html. The dieharder test suite implements a number of the NIST SP800-22 tests

3.3.3 SEMB GM/T 0005-2012

China's SEMB security standards body has published a statistical test suit GM/T 0005-2012 derived from the first version NIST SP800-22, but it excludes some of the tests and includes some improved tests such as Maurer's Universal Test Statistic. The later version of SP800-22 caught up with these changes, so the two test suites are very similar.

3.4 Entropy Measurement

There are two primary entropy measurements that are useful in measuring random data, Shannon entropy measurements and min-entropy measurement. These tests are typically run on non-post-processed data from an entropy source. Measuring the Shannon entropy gives a measure of the uniformity of the distribution of symbol frequencies from the source, and this gives a measure of the information capacity of the data. These tests are typically run on non-post-processed data from

an entropy source. it gives a measure of the uniformity of the distribution of symbol frequencies and measure of the information capacity of the data.

3.4.1 Shanon Entropy estimation

The average amount of information in data, as determined by the frequency distribution of the symbols that make up the data, is known as Shannon entropy. The information carrying capacity of a channel is described in Min Entropy Estimation | 149 communication theory. The more information a symbol carries when it is sent, the lower its probability. Since it considers the frequency of every symbol in the distribution, the average entropy is another way that Shannon entropy is frequently described. The common Shannon entropy equation describes this.

Code for Shanon entropy estimation is:

```
import random
from collections import Counter
import math
# Generate a list of 1000 random numbers between 1 and 10
data = [random.randint(1,10) for i in range(1000)]
# Count the frequency of each value in the data set
freq = Counter(data)
# Calculate the probability of each value
total = len(data)
probs = [freq[x]/total for x in freq]
# Calculate the Shannon entropy of the data set
entropy = -sum([p * math.log2(p) for p in probs])
```

```
print("Entropy:", entropy)
```

3.4.2 Min Entropy Estimation

Min-entropy estimate algorithms come in two forms. The first type of test tries to establish a number that is close to the actual min-entropy of the data. The second type tries to give a lower bound for the entropy in a body of data, under a well-defined model of entropy, such as Rényi entropy.

Code for min entropy estimation is:

```
import math

estimations = [1, 2, 3, 4, 5, 6, 7, 8]

# Generate random estimation numbers and count frequency

freq = {}

for i in range(1000):

    num = random.choice(estimations)

    freq[num] = freq.get(num, 0) + 1

# Calculate probability and entropy

entropy = 0

for num in freq:

    p = freq[num] / 1000

    entropy -= p * math.log2(p)

print("Minimum entropy:", entropy)
```

3.5 Model Eqivalnce Test

A model equivalence test is carried out by taking a sizable sample of data from a random source, calculating the mean and standard deviation, and then testing whether the data has a Gaussian

distribution. The model and the physical entropy source are equivalent if the measured mean and standard deviation agree with the values from the model and it can be demonstrated that the data has a Gaussian distribution. This enables the model to be used to theorize about the source's minimum entropy. Model equivalence testing is therefore frequently used to demonstrate that the model accurately represents the source rather than to test the source

3.6 Stat prerequisite testing

These tests typically involve measuring the data with straightforward math's. They can be useful for testing whether input data satisfies the specifications of the consuming algorithm or for analyzing data flaws (such as bias or correlation). For example:

3.6.1 Mean

The mean of data is the sum of all the data divided by the number of values.

3.6.2 Standard Deviation

The average amount of variability in your dataset. It tells you, on average, how far each value lies from the mean

3.6.3 The $\chi 2$ Test of Randomness

It is useful where data is random, and it ensures that it is unbiased.

3.6.4 Serial Correlation Coefficient

It gives a measure of how much a value in a list of random values is linked to its neighbouring bits. Lag-N correlation computes the correlation between each value, and the value that is N positions away in the list. (Code avail on page 160)

3.6.5 Lag-N Correlation

The SCC algorithm computes the correlation between each bit and the next bit. We can extend the SCC algorithm to compute the correlation between each bit and the bit that is n steps away. (Code avail on page 162,164)

3.6.6 Efficient Computation of a Correlogram using FFTs

The computation of a correlogram takes O(n2) multiplications. So, for large amounts of data, this is computationally unfeasible. The Wiener–Khinchin algorithm achieves O(n log2(n)) multiplications using Fast Fourier Transforms. (Code avail on page 167)

3.7 The Problem Distinguishing Entropy and Pseudo randomness

Anyone attempting to test for entropy, or uniform, nondeterministic randomness, faces a fundamental problem. It can be demonstrated that it is computationally impossible to distinguish between cryptographically secure pseudorandom data and true randomness to an observer who does not have the proper key. However, it is simple for an observer who has the PRNG's key to tell a true random stream from a pseudorandom stream produced using that key. As a result, it is impossible to analyse a random number stream and declare it to be entropic in the general case. When you are only looking at the data and not the source, it might be pseudorandom, and there is no way to know that.

3.8 Statistical Tests of Uniformity

Statistical tests of uniformity are statistical methods used to determine if a data distribution is uniformly distributed across all possible values in a given range. These tests can help to determine

if there is any systematic bias or tendency in the data set. There are different statistical tests of uniformity, including the chi-square test, the Kolmogorov-Smirnov test, and the Anderson-Darling test, that can be used depending on the type and characteristics of the data distribution. These tests are widely used in various fields, including economics, biology, engineering, and social sciences, to analyze data and draw conclusions about the underlying population from which the data was obtained.

3.9 Codes

All the codes written in red are directly aquired from the book

David Johnston Random Number Generators—Principles and Practices

Chapter 4: Online Random Number Testing

The adaptive proportion test and repetition count test, which are both meant to test the data output from an entropy source, are two online tests that are described in the NIST SP800-90 standard. Both tests don't work very well. To test the output of the entire RNG at the DRBG output, a new FIPS 140-2 standard known as continuous random number generator test (CRNGT) is added.

4.1 Process for conducting Tests

Blocking or Tagging Online tests are typically run between data generation and usage, allowing the data to be discarded in the event of a failure. Online tests have two possible outcomes: blocking or tagging. Data transfer from the entropy source to the extractor is halted during a blocking test until the test is finished. Consequently, information must be kept in memory. For conduction of test. The data in memory is deleted if the test is unsuccessful. The amount of data that the test can examine, and the storage hardware required to store it must be balanced in a hardware implementation. In a software implementation, leaving random data to be used as keys lying around in memory could be a security risk. In a different approach called tagging, data is tested before moving on to the next stage, and the test appends the outcome to the end of a tested block of data. Implementation of an online health test tagging. The test's outcome is also sent to the extractor at the end of a data block. After receiving the tested data, the following stage will make the proper decisions based on the health tag, such as discarding the extracted data or adding more data before releasing the extracted data to the following stage.

4.2 What Online Test Testing for?

Online tests are those in which it is not anticipated that the data from an entropy source will be completely random. Therefore, it can be said that online tests are evaluations of specific entropy source failure modes over small data blocks. Entropy sources' failure modes are frequently described as producing high bias or high serial correlation. Simple examples include flatline outputs, where all the data is one or zero. Analysis of all SPOF (Single Point of Failure) and DPOF (Double Point of Failure) errors and simulation of output for each case is the fundamental engineering approach to identifying failure modes. Usually, a small number of output defects account for most of the outcomes, if not all of them. Online health tests frequently have the design of the tests coupled with the source's failure modes

4.3 Standard-based online tests

4.3.1 FIPS 140-2 continuous RNG Test

It is a standard that sets the requirements for cryptographic modules used in information technology systems by various security agencies. One of the requirements included in this standard is RNG (Random Number Generator) testing. The FIPS 140-2 Continuous RNG Test is a test performed on an RNG to ensure that it generates true random numbers continuously without any patterns or predictability. This test involves monitoring the output of the RNG for a specified period to ensure that the generated numbers are statistically and distributive random. (CODE avail at Page 177).

4.3.2 SP800-90B Repetition count Test

It is a statistical test that is used to make sure that pseudorandom number generators (PRNGs) are random and unpredictable. This test counts the number of times a PRNG produces outputs that are

exactly the same after another. The test measures how frequently an output value sequence repeats itself within a given sample set. If the repetition count exceeds a certain threshold, the PRNG is not secure and fails the test. This is because the occurrence of identical outputs in a PRNG should be statistically improbable. The SP800-90B repetition count test is used to detect the presence of repeated values in a sequence. The algorithm is as follows:

Define a sequence of values.

Initialize a counter variable to 0.

For each element in the sequence, compare it to the previous element.

If the current element is the same as the previous element, increment the counter.

If the counter exceeds a predetermined threshold (i.e. the repetition count limit), return the test result as FAIL.

If the entire sequence is processed without the counter exceeding the repetition count limit, return the test result as PASS.

Code for Reptation count test

```
def repetition_count_test(sequence, limit):
    count = 0
    for i in range(1, len(sequence)):
        if sequence[i] == sequence[i-1]:
```

```
            count += 1
            if count > limit:
                return "FAIL"
        else:
            count = 0
    return "PASS"
sequence = [1, 2, 3, 3, 3, 4, 5, 5]
limit = 3
result = repetition_count_test(sequence, limit)
print(result)  # prints "FAIL"
```

4.3.3 SP800-90B Adaptive proportion Test

It is similar to the RCT, but instead of checking that the value is repeated continuously as the RCT does, it counts the frequency of a value within a range of RNG output values and makes sure that the frequency is not excessive. A block's starting data is used to determine the value to be searched for. The test restarts once a number of values have been examined, choosing the following output value as the new value to count. There are problems with this test. the standard describes some of the problems, and these problems are not sufficient that tests from the standard must be omitted. Some of the standards are:

4.3.3.1 Some noise sources simply do not generate many samples. If an entropy source never processes as many noise source samples as appear in a window for this test, the test will never complete, and there will be little or no benefit in running the test at all.

4.3.3.2 A larger window size allows for the detection of more subtle failures in the noise source.

4.3.3.3 A larger window size means that each test takes longer to complete.

4.4 Additional Verification

Beside these tests there are several other Tests available which adds Health in the randomness of the RNG:

4.4.1 Pattern Counting Health Test

This test replaces the Adaptive Proportion Test and the Repetition Count Test and is implemented in the RNG in Intel CPUs. A certification lab has approved this test as a valid replacement for the required tests. The algorithm, which is essentially a tagging test, examines groups of 256 bits as they emerge from the entropy source and marks each block with a single bit healthy flag. Six different bit patterns within the data are counted over a sliding window to determine their frequency. The six patterns are 1, 01, 101, 1001,010, and 0110. This test is implemented by the Python programmed. (Code avail on page 187).

4.4.2 Online Mean and Serial Correlation Test

The serial correlation coefficient (SCC) of the data is typically found to be the property that is directly related to the min entropy of the data if you use an entropy source circuit that uses electrical

feedback to keep the bias centred at 50% ones and 50% zeroes. The SCC of 256-bit blocks over data from a binary file is computed using a brief Python programme. (Code avail on page 189).

4.4.3 Online Source Independence Test

This is an algorithm that requires multiple entropy source inputs, typically 2 or 3, and requires those inputs to be statistically independent of each other. So, in an RNG using a multiple-input extractor, it would be appropriate to test that the entropy sources are independent. This can be computed directly. For example, the Python (Code avail on page 195).

4.5 Codes

All the codes written in red are directly aquired from the book

David Johnston Random Number Generators—Principles and Practices

Chapter 5: Conclusion

Chapter 6: Future Work

Future milestones that need to be achieved to commercialize this project are the following.