Qualification of a QRNG and exemplary application in Differential Privacy

Cesare Gerolimetto Fabrello

Advisor: Massimo Caccia
Tutor: Alberto Trombetta

UNIVERSITA' DEGLI STUDI DELL'INSUBRIA

Department of Theoretical and Applied Sciences
Ph.D. in Computer Science and Mathematics of Computation

Qualification of a QRNG and exemplary application in Differential Privacy

Doctoral dissertation of:
Cesare GEROLIMETTO FABRELLO

Advisor: Massimo CACCIA
Tutor: Alberto TROMBETTA

XXXVIII Cycle
2022-2025

"Do you ever feel like Goldilocks
in a world where soup is never
just right?"

Acknowledgements

A heartfelt *thank you* to all the people who supported me in this long and at times difficult journey, from my family to my dearest friends.

Contents

List of Figures

List of Tables

Abstract

This thesis investigates how the quality of randomness affects the privacy guarantees provided by differential privacy mechanisms. We develop and characterize a quantum random number generator based on Silicon Photomultipliers, establishing both offline validation procedures using standard statistical test suites (NIST SP 800-22, NIST SP 800-90B, TestU01) and online health monitoring capable of detecting catastrophic failures in near real-time. Using controlled entropy manipulation experiments, we measure the empirical privacy loss produced by differential privacy implementations under varying randomness conditions. Our statistical framework detects deviations from theoretical guarantees when approximately one bit in every 8 to 16 is manipulated, revealing that differential privacy mechanisms exhibit considerable resilience to subtle entropy degradation compared to cryptographic applications. Additionally, we identify and diagnose artifacts in OpenDP's discrete Laplace sampler, tracing the issue to the Bernoulli sampling procedure, and propose an alternative implementation using exact rational arithmetic that eliminates these artifacts.

Introduction

Keeping sensitive information private isn't exactly a new problem. Consider, for example, Caesar's cipher, used by Julius Caesar to communicate military information. Some more recent approaches to sharing sensitive data while avoiding the disclosure of personal information involves removing personal identifiers such as names, addresses, and so forth. However, with technological advancement and the expansion of web interconnections, and thus the exponential increase of information available online, such approaches have proven insufficient to protect people's anonymity.

Data protection mechanisms, whether deterministic or probabilistic, share one common brick from which they build upon: unpredictability. Thus, along with technological advancements that allow sharing of massive amount of data on the internet, it came the need of producing high quality random numbers, where the term "quality" quantifies how unpredictable these numbers are. This need was also highlighted in some unfortunate cases where low entropy caused serious protection failures of the mechanisms feeding on the underlying faulty source.

Random Number Generators are the tools used to provide entropy. Some are purely algorithmic, and therefore no matter how scrambled the

sequences they produce might look, they will always be deterministic in nature. Others harness entropy from physical processes, guaranteeing genuinely unpredictable randomness. And then there are Quantum Random Number Generators, which exploit quantum phenomena that are inherently unpredictable.

That's the starting point for this thesis. We'll discuss why high-quality randomness matters, describe our QRNG and the physics behind it, and detail how we've built and tested it. We've developed a qualification procedure using established test suites, plus an online monitoring system that can catch catastrophic failures through statistical analysis. We also analyze a practical application: differential privacy.

Differential privacy is particularly interesting because it lets you release data useful for statistical analysis while maintaining rigorous mathematical guarantees about privacy protection. The level of protection is controlled by a quantifiable parameter, giving precise control over the privacy-utility tradeoff. What we want to understand is how the quality of your randomness source affects the privacy guarantees that differential privacy provides.

This thesis is organized as follows. Chapter 1 discusses different types of randomness sources and why high entropy sources matter, including some notable privacy and cryptography failures that happened because of weak randomness. We'll also mention a few cases where predictable randomness is actually useful, though those don't involve data protection. Chapter 2 describes Random Power's QRNG: the physics, how random bits are extracted, and the hardware and software architecture developed by the project team. Our specific contribution lies in the statistical validation framework and quality monitoring procedures described in subsequent chapters. Chapter 3 explains why testing randomness is just as important as generating it, covering well-known statistical test suites and the scoring system we created to evaluate our QRNG against both baseline sources and deliberately corrupted streams where we introduced biases. Chapters 4

and 5 present an online anomaly detection procedure that we developed to monitor the bit-stream in near real-time and that can effectively catch catastrophic failures. Chapter 6 provides background on differential privacy, and Chapter 7 reports our results testing differential privacy with different randomness sources. Chapter 8 includes an alternative Bernoulli sampling implementation that appears to fix an artifact issue we came across in the OpenDP library, an open source project for Differential Privacy written in Rust.

CHAPTER 1

The Modern Role of Randomness

Randomness serves as foundation for modern computing, providing the unpredictability essential for secure, fair, and robust computational systems [5]. Insufficient entropy can undermine cryptographic protocols, introduce bias into simulations, compromise reproducibility in scientific computing, and enable unauthorized disclosure of sensitive data in privacy-preserving systems. The applications of random number generation span virtually every domain of computation: Monte Carlo methods in scientific research, procedural generation in gaming, stochastic optimization in machine learning, secure key generation in cryptography, and noise injection in privacy technologies such as differential privacy [6] [7].

Yet randomness does not come in a single, absolute form: it exists on a spectrum. The key challenge is balancing computational efficiency against the need for unpredictability. Patterns that are harmless (or even useful) in simulations or games can be catastrophic in cryptographic systems, where even minor regularities can compromise security. This raises a fundamental question: what kind of randomness is "good enough" for a given task? A pseudo-random sequence that suffices for a game or a physics simulation might be severely inadequate for generating encryption keys. Conversely, obtaining truly random bits for every operation can be overkill and impractical in less critical applications.

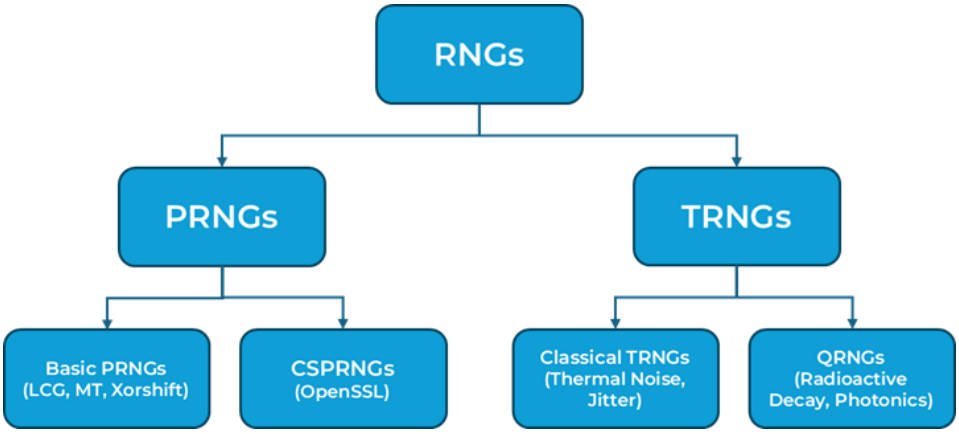


Figure 1.1: Taxonomy of random number generators. RNGs can be broadly divided into algorithmic (PRNGs) and physical (TRNGs) sources, with further subcategories.

This chapter discusses the role of randomness in modern technology, beginning with an overview of how randomness is generated through Pseudo-Random, Cryptographically Secure, and True Random Number Generators, along with hybrid designs that combine them. We then examine the requirements of repeatability and unpredictability across real-world applications: reproducible variability in gaming and Monte Carlo simulations versus uncompromising unpredictability in cryptography, secure communications, and authentication tokens. The discussion highlights both successful practices and notorious failures, including the Debian OpenSSL [8] [9] bug and the `Dual_EC_DRBG` controversy, where flawed randomness undermined entire systems.

1.1 Random Number Generators

Random number generators can be broadly categorized by the source and quality of randomness they provide. There are two main classes of generators: Pseudo-Random Number Generators (PRNGs) and True Random

Number Generators (TRNGs). Each has its role, with different performance characteristics and levels of security. A schematic representation of the macro-categories of RNGs is displayed in Figure 1.1. This section presents each of them, along with the trade-offs involved in choosing one over another.

1.1.1 Pseudo-Random Number Generators

PRNGs are essentially algorithms. They produce sequences of bits that at a first glance may look random, but are the product of deterministic operations, and are entirely dependent on an initial value known as seed. If the PRNG is given the same initial seed, it will produce exactly the same sequence every time, making it reproducible. John von Neumann once joked that *"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin"* [10].

Classical PRNG algorithms include linear congruential generators, the Mersenne Twister, Xorshift [11], and various linear feedback shift register (LFSR) based methods. Modern algorithms can produce very long sequences with good statistical properties; for example, the Mersenne Twister has a period of $2^{19937} - 1$ and has been widely adopted for simulations and games due to its speed and uniformity [12]. Similarly, Xorshift generators trade a bit of statistical quality for extreme speed, using simple bitwise operations to generate streams of numbers suitable for non-critical randomness needs.

The advantages of PRNGs lie in their efficiency and repeatability. They are much faster than true random sources and can generate large volumes of random numbers with minimal computational cost. They also allow algorithms and experiments to be run with the same "random" conditions multiple times by reusing the seed, which is invaluable for debugging and testing purposes, along with reproduction of scientific results.

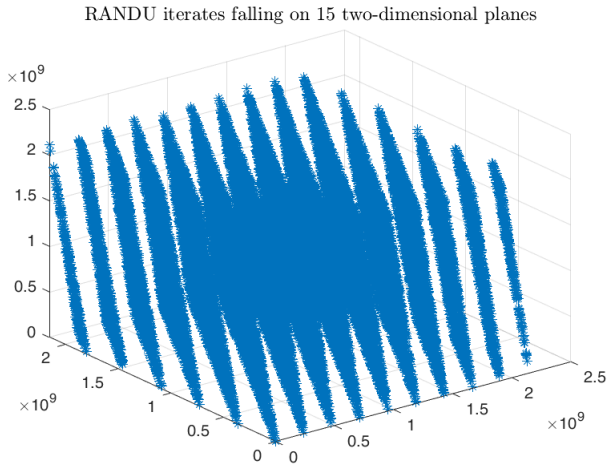


Figure 1.2: Three-dimensional plot of RANDU iterates. Each point represents 3 consecutive samples generated by the PRNG, and lies on one of 15 parallel two-dimensional planes.

This condition is also their limitation. If an attacker learns the seed or internal state of the PRNG, the entire future output can be predicted. This is a fatal flaw for any security-sensitive application. Even if the seed is secret, many PRNGs are not designed to withstand cryptanalytic attacks. An adversary who observes a portion of the output might be able to infer the generator's state and predict subsequent outputs. For instance, the classic linear congruential generators can be solved after observing a few outputs due to their linear structure. Moreover, PRNG sequences eventually repeat (due to finite state). A good algorithm has an extremely large period, so repetition is not a practical concern for typical usage. But shorter-period generators can cycle within feasible time. A well-known example is the RANDU generator from the 1960s, which had a period of only 2^{29} for odd seeds (despite using modulus 2^{31}), and produced visible patterns, as shown in Figure 1.2 [13].

Cryptographically Secure PRNGs (CSPRNGs)

A CSPRNG is a pseudo-random number generator with additional properties that make it suitable for cryptographic use. Informally, a CSPRNG's output should be indistinguishable from true random noise to any polynomial-time adversary, even one who knows the algorithm and has seen a portion of the output. More concretely, a secure PRNG must pass stringent statistical tests (like those in NIST SP 800-22 [14]) and also resist prediction attacks. The German BSI (Federal Office for Information Security) defines criteria K1–K4 for random generators, of which K3 and K4 require that it be impossible to predict future output even if some internal state or past outputs are known [15]. In other words, a CSPRNG has the property that knowledge of its current state (or a segment of its output) is not enough to reconstruct either its previous outputs or its future outputs. This property means that even if part of a system is compromised, the random values it produced earlier or will produce later remain safe.

CSPRNGs are typically built from cryptographic primitives such as ciphers or hash functions. For example, NIST's SP 800-90A standard specifies several approaches: one based on HMAC (HMAC-DRBG), one on hash functions (Hash-DRBG), and one on block ciphers in counter mode (CTR-DRBG) [16]. These designs take an initial seed and then generate an arbitrary stream of outputs by repeatedly applying the cryptographic function, occasionally re-seeding with new entropy.

In practice, a good CSPRNG provides a nice balance between performance and security. They are slower than simple PRNGs because cryptographic computations are more expensive than arithmetic or bitwise operations, but modern CPUs are fast enough that CSPRNGs can often be used for many purposes without a noticeable loss in performance. For example, generating a few kilobytes of random data for a key or a token is trivial on today's hardware using a CSPRNG. However, if an application requires

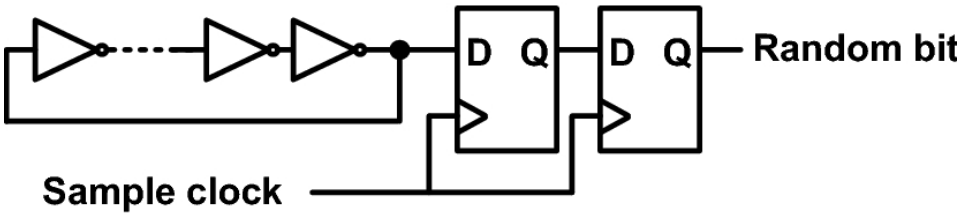


Figure 1.3: Circuit schematic of a basic ring oscillator TRNG model [1].

gigabytes of random data per second (like Monte Carlo simulations), using a CSPRNG for the entire stream might be impractical.

Operating systems expose vetted CSPRNG interfaces such as Linux's `/dev/urandom` and Windows' `BCryptGenRandom` [17]. However, trust in the algorithm is critical. Developers are strongly encouraged to use these vetted sources rather than building their own random generators.

1.1.2 True Random Number Generators

In contrast to PRNGs, a True Random Number Generator relies on a physical process to produce entropy. These sources exploit inherently unpredictable phenomena in the physical world: examples include thermal noise in electronic circuits, radioactive decay, quantum phenomena, and so on. Because they are based on stochastic physical phenomena, TRNG outputs are not predetermined and not reproducible, even if one could somehow "reset" the system: rerunning the same physical process will produce different results. TRNGs are also known as Hardware Random Number Generators because they typically need dedicated hardware components, such as ring oscillator circuits, noise diodes, or sensors to capture the entropy [18].

The advantage of TRNGs is that they provide truly unpredictable bits that are not vulnerable to algorithmic prediction or backdoors. They are the gold standard for seeding cryptographic applications. For instance, Hardware Security Modules (HSMs) typically include a small TRNG to

generate keys and seeds internally. Modern CPUs often come with instruction sets for random number generation (e.g., Intel's `RDRAND` and `RDSEED` instructions) that pull from internal entropy sources such as metastability in logic circuits or thermal noise amplifiers. Research into TRNGs remains active, with efforts to improve their speed and reliability. For example, a recent design by Frustaci et al. (2024) demonstrates a high-speed, low-power TRNG architecture suitable for FPGA implementations [19].

However, TRNGs come with trade-offs and downsides. Firstly, physical entropy sources are generally slower than algorithmic generators. They might produce bits at rates ranging from a few kilobits per second up to maybe a few hundred megabits in high-end devices, which is orders of magnitude slower than a PRNG that can generate up to many gigabytes per second. If an application requires a large volume of random data quickly, a pure TRNG may become a bottleneck.

Secondly, real-world entropy sources can be subject to biases and correlations. For example, an electronic noise source might change with temperature or age, or a quantum source might have detector biases. TRNG outputs often need "whitening" or post-processing (e.g., using a hash or XORing multiple samples) to remove biases and ensure a uniform distribution of outputs [20]. Moreover, hardware can fail: there have been instances of supposed hardware RNGs that, due to design or manufacturing flaws, output a constant value or a very limited set of values (for instance, a notorious bug in an AMD processor's RDRAND instruction returned repeated values under certain conditions, which had to be mitigated at the software level [21, 22]). Therefore, TRNG designs usually incorporate health checks; for example, the NIST standards recommend continuous tests that ensure the noise source has not failed catastrophically (e.g., checking that not too many 0s or 1s appear in a row) and periodic statistical tests on output blocks [14].

Finally, high quality TRNGs often require specialized hardware, such as analog circuit components or photonic devices, that cost more and add

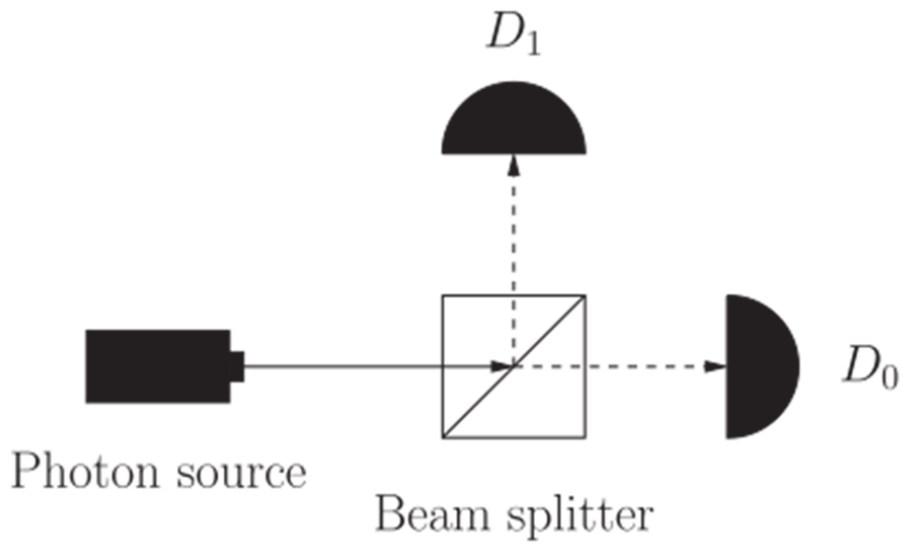


Figure 1.4: Optical QRNG: a single photon emitted by the source encounters a 50/50 beam splitter and is detected by either D_0 or D_1 with equal probability. Since quantum mechanics principles prevent predicting which detector fires, each detection event provides one genuinely random bit, with $D_0 \to 0$ and $D_1 \to 1$, yielding a simple mapping to a binary bit-stream.

complexity compared to purely digital logic. Integrating a TRNG into a system requires hardware support and often calibration. Not all platforms have built in TRNGs, especially older or cheaper microcontrollers. In these cases, developers sometimes try to use whatever physical jitter is available (like CPU clock variations or uninitialized memory), but this can be risky if not done properly [23].

QRNGs

Quantum Random Number Generators (QRNGs) exploit inherently probabilistic quantum phenomena, such as the detection of single photons (as seen in Figure 1.4), phase noise in lasers, or radioactive decay, to produce

entropy [24]. Unlike classical noise sources, whose unpredictability might ultimately reduce to unmodeled complexity, quantum processes are understood to be fundamentally stochastic (as described in Chapter 2). This gives QRNGs a particularly strong claim to unpredictability: their outputs cannot, even in principle, be reproduced or predicted by any observer.

QRNGs have moved in recent years from laboratory prototypes to commercially available devices, ranging from standalone embodiments to integrated chips for mobile and embedded platforms. Their promise lies in offering randomness rooted in physics, with no known method to bias or foresee the outcomes [25]. Even by an adversary with large-scale quantum computing power will not be able to predict the outputs, making QRNGs quantum-attack resistant. This makes them attractive for high-assurance cryptographic key generation, secure communications, and privacy-preserving data processing.

Nonetheless, QRNGs also face practical challenges. Photon detectors and optical setups can introduce biases due to imperfections in alignment, efficiency, or noise in electronics, possibly requiring careful post-processing to ensure uniformity. Throughput and cost remain limiting factors compared to algorithmic generators, though ongoing advances in hardware integration are mitigating these aspects. In practice, QRNG outputs are often combined with algorithmic CSPRNGs in a hybrid architecture, seeding cryptographic generators with quantum entropy while preserving efficiency and scalability.

This is formalized in NIST SP 800-90B [4], which provides practical specifications for the implementation of Deterministic Random Bit Generators (DRBGs) seeded directly by a validated entropy source such as a QRNG, ensuring that the resulting bit-stream meets rigorous cryptographic standards while maintaining high throughput. This hybrid approach exemplifies a key theme across all random number generators: the complementary nature of different RNG types, each optimized for specific requirements.

1.1.3 Practical Guidelines for RNG Selection

Given this taxonomy of random number generators, practical guidelines emerge for selecting appropriate randomness sources. Basic PRNGs should only be used in contexts where reproducibility and speed matter more than unpredictability. They provide sufficient randomness for tasks such as rendering graphics, generating game worlds, or performing Monte Carlo simulations, but they must not be employed in cryptographic or security-sensitive settings.

By contrast, TRNGs and QRNGs are appropriate for cryptography and security applications, though their limited throughput may become a bottleneck in large-scale deployments.

Rather than being viewed as competitors, PRNGs and TRNGs are best understood as complementary. In practice, hybrid systems rely on TRNGs (or QRNGs) to supply secure seeds, which are then expanded by fast PRNGs acting as entropy amplifiers. This combination profits both from the unpredictability of physical sources and the efficiency of algorithmic generation, ensuring both security and scalability.

1.2 Randomness in Simulations and Gaming: Repeatability

In domains like scientific computing, simulations, and gaming, randomness is often used to introduce variability while still allowing repeatability when needed. In these scenarios, PRNGs with fixed seeds are commonly employed. By using a known seed value, one can ensure that a sequence of "random" events can be reproduced exactly, which is invaluable for debugging, testing, or comparing results. For example, a game developer might want the same procedurally generated map layout to appear every time a particular seed is used, so that players or testers can reliably encounter the same scenario. Similarly, scientific Monte Carlo simulations, which are widely used to evaluate complex integrals or model stochastic processes,

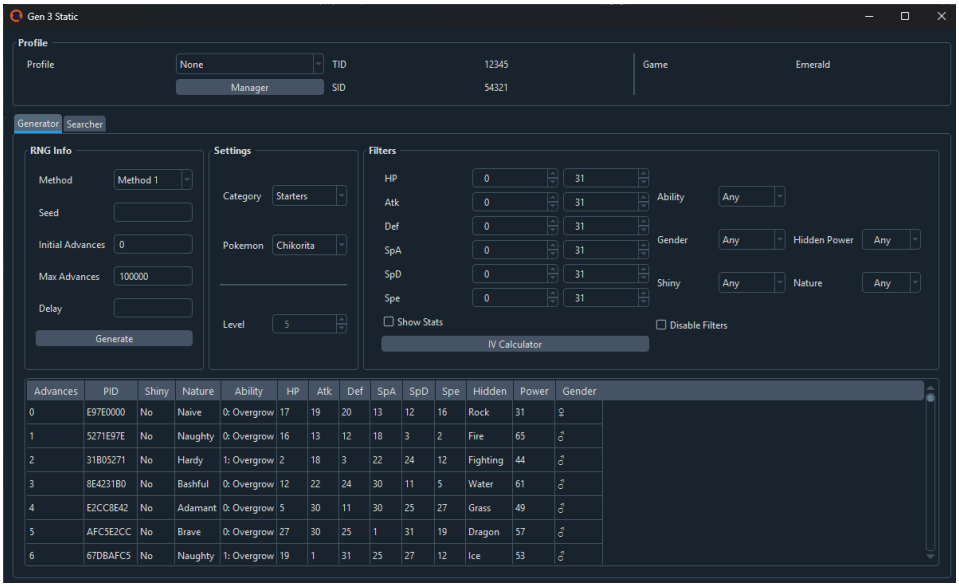


Figure 1.5: Exemplary image showing a tool able to compute, for each in-game frame (column "Advances"), the corresponding personal ID and game values, due to the known initial state of 0 of the LCG in Pokemon Emerald after each console reset.

often rely on PRNGs and allow runs to be replicated by reusing the initial seed [6]. This reproducibility is crucial for verifying results and performing comparative studies.

A notable example is procedural content generation in video games. Titles like Minecraft, Valheim, or No Man's Sky, which generate vast worlds algorithmically, use a seed-based PRNG to ensure that a given seed always produces the same world. This approach lets developers and players share worlds or replay the same content consistently. The randomness provides variety and surprise, but the determinism via seeds ensures control. In game design, in fact, perfectly uniform randomness may not even be desirable: developers sometimes tweak random mechanics to avoid extreme bad-luck streaks or other outcomes that would frustrate players. In short, games

often require controlled randomness: random enough to provide variability and challenge, yet sufficiently deterministic to allow fairness, balance, and repetition when needed. PRNGs satisfy these needs by producing sequences that appear random and have statistically uniform distributions, while being completely determined by their initial seed.

Another instructive case comes from the Pokémon series. In Pokémon Emerald (2004, Game Boy Advance), due to a programming error, the built-in PRNG initial seed was set to 0 every time the console was powered on, and the generator itself was a simple linear congruential algorithm. This design made in-game randomness highly predictable. Players quickly discovered techniques known as "RNG manipulation", where by carefully timing button presses after resetting the console, they could control ostensibly in-game random events such as hidden values and battle outcomes. Far from being random, these events became deterministic and reproducible (Figure 1.5 depicts a tool used to compute hidden values generated on each frame of the game), showing how weak PRNG design and shallow seeding strategies enable easy prediction and exploitation by end users [26, 27].

Monte Carlo methods in computational science likewise benefit from PRNGs' efficiency and repeatability. These simulations demand a stream of random samples (e.g., uniform $[0, 1]$ numbers) to model probabilistic events or perform numerical integration. An example is shown in Figure 1.6, where an approximation of π is computed by mapping random samples within a square and its circumscribed circle and taking the ratio of the cumulative sums. Quality requirements here pertain to statistical properties: the PRNG should produce values that are independent and uniformly distributed, as any bias or correlation could skew the simulation's outcomes. The same statistical test suites used to evaluate CSPRNGs apply here, though without the additional cryptographic security requirements. What matters is that the pseudo-randomness passes tests for uniformity and independence over the timescale of the simulation. In fact, many basic PRNGs that were once

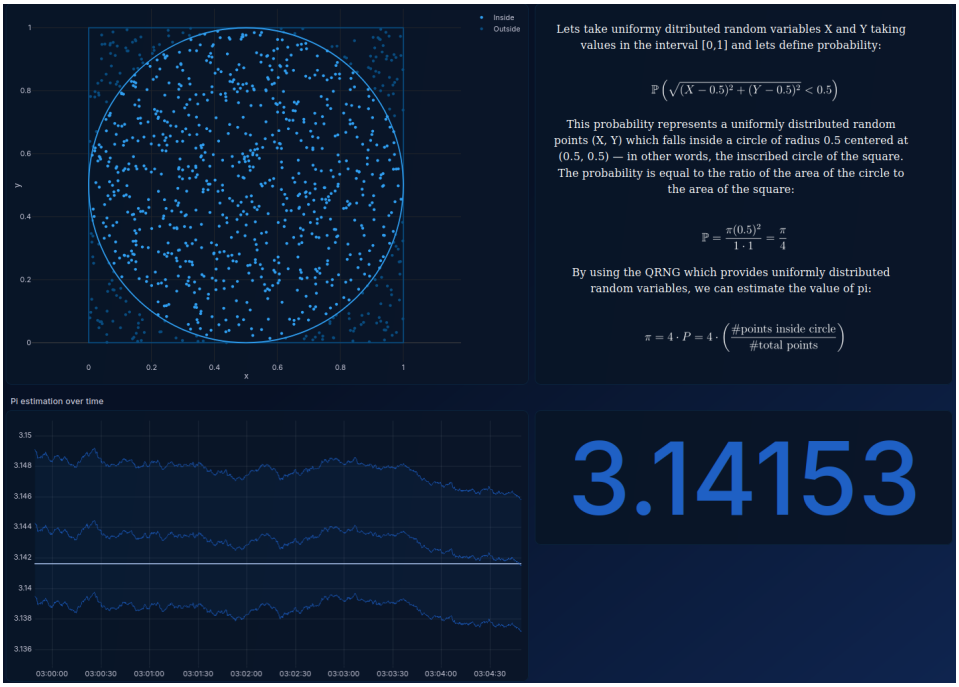


Figure 1.6: Monte Carlo simulation for the computation of π. Generated values are mapped within the square and circle areas, and the ratio of the cumulative sums is used for the estimation of the value.

common would now fail these tests given the power of modern computers to detect periodicity or correlations.

In practice, widely-used PRNG algorithms such as the Mersenne Twister are popular for Monte Carlo work due to their long period (e.g. $2^{19937} - 1$) and excellent distribution characteristics [12]. These generators are extremely fast and have been tuned to avoid detectable biases in simulation contexts. On the other hand, simpler generators like linear congruential generators (LCGs), while very fast, may have short periods or noticeable patterns and thus are generally avoided for large-scale simulations.

Through careful selection and testing of PRNGs, scientific applications can achieve the necessary balance of performance and randomness quality.

Equally important, by recording the PRNG seed one can exactly reproduce any Monte Carlo experiment, important for scientific transparency and debugging.

1.3 Randomness in Security and Cryptography: Unpredictability

In security-critical applications, the fundamental challenge shifts from the reproducible variability needed in simulations to absolute unpredictability. As established earlier, this unpredictability, or entropy, must resist any algorithmic prediction, even by adversaries who know the generation method and observe partial outputs. The consequences of failing this requirement are severe, as demonstrated by numerous real-world security breaches where predictable randomness became the system's weakest link. Some common security-sensitive uses of randomness include:

- **Cryptographic key generation:** Keys for encryption algorithms (AES, RSA, etc.) must be chosen from a vast space of possibilities such that an attacker cannot guess them. A 128-bit AES key, for instance, should be generated by a process that gives each of the 2^{128} possibilities an equal chance, with no bias. If the key is even slightly biased or has fewer than 128 bits of entropy, the effective security can be much lower than intended [28].

- **Nonces, IVs, and salts:** Many protocols need random nonces (numbers used once), initialization vectors for encryption modes, and salt values for password hashing. These prevent replay attacks and stop precomputation attacks like rainbow tables by making each session or operation unique. They need not be secret in all cases, but they must be unpredictable and never reused. Problems like predictable TCP sequence numbers or reused ECDSA nonces can create serious vulnerabilities [29].

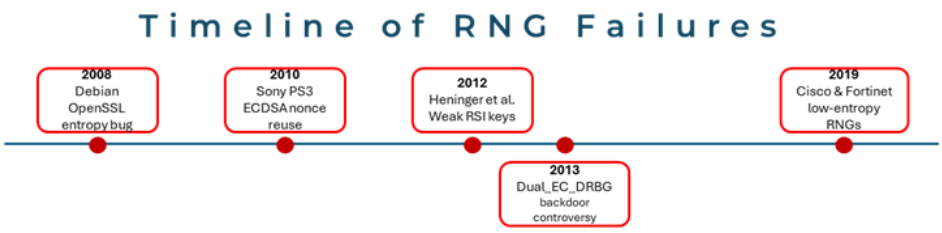


Figure 1.7: Timeline of notable RNG failures: elected case studies where weaknesses in random number generation led to major security compromises: Debian OpenSSL entropy bug (2008), Sony PS3 ECDSA nonce reuse (2010), weak RSA/DSA keys discovered by Heninger et al. (2012), Dual_EC_DRBG controversy (2013), and Cisco/Fortinet entropy flaws (2019).

- **Session tokens and authentication challenges:** Web applications and identity systems commonly issue random session IDs or tokens for authentication. These tokens act as temporary keys that grant access, so they must be impossible to guess. A user's session cookie, for example, should be a random 256 bit or 128 bit value generated by a secure random generator; otherwise, an attacker who can predict the token could hijack sessions [30].

In all the above cases, randomness quality is directly tied to security. As discussed in Section 1.1.1, CSPRNGs must satisfy the stringent indistinguishability requirement (i.e., their output should be computationally indistinguishable from true random noise). This cryptographic strength is essential because attacks on systems often target random number generation as a potential weak link, and even theoretical security proofs can be undermined by real-world randomness flaws [5]. As it will be outlined in the following paragraphs and illustrated in Figure 1.7, the consequences of inadequate randomness have been repeatedly demonstrated through serious security breaches spanning over a decade of cryptographic implementations.

In 2008, a modification to Debian's OpenSSL package disastrously reduced the entropy used for key generation. The only seed input to the OpenSSL PRNG became the process ID, resulting in only 2^{15} possible keys [31]. All cryptographic keys (SSH keys, SSL/TLS certificates, etc.) generated on affected Debian-based systems for a period of 2 years were predictable, leading to a massive compromise of security [8] [9].

In 2010, the Sony PlayStation 3 console's security was fundamentally broken due to a randomness flaw. Sony's firmware updates were signed with ECDSA (Elliptic Curve Digital Signature Algorithm); however, the implementation reused the same pseudo-random nonce in multiple signatures. This nonce repetition allowed hackers to solve for the private signing key, resulting in a complete jailbreak of the console [32]. The root cause was a failure to generate a fresh unpredictable random number for each signature.

In 2012, researchers Heninger et al. scanned public keys on the Internet and discovered numerous devices (especially embedded and network appliances) that shared prime factors in their RSA keys or repeated DSA/ECDSA keys [30]. In many cases, this was traced to insufficient entropy during key generation (for example, devices that generate keys on first boot with uninitialized PRNG state). The result was that many public keys could be easily broken, as the presence of shared factors allowed the researchers to factor RSA moduli and recover private keys.

More recently, in 2019, both Cisco and Fortinet disclosed high-profile vulnerabilities due to low-entropy random number generation in their products. In Cisco's case, a hardware random number generator in certain ASA and FirePower security appliances was not properly seeded, undermining the security of cryptographic operations [33]. Similarly, a vulnerability in Fortinet's FortiOS (CVE-2019-15703) meant that cryptographic keys and session tokens generated by its PRNG could be predicted by an attacker who observed enough outputs [34]. These incidents show that even in modern systems, entropy failures can slip past and remain unnoticed until

a catastrophic weakness is revealed.

Even standards and algorithms themselves have come under scrutiny. A notable controversy was the NSA-designed `Dual_EC_DRBG` algorithm, which was standardized by NIST as a cryptographically secure PRNG. Researchers found that `Dual_EC_DRBG` had a subtle mathematical relationship that could serve as a backdoor: whoever generated certain internal constants could potentially predict future outputs of the PRNG after observing a small sample. In 2007, cryptographers first pointed out this issue [35], and later revelations (in the wake of the Snowden leaks) suggested that the NSA may have engineered those constants to its advantage [36, 37]. By 2013 NIST and industry experts recommended abandoning Dual_EC and it was removed from standards [38]. This episode underscored that not only implementation bugs but even intentionally introduced weaknesses in randomness generators can undermine security, and it reinforced the importance of using well-vetted, trustworthy algorithms in cryptographic practice.

These examples all highlight a common theme: cryptographic security cannot withstand weak or predictable randomness. Attackers will exploit any deterministic patterns to undermine encryption, forge signatures, hijack sessions, or otherwise breach systems. As computing power grows, the bar for what constitutes "securely random" also rises, since patterns or biases that might have been infeasible to detect or exploit years ago may become exploitable with modern techniques. Furthermore, the advent of quantum computing amplifies concerns: for instance, Grover's algorithm can quadratically speed up brute-force searches, effectively halving the bit-strength of keys. Simply increasing key sizes (e.g. from 128-bit to 256-bit keys) is not sufficient if the entropy used to generate those keys is inadequate. Ultimately, it is the quality of the entropy behind cryptographic keys and secrets, rather than just their nominal size, that determines the true security level of a system [28]. Ensuring high entropy in all security-critical

random values is therefore absolutely essential.

Beyond the generation process itself, rigorous testing of randomness quality is equally critical. Several test suites were devised to analyze the statistical properties of generated bit-streams, to detect possible hidden biases, dependencies, or temporal correlations. More details on these methodologies are provided in Chapter 3.

1.4 Summary

Random numbers are indispensable to modern cryptography and security, but their trustworthiness depends critically on how they are generated and evaluated. While deterministic generators offer speed and reproducibility, they lack intrinsic unpredictability. TRNGs, by contrast, harness physical noise sources. Yet, RNG reliability cannot be taken for granted: poor designs and inadequate evaluation have historically led to severe vulnerabilities. Therefore, the study of entropy sources, their characterization, and their robustness against bias or manipulation is essential.

The next chapter will focus on a concrete embodiment of a Quantum Random Number Generator (QRNG), analyzing its architecture and the underlying physical principles. The subsequent chapter will then address the crucial aspects of entropy validation and health testing, providing the framework needed to ensure continuous trust in the generator's output.

CHAPTER 2

Random Power!'s QRNG

The *In-silico Quantum Generation of Random Bit Streams (Random Power!)* project addresses the need for high-quality entropy sources by exploiting the stochastic nature of avalanche processes in Silicon Photomultipliers (SiPMs). These devices are essentially arrays of p-n junctions that, biased beyond breakdown voltage and operated in complete darkness, spontaneously generate pulses seeded by direct or trap-assisted energy band tunneling. The resulting avalanches, amplified to easily detectable levels, form the basis of a compact, scalable, and certifiable entropy source. The underlying principle of randomness extraction from SiPM dark pulses is protected by an international patent (WO2020070641), granted in Italy, United States, European Union, China, Japan, and South Korea [39].

This chapter presents the Random Power! platform, developed collaboratively by the project team. The hardware design, FPGA firmware implementation, and TDC development were led by other team members. This thesis contributes specifically to: the statistical characterization and validation methodology presented in Chapters 3, 4, and 5, the design and analysis of on-line monitoring procedures, and elements of the software architecture for bit-stream delivery and quality assessment described in Section 2.3.2. The comprehensive system description provided here establishes the necessary context for the statistical testing procedures and differential

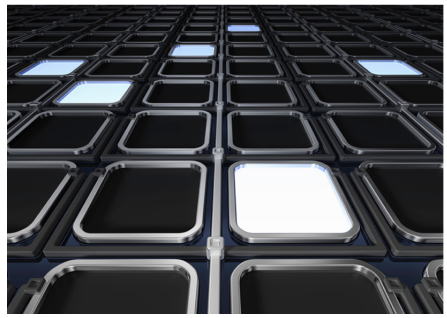


Figure 2.1: SiPMs may be seen as collections of binary cells, which are fired when a photon is absorbed.

privacy analysis that follow. After presenting the physical principle underpinning randomness extraction, we describe the TDC and firmware implementation, followed by three hardware embodiments: the *Single Generator Board*, a compact demonstrator developed in the ATTRACT Phase I project, the *64x Generator Board*, a scalable architecture designed for infrastructural applications, and the ASIC (Application Specific Integrated Circuit), a fully integrated single-chip solution combining entropy generation, cryptographic post-processing, and Root-of-Trust functionality for deployment in resource-constrained and security-critical environments.

2.1 Physical Principle

The quantum random number generator developed here is based on the intrinsic stochastic processes that occur in semiconductor junctions. The core element is a Silicon Photomultiplier (SiPM), an array of p–n junctions.

When operated under normal conditions, SiPMs function as highly sensitive photon detectors: whenever the photon hits the cell, it triggers a cascading effect (known as avalanche breakdown) due to the high electric field, producing a large, easily detectable current pulse. The amplitude of this pulse increases with the number of photons detected, effectively giving

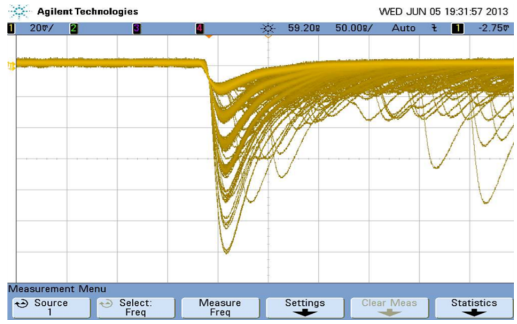


Figure 2.2: Oscilloscope traces of SiPM output pulses, with multiple acquisitions overlaid to reveal the discrete amplitude levels corresponding to different numbers of simultaneously fired cells. Since each cell contributes a fixed charge, the total pulse amplitude scales linearly with the number of fired cells. The spread of traces at each level reflects the stochastic nature of the underlying avalanche process.

it single-photon sensitivity.

However, for quantum random number generation, the device is instead operated beyond its breakdown voltage and kept in complete darkness. Under these conditions, no external photons are required to trigger avalanches [40]. Instead, random charge carriers are spontaneously generated within the semiconductor material due to various quantum-mechanical processes at the atomic scale. These carriers originate from several microscopic processes (including thermally driven trap-assisted generation and recombination, re-emission of carriers from metastable states, field-assisted emission, and direct band-to-band tunneling) [41, 42]. Each of these mechanisms has a fundamentally quantum nature, tied to the structure of semiconductor energy bands and the Fermi–Dirac distribution of electrons, and together they provide a steady supply of potential seeds for avalanches. SiPMs can be imagined as a matrix of binary cells, as shown in Figure 2.1.

When a single carrier enters the high-field region or conduction band, it can initiate an avalanche breakdown, releasing millions of electrons within tens of nanoseconds. The resulting pulses (shown in Figure 2.2) have large

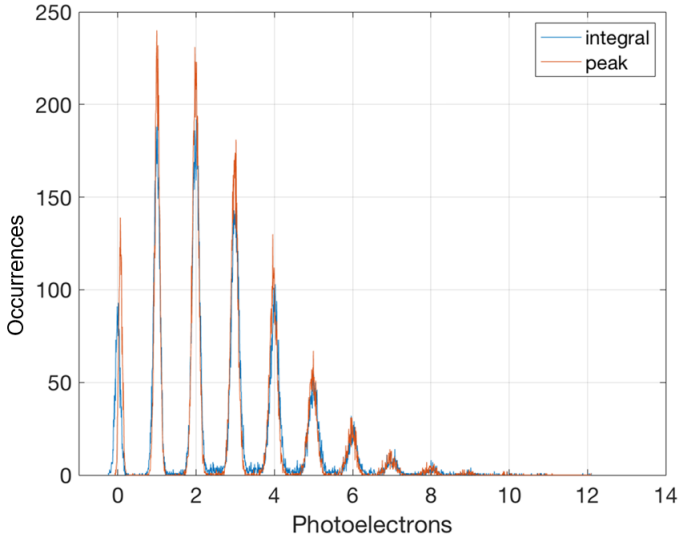


Figure 2.3: Histogram of the response to a high statistics of low intensity light pulses.

amplitudes, easily distinguishable from electronic noise thanks to the large gain, and exhibit rise times at the sub-nanosecond scale, enabling precise time tagging.

This high intrinsic gain also assures excellent photon number resolving capability. Such capability is clearly illustrated by the charge spectrum, called multiphoton peak spectrum, reported in Figure 2.3. Each entry corresponds to the integrated signal over a time window. The well-separated peaks represent different numbers of fired cells. The peak-to-peak distance Δ_{pp} is proportional to the single-cell gain, growing linearly with the applied overvoltage, and provides the calibration factor to convert the digitized charge into an equivalent number of photoelectrons [43].

Since the seeding events occur independently and is distributed according to the Poissonian statistics, the time series of avalanches constitutes a fundamentally unpredictable entropy source. By recording the arrival times of pulses and converting them into binary symbols, the device produces a

random bit stream with high efficiency: at room temperature and a moderate overvoltage, pulse rates reach the MHz/mm² range, corresponding to compact devices with extraction efficiencies of around 40% (four bits generated for every nine pulses).

The complete bit-generation pipeline (Figure 2.5) proceeds as follows. Avalanche pulses are spontaneously generated by the SiPM. The analog output is digitised by a fast comparator, and each detected pulse is time-tagged by the TDC (at sub-nanosecond resolution). The resulting sequence of timestamps is then processed by a simple extraction algorithm that compares successive inter-arrival time intervals and maps them into 4-bit symbols, producing the final random bit-stream.

A critical requirement for the validity of this approach is the statistical independence of successive avalanches. While in principle each avalanche is a spontaneous and independent event, SiPMs are known to suffer from afterpulsing, where carriers trapped during a primary avalanche are released at later times, or secondary photons generated in the avalanche region retrigger nearby cells. These effects create correlations on short time scales that can compromise the purity of the randomness, especially over very long sequences. In the proposed device, this issue is overcome by introducing a carefully chosen hold-off time in the pulse tagging electronics, which effectively suppresses afterpulsing while preserving the natural stochasticity of the primary processes. As a result, the raw bit-stream generated from the system passes randomness tests without requiring whitening or other post-processing techniques.

2.2 TDC and Firmware Implementation

The hardware platform centers on a proprietary Time-to-Digital Converter (TDC) fully embedded within the FPGA. This integrated approach replaces the discrete commercial TDCs used in early embodiments, providing higher

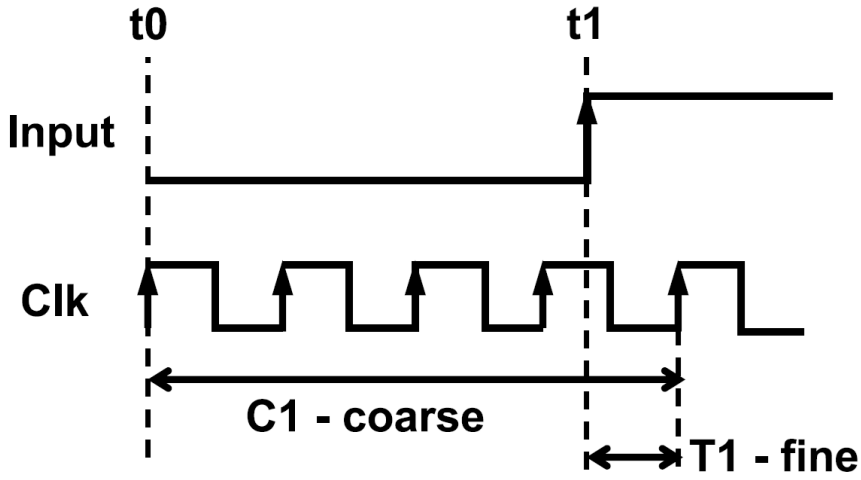


Figure 2.4: Timing diagram of the TDC measurement [2]. The coarse counter C_1 counts complete clock cycles between t_0 and the input transition at t_1, while the fine counter T_1 measures the residual sub-clock interval using a delay line, together achieving sub-nanosecond time resolution.

bit rates, reduced complexity, and improved flexibility through firmware reconfigurability.

The custom TDC implementation FPGA combines a delay line and coarse counter operating in the fast clock domain. The delay line, used to measure the fine value, utilizes carry chain blocks (Carry4) where input signals propagate through multiplexers, creating a multi-tap delay chain that covers more than one clock cycle. Time measurement follows the relationship

$$t_1 = C_1 \cdot T_{Clk} - \frac{F_1}{F_{max}} \cdot T_{Clk}, \tag{2.1}$$

where C_1 represents the coarse counter cycles, F_1 is the number of delay taps the signal traverses, and F_{max} is the total taps per clock period. Fine timing counters are corrected using calibrated delay values stored in look-up tables. Figure 2.4 illustrates the TDC timing principles, showing how the coarse counter C1 measures complete clock cycles while the T1 captures

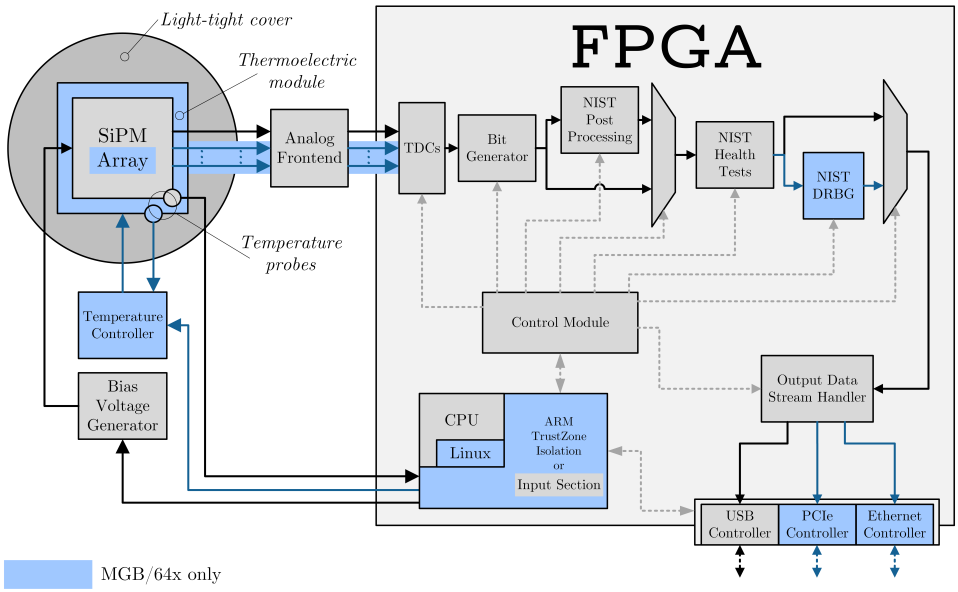


Figure 2.5: Block diagram of the QTRNGs architecture developed by Random Power! [2, 3]. The pulses are generated by the SiPM, with dedicated temperature control. A dedicated analogue circuitry (substituted by a dedicated chip in the 64x) implements the triggering logic, generating a digital signal that is fed directly into the FPGA. Inside the FPGA, it is timestamped by the TDC, and the timestamps are then analyzed by the Bit Generator to produce 4-bit symbols. Optional NIST post-processing, health tests, and DRBG modules operate. Data is managed by CPU/Linux and output via USB, PCIe, or Ethernet. The main difference between the 64x and the SGB is highlighted in light-blue.

fine timing measurement.

The TDC operates in three distinct modes: startup calibration calculates and stores individual tap delays after power-up; on-line calibration continuously recalibrates without interrupting measurements to compensate for temperature and voltage variations; and normal measurement mode captures timing data through cross-domain FIFOs for processing by the embedded Microblaze processor. This calibration strategy ensures

tap uniformity and addresses systematic effects in the carry chain. The full characterization of the TDC, including resolution measurements and systematic tests, is reported in [2].

The FPGA firmware handles bit extraction, health monitoring for quality assurance, and communication tasks. Real-time NIST tests (MONOBIT, RUNS, Adaptive Proportion Test, Repetition Count Test) monitor the bit stream continuously, while integrated SHA-256 whitening processing handles certification requirements and application-specific needs.

2.3 Embodiments

The QRNGs developed are implemented in three main embodiments: an ASIC (Application-Specific Integrated Circuit), a Single Generator Board, and its natural enhanced version: the 64x Generator Board. Table 2.1 lists specifications for the two boards, while Figure 2.5 shows the architecture block diagram, where the main differences between the two boards are highlighted in blue. These systems were developed as part of the *In-silico Quantum Generation of Random Bit Streams (Random Power!)* project, funded by the European Union's Horizon 2020 Research and Innovation Programme within the ATTRACT cascade grant (contract no. 101004462).

2.3.1 Single Generator Board

The Single Generator Board (SGB), shown in Figure 2.6, represents Random Power!'s first complete entropy generation system, developed during the ATTRACT Phase I project. The 8×3.5 cm^2 board, which is roughly half a credit card's size, contains the entropy source and all post-processing electronics.

A Hamamatsu S13360-1350 Silicon Photomultiplier serves as the quantum entropy source, with an integrated temperature sensor for monitoring. The high-voltage supply operates between 20-100 V, while feedback com-

	Single Generator Board	64x Generator Board
Dimensions [cm^2]	8×3.5	$11.1 \times 31.2 \times 2.0$
No. Generators	1 array	64 arrays
Raw Bit-stream	100 kbps	32 Mbps
NIST DRBG Output (SP800-90 A, B, C)	NA	1 Gbps
Control	Xilinx Spartan 7	Xilinx KRIA K26 SOM
I/O	USB or bits-on-pin	Eth or PCI-Express
Power Supply	USB (5V, 0.5A)	12V, 8A
Power Consumption	<2.5W	20W
Encryption of the bit-stream	No	Encryption and Authentication (AES256)
Specific Features	· Firmware implemented on-line sanity checks (Monobit, RUNs)	· Firmware implemented on-line sanity checks (Monobit, RUNs, APT, RCT)
	· Auxiliary post-processing through SHA256 function	· Run control through Trusted Execution Environment
		· Temperature control through Peltier cooler
		· FIPS-140-3 compliant by design

Table 2.1: Table of specifications for the two embodiments of Random Power's QRNG: the Single Generator Board and the 64x Generator Board.

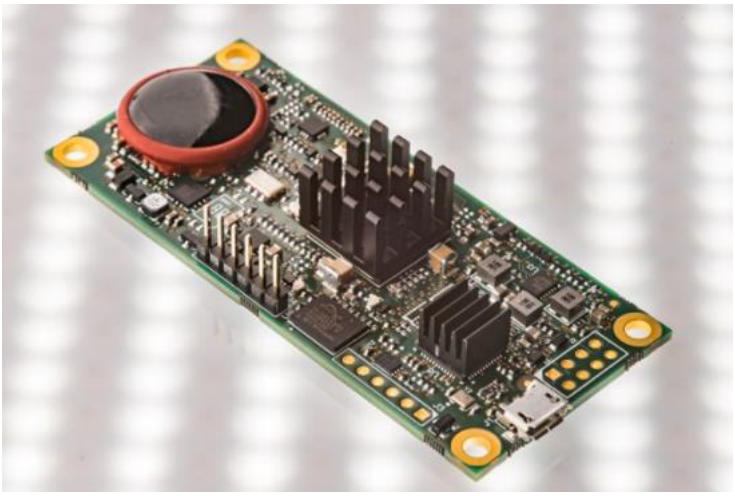


Figure 2.6: The Single Generator Board developed during ATTRACT Phase I project. With its compact 8×3.5 cm^2 design, it delivers a continuous bit-stream of ~ 100 kbs.

pensation maintains stable gain across temperature variations.

Signal processing is handled by a fast comparator that digitizes the SiPM output, followed by the TDC that records pulse arrival times. Early versions used a commercial TI TDC7200, which provided good resolution but limited bit rates to approximately 20 kb/s. The current design uses the proprietary embedded FPGA-based TDC to overcome this limitation, boosting entropy generation rates up to 100 kb/s.

The processing pipeline time-stamps incoming pulses, digitizes inter-arrival intervals, and applies a bit extraction algorithm that converts nine-pulse sequences into 4-bit symbols, creating 16 possible outputs. MONOBIT and RUNS tests verify bit-stream quality immediately after generation. GPIO and JTAG interfaces support debugging operations, while an FTDI chip is responsible for the delivery of the stream via USB. Therefore, the board provides a compact, stand-alone entropy generator with embedded calibration, extraction, and near real-time quality monitoring.

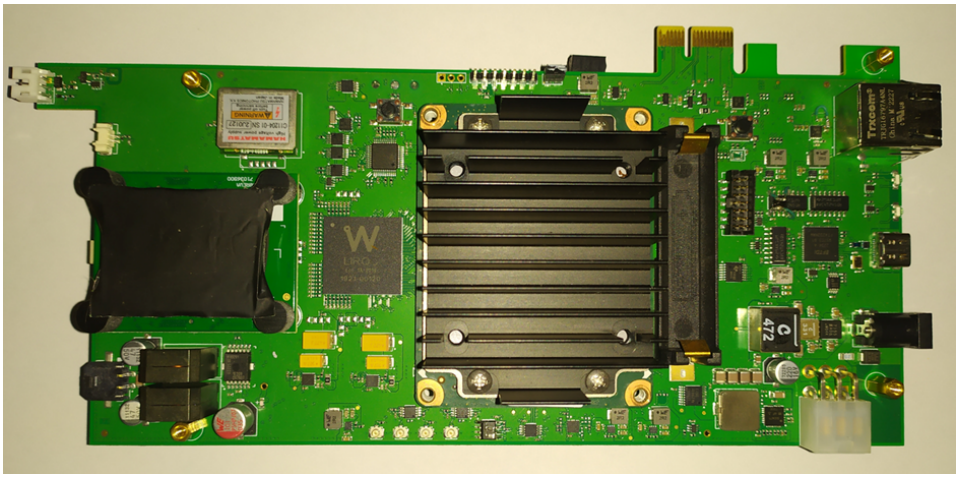


Figure 2.7: The 64x Generator Board embodiment. With its scalable architecture, it delivers a continuous bit-stream of ~ 32 Mbs.

2.3.2 64x Generator Board

To demonstrate scalability for infrastructural applications, the next step of the project was to develop a multi-generator architecture, the 64x Generator Board. This embodiment extends the single-board principle to an array of 64 parallel SiPM-based generators, integrated with a front-end ASIC and high-throughput digital backend. The hardware architecture is shown in Figure 2.7.

The core enabling technology is the LIROC chip, originally designed for LIDAR applications, which provides precise time stamping of single-cell avalanches with 50–100ps resolution. By reusing LIROC in the Random Power! context, the system achieves the same level of precision required for extracting randomness from SiPM dark counts, while offering the compactness and robustness of ASIC integration.

Each 64x board consists of an array of sensors, front-end biasing and control, and an FPGA responsible for parallelized TDC operation, symbol extraction, and quality monitoring. The design is conceived for scalability:

in standalone operation, the board delivers a secure bit-stream via Ethernet, while multiple boards can be interconnected via a custom motherboard to form a “randomness farm”, aggregating the output of several generators and delivering it to many users simultaneously over PCIe.

System integration uses a Xilinx KRIA K26 System-on-Module containing FPGA (programmed with the firmware responsible for TDC operation, bit extraction, and health monitoring) and an ARM processor running Ubuntu. This enables secure distribution protocols, compartmented architecture to reduce attack surfaces, and on-board cryptographic post-processing, including AES-256 and Root-of-Trust IP cores provided by NAGRA.

The 64x configuration provides bit rates of about 32 Mbps while maintaining raw entropy quality, validated through constant health test monitoring (MONOBIT, RUNS, Repetition Count Test, Adaptive Proportion Test). Bit rates can be boosted up to 1 Gbps when seeding the integrated NIST DRBG of the board with the generated raw bit-stream, acting as an entropy amplifier.

Software Architecture

The software architecture of the 64x generator board is built on a Ubuntu 22.04.5 LTS Linux environment, running on the quad-core ARM Cortex-A53 processor of the Xilinx Kria K26 SoM (System on Module).

When the board powers on, the Linux kernel initializes the hardware and passes control to `systemd`, which launches services in the proper order. Early in the sequence the **Kernel Module** is inserted, responsible for handling firmware interrupts and setting up the DMA memory area. Next, the **Startup Component** is launched and performs initial system setup; if it fails, `systemd` retries until it returns a successful exit code. Once the **Startup Component** is active, dependent services such as the **Scheduler**

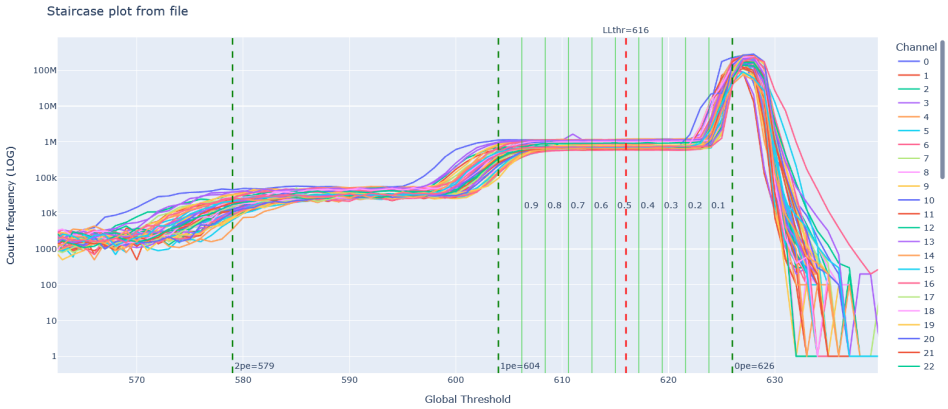


Figure 2.8: Staircase plot demonstrating LIROC channel alignment after equalization. The staircase is obtained by sweep the global threshold step by step and counting impulses within a fixed time window for each channel. Repeating this process across all 64 channels shows aligned transitions, confirming successful calibration. A common global threshold is then chosen in the 0pe-1pe region to ensure uniform operating point. Optimal values lie between 0.5pe-0.8pe.

and the **GUI Client** are released. The **Scheduler** then registers itself to the **Kernel Module** via an `ioctl` call, enabling it to receive interrupt-driven signals from the firmware. At this point, with all services running, Linux reaches its target state and the board is ready to process random bit requests.

The software architecture consists of the aforementioned components, each with its own specific function, described in the following:

- **Startup Component:** The startup component software is launched immediately after the boot-up of the board and loading of the operating system. It is responsible for initializing the system and performing calibration of the embedded TDC and LIROC front-end before normal operation begins. This initialization sequence follows a specific series of steps: first, the component loads the bitstream, which defines the digital connections between blocks inside the FPGA, then loads calibration data for both the temperature controller and analog frontend

equalization. Next, it sets up the external precision high-frequency oscillator used in the TDC, enables the thermoelectric module, and sets the temperature according to calibration data. After waiting for temperature stabilization, the component enables the high voltage bias Power Supply Unit (PSU) and sets the voltage according to calibration data. The equalization parameters are then applied to the analog frontend chip (LIROC), ensuring that every channel is aligned with each other (as seen in Figure 2.8). Subsequently, the TDC is calibrated to account for nonlinearities within the FPGA. Finally, the system performs self-tests, generates bits and timestamps, and conducts initial analysis to guarantee high-quality bits before the system enters normal operation.

- **GUI Client:** this Django-based [44] software provides an accessible web application that allows users to forward production requests to the board. These requests range from raw bit-streams directly from the generator, DRBG outputs, or the generation of keys. Users have the possibility to encrypt the result of their requests by providing a public key too. User data is stored directly in the Django database. The GUI interfaces with the back-end architecture of the board through UNIX sockets.

- **Scheduler:** this component is the back-end infrastructure that manages the board. On one side, it ensures communication with external users through the GUI, receiving their requests and managing their production, guaranteeing a fair access to the resources for each user, minimizing latency. In fact, requests are internally split in smaller sub-requests, and production is rotated in a round robin fashion. Requests are passed to the firmware via a Trusted Execution Environment (OP-TEE running on version 4.0 [45]), which acts as the secure gateway for all critical and low-level operations. The TEE is responsible for

configuring parameters, handling startup and calibration procedures, and enforcing the Root-of-Trust. It also guarantees the encryption of the bit stream and supports cryptographic key generation. Moreover, the scheduler is in a constant listening state for interrupts incoming from the firmware, which signal that requested bits have been produced and are ready to be read and forwarded to the appropriate user through a DMA operation. Finally, the scheduler is also responsible of gathering, always from the firmware, production statistics, also forwarded to the GUI in order to constantly monitor the state of the board.

- **Client CLI:** The Client Command Line Interface (CLI) is a command-line extension of the GUI client. It connects to the Django back-end by bypassing the GUI, forwards production requests, and downloads results directly from the terminal. Under the hood, it makes an HTTP GET request to the back-end. To perform user authentication, prior access to the GUI is required, as it is necessary to generate a unique token with specific privileges to attach with every request.

An additional element of the architecture is the **Kernel Module**, a loadable block of code that extends the kernel's functionality at runtime and provides the firmware with an interface to system software components (startup component and scheduler). It resides in the kernel and performs auxiliary functions, such as handling hardware interrupts and converting them into signals recognizable by the software.

2.3.3 ASIC

The RaP! ASIC integrates all essential functions for quantum entropy generation and cryptographic post-processing into a single mixed-signal device, in 180 nm CMOS technology node. The chip operates from a single-rail power supply and interfaces seamlessly with a wide range of platforms

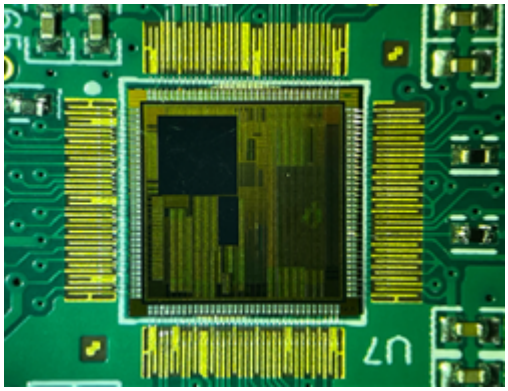


Figure 2.9: Top view of the RAP!'s ASIC.

(including SoCs, CPUs, FPGAs, and MCUs) through dual 24 MHz SPI channels. The first SPI is dedicated to configuration and parameter tuning and can be permanently disabled to prevent unauthorized access, while the second enables user control, recalibration, and entropy generation.

The ASIC implements the full NIST SP 800-90 A/B/C DRBG suite, achieving FIPS 140-3 compliance up to prediction resistance. Optimal operational parameters are stored in on-chip One-Time Programmable (OTP) memories, ensuring both stability and tamper resistance. In addition to raw entropy output, the chip supports AES-256 encryption, key signing, and an internal Key Derivation Function, that derives keys from an immutable silicon root combined with device and customer identifiers. This secure architecture prevents key extraction and enables on-chip generation of certified cryptographic material.

Owing to these embedded capabilities, the device extends beyond a traditional entropy source, serving as a fundamental building block for Root-of-Trust architectures, secure IoT nodes, Hardware Security Modules (HSMs), and Quantum Key Distribution (QKD) systems.

2.4 Final Remarks

An essential feature of the Random Power! platform is the integration of on-line health testing during bit generation. In the Single Generator Board, firmware implements sanity checks such as the MONOBIT and RUNS tests, while the 64x Generator Board extends this functionality with the Repetition Count Test (RCT) and Adaptive Proportion Test (APT). These tests serve both as safeguards against catastrophic failures of the entropy source and as continuous quality monitors when seeding the DRBG, ensuring robustness even in real-time streaming conditions.

Beyond real-time monitoring, post-production validation remains crucial. Generated sequences are subjected to standard statistical test suites, including NIST SP800-22, NIST IID validation, and the TestU01 battery. These complementary assessments provide additional assurance of entropy quality and are described in the next chapter. Measurements of Shannon entropy and min-entropy on the Single Generator Board confirm that the raw entropy is extremely close to the theoretical maximum, demonstrating the quality of the endogenous source. Together with the proof of IID compliance already achieved on SGB streams, these results establish the platform's suitability for certification.

The combination of real-time anomaly detection and embedded DRBG qualification (presented in Chapter 5), and extensive offline statistical testing establishes a robust framework for ensuring the reliability of the entropy source. This approach safeguards robustness against hardware malfunctions and environmental variations, while also advancing the certification road-map toward FIPS-140-3, supporting Root-of-Trust functionality, and ensuring readiness for deployment in secure infrastructures.

In the following chapter, a systematic overview of the statistical test suites employed for post-production entropy validation is presented.

CHAPTER 3

Statistical Test Suites and Entropy Validation

The key to achieving robust randomness does not lie only on generation processes, but also on the continuous monitoring of entropy quality. Standards such as NIST SP 800-90B [4] and FIPS 140-3 require entropy estimation, continuous health checks, and statistical validation. Test suites like NIST SP 800-22 [14] and TestU01 [46] define a series of tests aimed for the detection of biases and problems with random number generation. These tests are based on a test statistic Y, computed from a finite sample of the generator's output values (or bits, in the case of bit-level generators), and whose distribution under the null hypothesis H_0 is known or can be estimated. Passing a test, or a battery of tests, does not prove that the generator is universally reliable for all applications. However, poor generators will consistently fail even the simplest tests, whereas robust generators might fail only most sophisticated ones. Moreover, a certain degree of failure is due to be expected even by good generators due to statistical fluctuations in the production. For these reasons, a rigorous qualification of the bit-stream requires evaluating results from multiple test batteries. Since each battery targets different and complementary properties, their combined assessment provides a complete picture of the bit-stream's statistical quality.

In this chapter, we describe the test suites used for qualifying the bit-streams generated by our QRNGs, along with an analysis of NIST's IID

validation procedure and its results. We also present our qualification framework for classifying bit-stream quality based on test outcomes, and demonstrate its effectiveness using four types of datasets: a well-calibrated baseline QRNG, a mis-calibrated QRNG, a CSPRNG, and controlled manipulations designed to systematically reduce entropy.

3.1 TestU01

TestU01 is a comprehensive software library implemented in `ANSI C`, designed for the empirical statistical testing of uniform random number generators (RNGs) [46]. Developed by Pierre L'Ecuyer and Richard Simard at Université de Montréal, TestU01 represents one of the most extensive and flexible open source testing framework available for evaluating the statistical quality of RNGs.

The library offers three predefined test batteries of increasing complexity: **SmallCrush**, **Crush**, and **BigCrush**. SmallCrush applies 10 tests with 15 statistics, Crush applies 96 tests computing 144 statistics, while BigCrush applies 106 tests. For binary sequences, the batteries **Rabbit**, **Alphabit**, and **BlockAlphabit** provide specialized testing capabilities.

What makes TestU01 particularly useful is how flexible and extensible it is. Users can test around 200 generators already implemented from the literature, their own custom generators through a simple interface, random numbers stored in files (either binary or text), or output from physical random number devices.

Rather than using fixed significance levels, TestU01 reports p-values for all tests, just like the NIST Test Suite. A p-value is computed as $p = P[Y \geq y \mid H_0]$ where Y is the test statistic and y is the observed value. A p-value below a predetermined significance level suggests the sequence exhibits non-random behavior. This approach allows researchers to assess the degree of randomness and identify specific statistical weaknesses in the

generator. This makes more sense for testing random number generators because we can always increase the sample size if needed, and if we see something suspicious, we can investigate further by running additional tests on separate subsequences.

TestU01 has had significant impact on RNG development and evaluation. Generators based on simple linear congruential methods with small moduli consistently fail multiple tests, while well-designed combined generators with long periods and good theoretical properties generally pass all batteries.

3.1.1 Application to QRNG Evaluation

For our QRNG evaluation, we run the Crush and Alphabit test batteries on sets of 20 files (40 Gbit total). By default, TestU01 runs each test only once per dataset, which means that tests requiring fewer bits would not fully utilize our sample. To address this, we modified the library to repeat less demanding tests cyclically until the entire sample is consumed (a “callback" mode), effectively running them multiple times and providing a more thorough evaluation across the complete dataset without biasing other tests.

Additionally, we run the Rabbit battery on each 2 Gbit file individually, allowing us to detect potential anomalies that might appear in specific segments of the data. This layered view distinguishes single-sequence anomalies from systematic deviations repeated across files.

3.2 NIST Statistical Test Suite

The NIST Statistical Test Suite is a comprehensive collection of statistical tests specifically designed to evaluate the randomness of binary sequences produced by RNGs. Developed by the National Institute of Standards and Technology, the suite is documented in Special Publication 800-22 [14] and has become a widely recognized standard for assessing the quality of

random number generators used in cryptographic applications.

The suite comprises 15 distinct statistical tests, each targeting specific types of non-randomness that could compromise cryptographic security. These tests include the Frequency (Monobit) Test, Block Frequency Test, Runs Test, Longest Run of Ones Test, Binary Matrix Rank Test, Discrete Fourier Transform Test, Non-overlapping Template Matching Test, Overlapping Template Matching Test, Maurer's Universal Statistical Test, Linear Complexity Test, Serial Test, Approximate Entropy Test, Cumulative Sums Test, Random Excursions Test, and Random Excursions Variant Test. Each test examines different statistical properties of the binary sequence, from basic bit distribution to more sophisticated measures like spectral characteristics and linear complexity.

With the NIST test suite, users can test binary sequences from various sources, including TRNGs, PRNGs, or data stored in binary files. The suite processes sequences of varying lengths, though each test has specific minimum length requirements to ensure statistical validity.

Just like TestU01, the NIST suite reports p-values for each test rather than simple pass/fail outcomes. This library has become an essential benchmark in cryptographic RNG evaluation, with many standards and certification processes requiring generators to pass these tests.

3.2.1 Application to QRNG Evaluation

For our QRNG evaluation, we run the complete NIST test suite on each 2 Gbit file individually. Each file is partitioned into 2000 sequences of 10^6 bits, meeting the requirements of all tests including Maurer's Universal Statistical Test. For each file we record the per-sequence pass/fail outcomes at the stated significance level, check whether the total number of passing sequences matches what we would expect from a binomial model under H_0 (a fair coin), and verify that the distribution of p-values across sequences

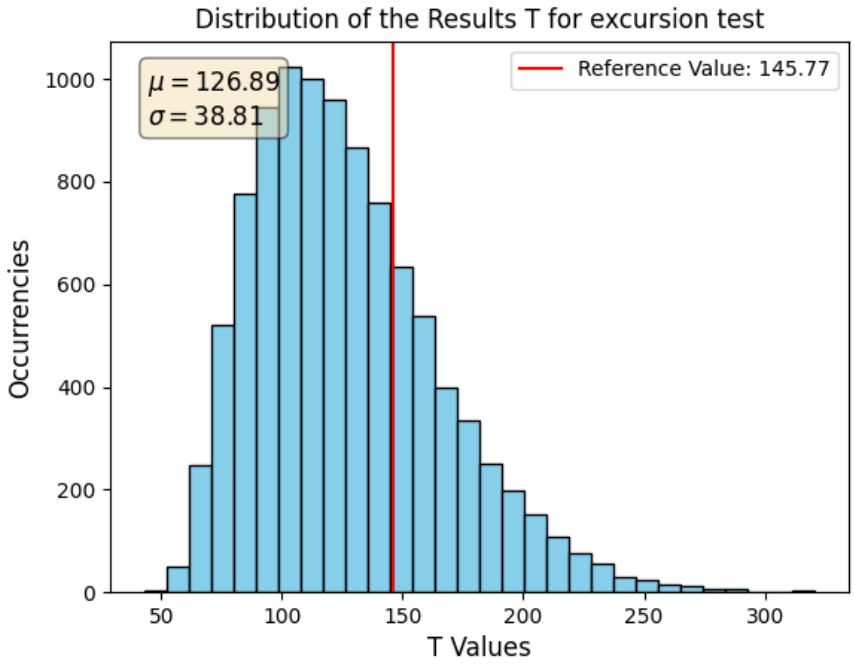


Figure 3.1: Histogram of T_i values for the excursion test with 10^5 sequences of 10^3 symbols. The red vertical line marks the reference T_x of the input sequence; the population mean and standard deviation are also reported.

is statistically compatible with $\mathrm{Unif}[0, 1]$ through the p-value uniformity check reported by the suite. It is important to note that some failures across the 2000 sequences are expected due to statistical fluctuations, and given a 3σ confidence level, we expect about 36 failures purely by chance. Having no failures at all can actually be as suspicious as having too many, since it may indicate that the test is not operating properly or that the data has been artificially filtered.

3.3 IID Validation Procedure

The following results are presented in [3]. The standard procedure for estimating the minimum entropy of a binary source is described in [4]. It relies on the assumption that the symbols of the bit-stream are independently and identically distributed (IID). Chapter 5 of the NIST documentation details the evaluation of this IID hypothesis through a bootstrap methodology applied to eleven statistical tests. These tests fall into six categories: excursion statistics, run-based statistics, collision tests, periodicity tests, covariance tests, and compression tests. Representative pseudocode for each category is provided in Algorithms 2–6 in Appendix A.

A sequence of length n_{symbols} is used as input to the tests. The resulting statistics (denoted T_x) are compared against those obtained from shuffled versions of the same sequence, generated via the Fisher–Yates algorithm. For each shuffled sequence, if the statistic T_i exceeds T_x, a counter C_0 is incremented; if $T_i = T_x$, a second counter C_1 is incremented. After evaluating $n_{\text{sequences}}$ shuffled samples, the IID assumption is rejected if, for any test,

$$C_0 + C_1 \leq 0.0005 \times n_{\text{sequences}} \tag{3.1}$$

or

$$C_0 \geq 0.9995 \times n_{\text{sequences}}. \tag{3.2}$$

This ensures that T_x is not in the most extreme 0.05% of the distribution of T_i values. In the NIST reference case ($n_{\text{symbols}} = 10^6$, $n_{\text{sequences}} = 10^4$), these conditions correspond to $C_0 + C_1 \leq 5$ or $C_0 \geq 9995$. The procedure is summarized in Algorithm 7 in Appendix A.

Figure 3.1 shows an example distribution of T_i values. A qualitative check confirms that T_x is not extreme enough to reject IID. However, its location within the distribution strongly influences the probability of incrementing C_0, motivating a deeper statistical analysis beyond the NIST prescription.

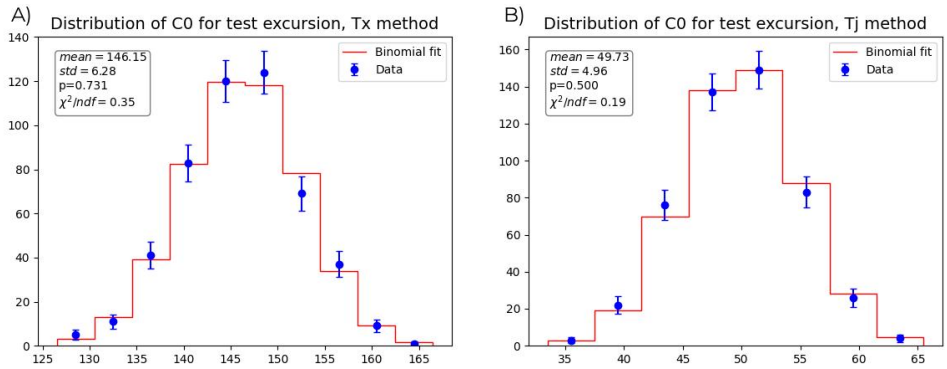


Figure 3.2: A) Distribution of 500 counters C_0 for the excursion test, each over 200 sequences of 10^3 symbols. Blue dots show observed values with binomial error bars; the red curve shows the expected binomial distribution.
B) Same setup but using the T_j^{Norm} pairwise normalization method, which enforces $p = 0.5$. Insets report mean, standard deviation, and reduced χ^2.

Our approach examines the distribution of C_0 across multiple runs rather than a single run. For each run, parameters are set to $n_{\mathrm{symbols}} = 10^3$ and $n_{\mathrm{sequences}} = 200$, repeated $n_{\mathrm{iterations}} = 500$ times. This ensures full use of a 250 MB binary file generated with *Random Power!*'s QRNG while leaving a buffer for additional tests. The expected distribution of C_0 is binomial with success probability p, determined by the percentile rank of T_x among the T_i values. Figure 3.2A compares observed results (blue points) with the theoretical distribution (red line), reporting mean, standard deviation, and χ^2 goodness-of-fit.

Because the reference T_x is effectively random, it may introduce bias. To remove this privileged role, we propose a normalization approach: sequences are compared pairwise, incrementing C_0 whenever the j-th result exceeds the $(j+1)$-th, and discarding ties. This guarantees $p = 0.5$ by construction, producing the centered distribution of Figure3.2B.

Once a source is validated as IID, its minimum entropy is calculated as

$$H_{\min} = -\log_2(p_{\max}),$$

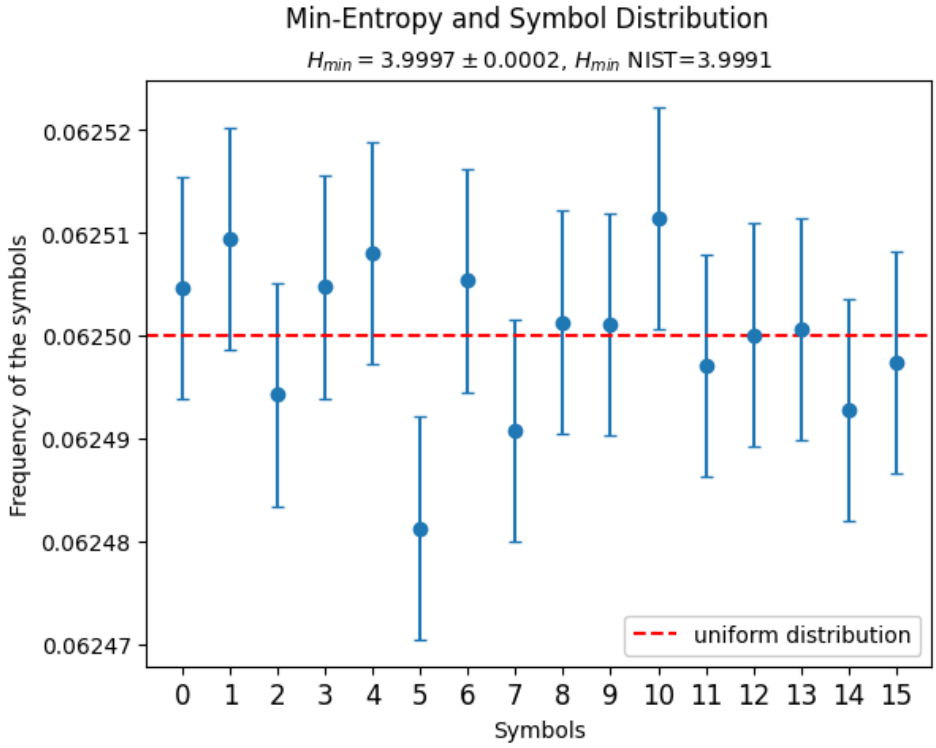


Figure 3.3: Frequencies of 4-bit symbols in a 250MB binary file generated with Random Power!'s QRNG with binomial error bars. The red dashed line indicates the uniform distribution. The measured minimum entropy $H_{\min}$ and the NIST-compliant bound $H_{\min}^{\text{NIST}}$ are reported.

where $p_{\max}$ is the frequency of the most probable symbol. NIST recommends a conservative lower bound by using an upper confidence limit on $p_{\max}$ (99.5%, corresponding to 2.576σ). Figure 3.3 shows 4-bit symbol frequencies from a 250 MB QRNG file, along with their binomial error bars and expected uniform distribution. The measured minimum entropy is 3.9997 ± 0.0002, with the NIST lower bound of 3.9991 also indicated.

The full Python implementation for IID testing, statistical analysis, and

entropy computation is publicly available[1].

3.4 Qualification of the Test Suites and IID Procedure

As each test suite presented in this chapter targets different properties of the bit-stream, a meaningful qualification requires understanding their scope, assumptions, and intended interpretation, then reading their outcomes jointly. By *qualification* we mean recognizing what each tool actually measures. NIST SP 800-22 and TestU01 examine the temporal structure for the bit-streams, looking for global bias, short-range dependence, non-stationarity, template or linear artifacts, and anomalous random walks. In contrast, NIST SP 800-90B's IID decision determines whether an IID model is adequate when we group the bits into symbols of a specific size (e.g., 4-bit nibbles) for the purpose of entropy estimation, while the min-entropy estimators bound how predictable individual symbols are, rather than whether dependencies exist across time.

3.4.1 Data Structure and Sample Sizing

Our testing protocol processes binary files organized to meet the requirements of the most demanding tests (in terms of sample size) while enabling meaningful statistical analysis. Each raw file contains approximately 2 Gbit directly produced from the generator and stored unmodified. Files are grouped into sets of 20, yielding a total sample size of 40 Gbit per set. This sizing ensures that demanding tests of our test suites operate in their nominal regimes.

For NIST SP 800-22, each 2 Gbit file is partitioned into $N_{\text{seq}} = 2000$ sequences of length $n = 10^6$ bits, meeting the sample requirements for all tests of the suite. Endianness and decoding are fixed and verified so that the logical bit order presented to the test batteries matches the generator order.

[1] https://github.com/RandomPower/IID_validation

Unless otherwise stated, all results in this chapter refer to these 40 Gbit sets and this per-file partitioning.

3.4.2 Classification and Acceptance Rules

To combine the results from individual tests into an overall verdict for each file and dataset, we implement a tiered classification system. This system recognizes that even a perfect random number generator will occasionally fail a test simply due to statistical fluctuations. After all, at a fixed significance level α, a small number of failures is statistically expected even for ideal generators.

Binary file classification (NIST SP 800-22 + Rabbit): We assign each file one of four grades based on the combined results from both test batteries. Figure 3.4 shows an example of a complete NIST SP 800-22 report for a binary file, illustrating the detailed output that informs this classification. The grading system is defined as follows:

- GOLD: No failures in SP 800-22 and Rabbit.
- SILVER: At most one marginal failure in either SP 800-22 (pass count near the binomial lower/upper bound) or Rabbit.
- BRONZE: Two marginal failures or one in each battery.
- DISCARDED: Otherwise.

Set classification (20 files, Crush + Alphabit): For the battery-level tests operating on 40 Gbit aggregates (sets of 20 files), we evaluate both Crush and Alphabit batteries. Figure 3.5 shows an example Alphabit report, with the summary clearly stating "All tests were passed" after evaluating 17 distinct statistical tests. For acceptance, we typically use a two-sided 0.2% tail (99.8% confidence level) per statistic. Under H_0 we expect about 2.5

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is </home/ccesare/bins/bins_numpy/bins/32_20250523_111000.bin>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
203 205 218 163 204 210 207 173 221 196  0.080765   1987/2000    Frequency
202 216 190 200 210 191 191 198 186 216  0.799073   1981/2000    BlockFrequency
196 207 195 212 188 189 198 197 193 225  0.746903   1983/2000    CumulativeSums
211 199 193 202 188 206 212 186 207 196  0.924076   1990/2000    CumulativeSums
182 209 203 218 204 208 208 201 172 195  0.488534   1976/2000    Runs
190 202 190 208 201 199 209 201 232 168  0.213309   1979/2000    LongestRun
209 193 211 190 210 183 188 196 201 219  0.708536   1980/2000    Rank
213 207 178 224 204 201 203 188 196 186  0.504219   1972/2000    FFT
204 183 177 200 205 199 194 216 222 200  0.516113   1983/2000    NonOverlappingTemplate
186 197 211 185 196 213 214 201 191 206  0.811993   1990/2000    NonOverlappingTemplate
182 209 215 218 206 201 202 198 183 186  0.593478   1977/2000    NonOverlappingTemplate
203 210 214 194 223 180 189 196 221 170  0.131879   1983/2000    OverlappingTemplate
220 190 178 226 186 178 210 215 191 206  0.125563   1981/2000    Universal
204 223 185 199 214 187 203 201 189 195  0.693142   1980/2000    ApproximateEntropy
133 118 132 139 142 124 120 114 103 128  0.327058   1236/1253    RandomExcursions
129 121 140 141 117 126 140 106 132 101  0.111652   1239/1253    RandomExcursions
118 130 124 130 130 135 120 126 125 115  0.971423   1242/1253    RandomExcursions
117 131 120 122 122 136 119 133 122 131  0.952528   1242/1253    RandomExcursions
117 161 117 104 136 125 124 109 143 117  0.012439   1241/1253    RandomExcursions
120 113 124 121 133 128 119 133 125 137  0.909826   1238/1253    RandomExcursions
123 133 121 132 123 108 141 133 125 114  0.643783   1242/1253    RandomExcursions
130 109 128 112 135 138 118 131 117 135  0.552790   1240/1253    RandomExcursions
130 131 131 127 129 119 105 126 111 144  0.447676   1236/1253    RandomExcursionsVariant
143 124 119 135 135 106 113 119 124 135  0.381773   1238/1253    RandomExcursionsVariant
135 135 125 145 113 107 133 101 125 134  0.111652   1240/1253    RandomExcursionsVariant
142 135 132 122 130 109 121 109 131 122  0.498313   1243/1253    RandomExcursionsVariant
127 157 133 137  95 114 134 101 128 127  0.005327   1243/1253    RandomExcursionsVariant
122 159 137 120 116 122 114 110 120 133  0.094512   1240/1253    RandomExcursionsVariant
123 145 136 135 109  96 119 126 122 142  0.057948   1240/1253    RandomExcursionsVariant
127 115 139 144 117 116 128 135 114 118  0.467322   1240/1253    RandomExcursionsVariant
196 173 206 190 193 200 197 213 206 226  0.428095   1978/2000    Serial
203 205 186 180 198 202 206 208 207 205  0.907419   1979/2000    Serial
201 203 186 196 204 203 198 183 235 191  0.416320   1973/2000    LinearComplexity


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 1966 for a
sample size = 2000 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 1229 for a sample size = 1253 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
```

Figure 3.4: Example NIST SP 800-22 Statistical Test Suite report for a single 2 Gbit file, showing results for all 15 tests across 2000 sequences of 10^6 bits each. All tests passed: p-values are well-distributed (none near 0 or 1), the proportion of passing sequences falls within expected binomial bounds for each test, and no test failures are marked (failures would be indicated with an asterisk). The summary notes confirm that minimum pass rates meet requirements for both standard tests and random excursion variants.

failures across approximately 1236 statistics from the Crush test, with up to roughly 6 tolerated to keep the global rejection near 99% confidence level for well-behaved streams. Each test in these batteries reports multiple statistics with their associated p-values, as illustrated in the figure where the MultinomialBitsOver test shows Kolmogorov-Smirnov and Anderson-Darling statistics along with their p-values. We classify sets as:

- GOLD: Neither Crush nor Alphabit fails, as demonstrated in Figure 3.5.
- SILVER: Crush has ≤ 6 failures and Alphabit has none.
- BRONZE: Crush has ≤ 6 and Alphabit has exactly one.
- DISCARDED: Otherwise.

3.4.3 Empirical Validation with Controlled Entropy Manipulation

To validate this qualification framework and demonstrate how different test suites respond to distinct types of degradation, we analyzed seven datasets:

- **REF**: Well-calibrated QRNG (reference baseline). The naming refers to the fact that this serves as our reference standard with no artificial entropy manipulation of the bit-stream.
- **CSPRNG**: Data generated with a cryptographically secure pseudo-random number generator as a reference control [17]. Since CSPRNGs are designed to pass statistical tests, this dataset should behave similarly to the baseline (REF), confirming that our qualification framework correctly identifies high-quality randomness.
- **MIS**: Mis-calibrated QRNG, where the miscalibration resulted from misaligned local channel threshold offsets, resulting in the production of biased bit-streams.

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
          Starting Alphabit:   nb = 41940480000
          Version: TestU01 1.2.3
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
************************************************************
HOST = rapsrv, Linux

ufile_CreateReadBin:   set_1/alphabit.bin
smultin_MultinomialBitsOver test:
-----------------------------------------------
   N = 42,  n = 998582848,  r =  0,   s = 32,   L =    2,
       Sparse =  FALSE

       Number of bits = n = 998582848
       Number of cells = 2^L =                  4
       Expected number per cell =  2.4964571e+08
       Hashing =  FALSE

   For Delta > -1, we use the ChiSquare approximation
   Correction factor of the ChiSquare:
       Delta =     1,    Mu =  5.0070958e-10,    Sigma = 1

-----------------------------------------------
Test Results for Delta =   1.0000
Kolmogorov-Smirnov+ statistic = D+    :   0.015
p-value of test                       :    0.97

Kolmogorov-Smirnov- statistic = D-    :    0.18
p-value of test                       :    0.06

Anderson-Darling statistic = A2       :    1.84
p-value of test                       :    0.11

For the sum of the N observations, we use
      the Normal approximation:
Standardized empirical mean           :    1.91
p-value of test                       :    0.03

Standardized empirical correlation    :   -0.43
p-value of test                       :    0.67
-----------------------------------------------
CPU time used                    :  00:01:24.62

Generator state:
 41940479616  bits have been read.
========= Summary results of Alphabit =========

 Version:          TestU01 1.2.3
 File:             set_1/alphabit.bin
 Number of bits:   41940480000
 Number of statistics:  17
 Total CPU time:   00:15:48.09

 All tests were passed
====================Test End====================
```

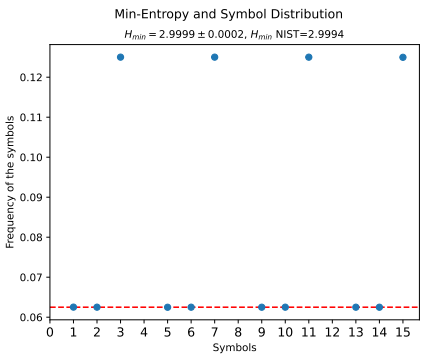
Figure 3.5: Example output from TestU01's Alphabit battery for a 40 Gbit dataset (set of 20 files). The report shows one representative test (MultinomialBitsOver) with its statistical results including Kolmogorov-Smirnov and Anderson-Darling statistics. All p-values are well within acceptable ranges (no values near 0 or 1), and the summary confirms that all 17 tests in the battery passed. The complete battery evaluates multiple statistical properties of the bit-stream, with each test reporting several statistics and their associated p-values.

- **Injected biases** (**M4**, **M8**, **M16**, **M32**, **M64**). We deliberately degrade the entropy of the bit-stream by partitioning it into consecutive *symbols* of size $b \in \{4, 8, 16, 32, 64\}$ bits and applying exactly one bit operation per symbol according to a deterministic rule. For a given symbol with integer value x (and, when needed, the previous symbol value x_{prev}), we compute a target bit index

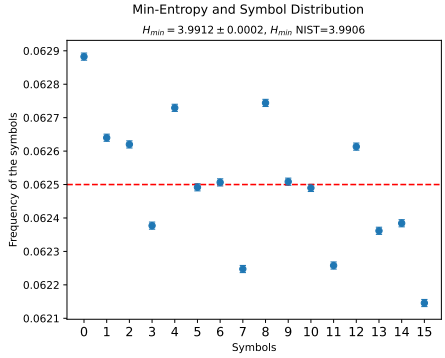
$$p = \begin{cases} x \bmod b, & \texttt{CLEAR}, \texttt{SET}, \texttt{FLIP}, \texttt{ALTERNATE} \\ x_{\text{prev}} \bmod b, & \texttt{CORRELATION} \end{cases}$$

  and then act on bit p of the current symbol: (i) `CLEAR`: clear it; (ii) `SET`: set it; (iii) `FLIP`: flip it; (iv) `ALTERNATE`: alternate clear/set on successive symbols while keeping p from the current symbol; (v) `CORRELATION`: symbol-wise manipulation of the stream, where we copy into bit p the bit value of the *previous* symbol at position p (i.e., enforce $x[p] \leftarrow x_{\text{prev}}[p]$). In the **M4** setting, the stream is treated as a sequence of nibbles (high then low), so x_{prev} refers to the preceding nibble. This procedure yields a controlled, reproducible entropy reduction: manipulations occur most frequently in **M4** (every nibble) and most sparsely in **M64** (every 8 bytes), enabling a graded study of how systematic bit-level biases impact qualification tests.

The `CLEAR` and `SET` manipulations have a severe effect: by deterministically clearing or setting one bit in each symbol, they drastically skew the symbol distribution. Some symbol values become "drugged" (artificially over-represented in the output), while others are suppressed or, in the case of 4-bit nibbles, they disappear entirely. This can be clearly seen in Figure 3.6a. These manipulations proved too catastrophic for meaningful analysis, as test batteries consistently failed even at the lowest manipulation frequency. For this reason, we focus our analysis in this chapter on the more subtle manipulation types. Nonetheless, summary tables for these



(a) Min-Entropy estimation and symbol distribution of the `FLIP-M4`.



(b) Min-entropy estimation and symbol distribution of `CORRELATION-M64`.

Figure 3.6: Exemplary representation of symbol distribution and min-entropy estimation of the IID procedure on two bit-streams manipulated with different entropy degradation procedures, namely `FLIP-M4` (Fig. 3.6a) and `CORRELATION-M64` (Fig. 3.6b). It is noticeable on Fig. 3.6a how some symbols are clearly missing (`0, 4, 8, 12`), while the occurrences of symbols `3, 7, 11, 15` are drugged.

manipulations can be found in Appendix B. The `FLIP` and `ALTERNATE` manipulations create moderate per-symbol distortions that still allow us to observe how the test batteries respond to varying levels of systematic bias, while `CORRELATION` introduces dependencies between consecutive symbols, adding a temporal dimension to the bias that should be particularly challenging for tests to detect.

These three manipulation types introduce a range of bias mechanisms: `FLIP` and `ALTERNATE` distort symbol frequencies, while `CORRELATION` introduces temporal dependencies. This allows us to study how different test batteries respond to distinct forms of non-randomness.

NIST STS and TestU01 (Rabbit) results: Table 3.1 summarizes the per-file average failure counts across 20 files for each dataset. The well-calibrated baseline (REF) and CSPRNG both perform essentially as expected, with virtually no catastrophic ("bad") failures and only minimal statistical failures (0.35 and 0.30 respectively) and Rabbit failures (0.25 and 0.10 respectively).

The mis-calibrated source (MIS) shows a different pattern: approximately 3 bad failures per file and a notable 11.40 Rabbit failures on average, indicating consistent but moderate issues with the bit-stream quality. This intermediate behavior clearly distinguishes it from both the healthy baselines and the deliberately manipulated datasets.

The injected-bias series demonstrates the dramatic impact of systematic manipulations across all three manipulation types. For `ALTERNATE`, the densest case (M4) produces 154 bad failures and 34.40 Rabbit failures per file, gradually decreasing to 12.15 bad failures and 23.65 Rabbit failures at M64. The `FLIP` manipulation shows a similar pattern, ranging from 146.45 bad failures at M4 down to 39.80 at M64. The `CORRELATION` manipulation, despite introducing only temporal dependencies, still produces substantial failures: 160 bad failures at M4 decreasing to 4.30 at M64. This clear monotonic decrease across all manipulation types directly reflects how diluting the systematic bias makes it progressively harder for the tests to detect non-random behavior. Interestingly, `CORRELATION` shows the steepest decline in bad failures from M4 to M64, suggesting that temporal correlations become particularly difficult to detect when sparsely distributed.

TestU01 (Crush and Alphabit) set totals: Table 3.1 reports the aggregate failure counts and resulting classification for each 40 Gbit dataset. Both reference datasets (REF and CSPRNG) earn Silver classifications with little failures: REF shows only 1 Crush failure and no Alphabit failures, while CSPRNG shows 3 Crush failures and no Alphabit failures. These results confirm that both well-calibrated QRNGs and CSPRNGs behave identically

under our qualification framework, validating the framework's ability to recognize high-quality randomness regardless of source.

The mis-calibrated source (MIS) shows 257 Crush failures and 12 Alphabit failures, receiving a Discarded classification and clearly indicating systematic issues that distinguish it from healthy generators.

The injected-bias series demonstrates clear monotonic behaviour across all manipulation types. For `ALTERNATE`, Crush failures decrease from 1149 at M4 down to 350 at M64. `FLIP` shows failures ranging from 1043 at M4 to 293 at M64, while `CORRELATION` ranges from 1057 at M4 to 441 at M64. All manipulated datasets receive Discarded classifications, and notably, all show consistent Alphabit failures (17 for most cases, with slight decreases at M64 for some manipulation types). The gradient directly reflects how the frequency of systematic bias affects the TESTU01's ability to detect non-random behavior, with `FLIP` generally showing slightly fewer failures than `ALTERNATE` or `CORRELATION` at sparser manipulation frequencies.

NIST 800-90B IID validation and min-entropy: Table 3.2 shows the IID validation rate and average min-entropy estimate per 4-bit symbol. The reference datasets (REF and CSPRNG) both achieve perfect scores: 20/20 files pass IID validation with essentially perfect min-entropy of ≈ 3.9995 bits per nibble. The mis-calibrated source (MIS) also passes IID validation on all files with very high min-entropy (although slightly lower than the reference's, ≈ 3.9954 bits per nibble), despite its numerous sequence test failures. This highlights the fundamental difference between entropy computation and temporal structure of the bit-stream.

The injected-bias series displays an interesting non-linear behavior that varies significantly by manipulation type. For FLIP, we observe the expected pattern: M4 passes IID on all files (20/20) but shows severely reduced entropy (≈ 2.9998 bits per nibble), mid-range manipulations (M8 through M32) fail IID entirely (0/20 validated), and the sparsest manipulation (M64)

Table 3.1: Statistical test results across datasets and manipulation types. Top: per-file average failures for NIST SP 800-22 and Rabbit. Bottom: aggregate failures for Crush and Alphabit batteries with set verdicts. Reference datasets (REF, CSPRNG) show minimal failures consistent with statistical fluctuations. MIS shows moderate consistent failures. Manipulated datasets display monotonically decreasing failures as manipulation frequency decreases, demonstrating how systematic bias becomes progressively harder to detect when diluted.

Dataset	**STS Bad Fails (avg/file)**			**STS Stat. Fails (avg/file)**			**Rabbit Fails (avg/file)**		
REF	0.00			0.35			0.25		
CSPRNG	0.20			0.30			0.10		
MIS	3.05			0.35			11.40		
	ALT.	**FLIP**	**CORR.**	**ALT.**	**FLIP**	**CORR.**	**ALT.**	**FLIP**	**CORR.**
M4	154.00	146.45	160.00	0.00	0.00	0.00	34.40	30.60	33.50
M8	144.15	160.00	150.00	2.55	0.00	2.15	33.15	30.65	29.60
M16	116.95	144.20	100.00	11.40	1.55	26.25	30.70	31.00	29.00
M32	57.30	100.80	34.75	14.95	8.90	17.50	27.60	30.00	24.45
M64	12.15	39.80	4.30	8.00	13.10	0.90	23.65	21.35	18.70

Dataset	**Crush Total (set)**			**Alphabit Total (set)**			**Set Verdict**		
REF	1			0			Silver		
CSPRNG	3			0			Silver		
MIS	257			12			Discarded		
	ALT.	**FLIP**	**CORR.**	**ALT.**	**FLIP**	**CORR.**	**ALT.**	**FLIP**	**CORR.**
M4	1149	1043	1057	17	17	17	Discarded	Discarded	Discarded
M8	1000	855	760	17	17	17	Discarded	Discarded	Discarded
M16	935	798	537	17	17	17	Discarded	Discarded	Discarded
M32	600	494	705	17	17	17	Discarded	Discarded	Discarded
M64	350	293	441	17	16	14	Discarded	Discarded	Discarded

Dataset	**IID Validated (files)**			**Min-entropy (mean)**		
REF	20/20			3.9995		
CSPRNG	20/20			3.9995		
MIS	20/20			3.9954		
	ALT.	**FLIP**	**CORR.**	**ALT.**	**FLIP**	**CORR.**
M4	0/20	20/20	0/20	3.4148	2.9998	3.1926
M8	0/20	0/20	0/20	3.6779	3.4149	3.7518
M16	0/20	0/20	0/20	3.8301	3.6780	3.9124
M32	4/20	0/20	0/20	3.9503	3.9021	3.9719
M64	19/20	13/20	0/20	3.9861	3.9722	3.9913

Table 3.2: IID validation rates and min-entropy estimates when grouping bits into 4-bit symbols. The baseline (REF) achieves near-perfect results on both metrics. The mis-calibrated source (MIS) interestingly passes IID and maintains high entropy at this granularity, even though it fails many temporal sequence tests. The injected-bias series shows entropy gradually recovering as manipulations become sparser, while IID validation behaves differently for each dataset.

recovers partial IID validation (13/20 files) with nearly full entropy (≈ 3.9722 bits per nibble).

The reason why M4-FLIP passes IID validation despite catastrophic sequence test failures lies in how the NIST SP 800-90B procedure operates. As explained in Section 3.3, the IID validation algorithm takes as input a string of N symbols from the test sample and generates shuffled permutations for comparison. The algorithm's universe of samples is therefore limited to what appears in the input and it is completely blind to the fact that some symbols might be missing from the theoretical alphabet. The `FLIP` manipulation at M4 frequency (on nibbles) entirely removes several symbol values from the 4-bit alphabet, creating a reduced but uniform distribution over the remaining symbols. When the algorithm shuffles only the symbols that

are present, it sees a consistent distribution and validates the sequence as IID.

In contrast, the min-entropy computation requires normalization over the alphabet size, which is therefore passed as input parameter, making the algorithm aware that symbols are missing. This is why M4-`FLIP` shows severely reduced min-entropy (≈ 2.9998 bits per nibble) despite passing IID validation: the min-entropy calculation correctly accounts for the reduced alphabet, while the IID test only evaluates independence among symbols that actually appear.

`ALTERNATE` and `CORRELATION` show different patterns. `ALTERNATE` fails IID completely from M4 through M16, begins recovering at M32 (4/20 files validated), and nearly fully recovers at M64 (19/20 files). `CORRELATION`, in contrast, fails IID validation entirely across all manipulation frequencies (0/20 for all cases), even at M64 where entropy has recovered to ≈ 3.991 bits per nibble (as it can be seen in Figure 3.6b). This persistent IID failure despite high marginal entropy demonstrates that the temporal correlations introduced by `CORRELATION` create patterns that remain detectable at the 4-bit symbolization level even when diluted.

3.5 Discussion

These results are fully consistent once we understand what each tool actually measures. At first glance, some outcomes might seem contradictory. For instance, how can MIS pass IID validation with nearly perfect min-entropy while simultaneously failing many sequence tests? The answer lies in recognizing the different aspects of randomness each tool analyses.

NIST SP 800-22 and TestU01 examine temporal structure: they look for correlations between successive bits, periodic patterns, and subtle drifts in statistical properties over time. In contrast, NIST SP 800-90B's IID test and min-entropy estimation focus on the symbol distribution at a specific gran-

ularity (in our case, 4-bit nibbles). The min-entropy calculation considers how predictable the next symbol is, given the observed symbol frequencies. It does not check correlations, but only the distribution of the test sample itself.

This explains the MIS results perfectly. With min-entropy of approximately 3.995 bits per nibble, the individual symbols appear nearly random when examined independently. The symbol frequencies are almost uniform, which is what the min-entropy calculation measures. However, the bit-level process that generates these symbols contains subtle patterns (potentially slight timing correlations) that sequence tests can detect, identifying the mis-calibration of the device.

The manipulated datasets reveal even more about how these tools operate. As we dilute the systematic bias from M4 to M64, we see clear improvements: Crush failures decrease from around 1000 to 300-400, and min-entropy recovers from approximately 3.0-3.4 bits per nibble up to nearly 4.0 bits per nibble. It is obvious how sparse biases are harder to detect across all batteries.

However, IID validation behaves unexpectedly, and differently for each manipulation type. `FLIP` passes IID at M4 (due to the algorithm's blindness to missing symbols, as explained earlier), fails completely at intermediate frequencies, then partially recovers at M64. `ALTERNATE` shows a different pattern: complete IID failure from M4 through M16, then gradual recovery at M32 and M64. `CORRELATION` fails IID at every manipulation frequency, even at M64 where it achieves 3.99 bits per nibble. The temporal dependencies that `CORRELATION` introduces create patterns that remain visible at the 4-bit granularity even at a rate of 1 every 64 bits on average, while `ALTERNATE`'s manipulations become less visible as they spread out.

These controlled experiments lead to several operational guidelines. First, complete qualification requires multiple test batteries. NIST SP 800-22, TestU01, and IID validation each evaluate different properties of the bit-

stream, and combining their results provides a comprehensive assessment of randomness quality. Second, when sequence tests fail but entropy looks acceptable, the issue is almost always in hardware calibration (threshold settings, timing alignment, afterpulsing and dead-times, or thermal stability), and these problems need to be diagnosed and fixed at the hardware level. Finally, understand what each tool actually measures: `FLIP` at M4 passed IID validation despite catastrophic sequence test failures because the IID algorithm only shuffles symbols actually present in the input, while `CORRELATION` failed IID at every manipulation frequency because the dependencies it introduces remain detectable at the 4-bit granularity no matter how sparse they become. Results need to be interpreted carefully, without relying on a single outcome.

3.5.1 Final Remarks

The statistical test suites presented in this chapter serve complementary roles in assessing QRNG quality. NIST SP 800-22 and TestU01 look at the temporal structure, searching for patterns, correlations, and dependencies that spread across the bit-stream. NIST SP 800-90B's IID validation and min-entropy estimation quantify the distribution of symbols and whether an independence assumption is justified at a chosen granularity.

Our empirical validation with controlled manipulations demonstrates that these tools measure fundamentally different, but complementary, properties. The well-calibrated baseline (REF) and CSPRNG both pass all tests, confirming the framework recognizes high-quality randomness regardless of source. The mis-calibrated source (MIS) shows the importance of not relying on single metrics. The manipulated datasets (M4 through M64) show how different forms of bias intteract with test batteries in distinct ways, with varying detectability as manipulation frequency decreases.

The downside of these test batteries is that they are computationally ex-

pensive, requiring up to 40 Gbit of data and hours of processing. They serve as essential diagnostic and certification tools during QRNG calibration and validation, but cannot run continuously during normal operation. This limitation motivated the development of online statistical tests (Chapters 4 and 5), designed to operate in near real-time on small data windows, enabling continuous health monitoring and early detection of drifts or anomalies. This framework applies well-known complementary statistical tests with a rigorous probabilistic analysis, designed to rapidly detect catastrophic failures in the generation process that would introduce systematic biases in the bit-stream.

Together, the offline validation methods presented here and the online monitoring framework provide a comprehensive strategy: offline tests certify that a calibrated QRNG produces high-quality randomness across both marginal and temporal dimensions, while online tests maintain that quality throughout the device's operational lifetime by catching degradation as it occurs.

CHAPTER 4

Standard Statistical Tests

Proving the randomness of a source presents significant challenges in both selecting appropriate statistical diagnostic tests and implementing them effectively. While the statistical test suites introduced in Chapter 3 provide comprehensive entropy assessment, they require large data volumes and substantial computational resources. These requirements make them unsuitable for real-time quality evaluation of bit-streams during generation. To address this limitation, NIST has established guidelines for implementing continuous health tests [4, 47].

This chapter introduces the statistical tests underlying our online framework, which will be presented in Chapter 5. NIST specifies the Repetition Count and Adaptive Proportion Tests as lightweight monitors designed to detect catastrophic entropy source failures, where each failure triggers rejection of the tested sequence. Building upon these recommendations, we developed an anomaly detection procedure that extends these two tests, along with the Monobit and RUNS statistics, into a unified statistical framework for analyzing sequences of test outcomes. Rather than discarding each failing sequence individually, our method accumulates and evaluates the distribution of results, distinguishing between sporadic fluctuations, statistically expected in random processes, and persistent deviations indicative of systematic bias.

This transition from single-sequence rejection to distribution-based monitoring enables on-line detection of long-term drifts in source quality, while simultaneously providing an estimate of the minimum entropy of the bit-stream. The procedure requires only a limited number of samples compared to the full test suites, thus offering a practical and efficient approach for continuous quality assessment of random bit generators. In this chapter, we describe the four tests forming the core of the method (namely, the Monobit and RUNS tests, together with NIST's Repetition Count and Adaptive Proportion Tests). The qualification of the procedure and the results obtained will be presented in the following one.

The four tests can be grouped into two main categories: *symmetry tests* and *runs tests*. The symmetry tests include the Monobit Test, which evaluates the balance between the number of ones and zeros in the bit-stream, and the Adaptive Proportion Test, which generalizes the Monobit Test to sources with an extended symbol alphabet. The runs tests, on the other hand, comprise the RUNS Test and the Repetition Count Test. The RUNS Test compares the observed number of bit flips within a sequence to the statistically expected value, while the Repetition Count Test verifies that the longest sub-sequence of identical symbols does not exceed a predefined threshold.

4.1 Symmetry Tests

4.1.1 Monobit

The Monobit test [14] evaluates the balance between zeros and ones within a binary sequence, thereby assessing potential asymmetry in the bit-stream. Given a sequence of n bits, the test statistic is defined as

$$S_n = \sum_{i=1}^{n} x_i = 2n_1 - n, \tag{4.1}$$

where $x_i = 2\epsilon_i - 1$, ϵ_i denotes the i-th bit, and n_1 represents the number of ones in the sequence. Assuming that the bit values are independent and identically distributed (i.i.d.), the number of bits set to one follows a binomial probability distribution:

$$B(n_1; n, p) = \frac{n!}{n_1!(n - n_1)!} p^{n_1} (1 - p)^{n - n_1}, \tag{4.2}$$

where p is the probability of generating a one. For an ideal random source, both symbols occur with equal probability, $p = 0.5$, yielding an expected value $\overline{S_n} = 0$ and a standard deviation $\sigma_{S_n} = \sqrt{n}$. The Monobit test fails whenever the observed statistic S_n exceeds a predefined alarm threshold $k\sqrt{n}$, where k is a parameter determined by the desired sensitivity and false alarm rate.

4.1.2 Adaptive Proportion Test

The Adaptive Proportion Test (APT) extends the binary analysis performed by the Monobit test to sources with an alphabet of m symbols. As specified in the NIST documentation [4], the bit-stream is partitioned into consecutive sequences of length n, and the frequency of occurrence of the first symbol is evaluated against the hypothesis of a binomial distribution with success probability $p = 1/m$.

The choice of the sequence length n depends on the size of the alphabet and is defined by NIST specifications. For binary sources, a sequence length of $n = 1024$ is recommended, whereas for non-binary sources, $n = 512$ is suggested. Within each sequence, the number of occurrences of the first symbol is counted and compared to a cut-off threshold derived from the binomial cumulative distribution function, under the assumption that each of the m symbols is equally probable. A test failure is registered whenever the observed frequency exceeds this threshold, indicating a deviation from the expected uniform distribution.

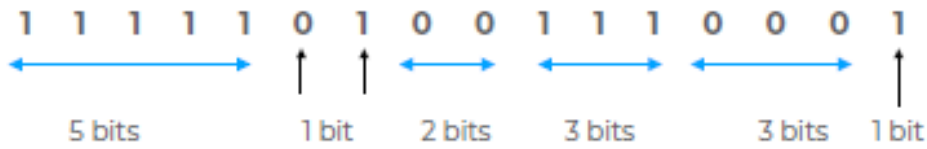


Figure 4.1: The RUNS test counts the number of sequences of consecutive identical bits in a bit-stream. In this figure, a sequence of $n = 16$ bits containing a total of 7 runs is shown.

4.2 Runs-based Tests

4.2.1 RUNS Test

The RUNS test counts the number of series of consecutive identical bits within a sequence of specified length, as illustrated in Fig. 4.1. This quantity is equivalent to the number of bit transitions plus one.

The test complements the Monobit test: for a given number of ones n_1, it evaluates whether the number of observed bit transitions is consistent with the hypothesis of a "fair coin". The underlying distribution describes the probability of observing runs of a given length, conditioned on the total number of ones in the sequence, with a success probability $p = 1/2$. Defining

$$\pi = \frac{n_1}{n}$$

as the fraction of bits set to one, the expected number of runs and the corresponding variance are given by [48]:

$$\begin{aligned} \overline{R} &= 2n\pi(1-\pi) + 1, \\ \sigma_R^2 &= \frac{n}{n-1}\left[2\pi(1-\pi)\big(2n\pi(1-\pi) - 1\big)\right], \end{aligned} \tag{4.3}$$

which represent the first and second moments of the associated probability distribution function. The purpose of the RUNS test is to verify whether

the alternation between subsequences of identical bits occurs at a rate consistent with statistical expectations. Sequences that exhibit transitions that are excessively frequent or infrequent relative to the expected value are indicative of bias or dependence in the source [49]. As in the Monobit test, a sequence is considered to fail whenever the number of runs R lies outside the acceptance interval $\overline{R} \pm k\sigma_R$, where k defines the desired confidence level.

4.2.2 Repetition Count Test

The Repetition Count Test (RCT) is conceptually related to the RUNS test but generalizes it to sequences of arbitrary symbols rather than binary bits. Instead of counting the number of symbol transitions, the RCT focuses on the length of consecutive identical symbols within the sequence.

The probability α of observing at least C consecutive occurrences of the same symbol can be expressed as

$$\alpha = P(k \geq C) = \sum_{i=1}^{m} (1 - p_i)\, p_i^C, \tag{4.4}$$

where k denotes the run length, p_i is the probability associated with the i-th symbol of the alphabet, and $(1 - p_i)$ represents the probability that the preceding symbol differs, thereby resetting the run counter. Assuming an ordering $p_1 \geq p_2 \geq \cdots \geq p_m$, the probability in (4.4) can be bounded above by

$$\alpha = P(k \geq C) \leq (m - 1)\, p_1^C = (m - 1)\, (2^{-H})^C, \tag{4.5}$$

where H is the min-entropy, defined as $H = -\log_2(\max\{p_i\})$ [4]. Given a desired false-alarm probability α, the corresponding cut-off threshold C can be computed as

$$C = \left\lceil \frac{1}{H} \Big[\log_2(m - 1) - \log_2(\alpha) \Big] \right\rceil. \tag{4.6}$$

It should be noted that (4.6) slightly differs from the original NIST prescription due to the sequential implementation adopted in this work, which assumes that symbols are analyzed in real time. This formulation reflects the viewpoint of an "observer symbol," which must differ from the preceding one and match the subsequent $C-1$ symbols to constitute a valid run.

NIST recommends selecting the parameter α within the range 2^{-20} to 2^{-40}, corresponding approximately to 5σ and 7σ confidence levels, respectively, under a Gaussian assumption for the test statistics. In this thesis, a value of $\alpha = 2^{-20}$ is adopted.

CHAPTER 5

On-line Statistical Testing Procedure

As anticipated in Chapter 4, NIST defines the Repetition Count and Adaptive Proportion Tests as lightweight mechanisms aimed at detecting catastrophic failures of the entropy source, with each failure leading to the rejection of the sequence under analysis. These tests operate on individual sequences independently, providing binary pass/fail outcomes but limited insight into long-term source behavior.

In this chapter, we present a novel extension of these statistical tests, together with the Monobit and RUNS statistics, into a unified anomaly detection framework. Rather than evaluating each sequence as stand-alone, our approach analyzes the distribution of test outcomes over time, allowing one to distinguish random fluctuations from persistent deviations that may indicate a systematic bias or drift in the generator.

We introduce the Inter-failure Sequence Number ($\overline{ISN}$) as a new qualification metric, representing the average number of sequences observed between two failures. This shift from single-sequence rejection to distribution-based monitoring enables on-line tracking of source quality and, critically, enables estimation of the minimum entropy during continuous operation, a property that is not provided by standard single-sequence testing.

The methodology was validated on 100 Gb of data from quantum random number generators and implemented in FPGA hardware with minimal

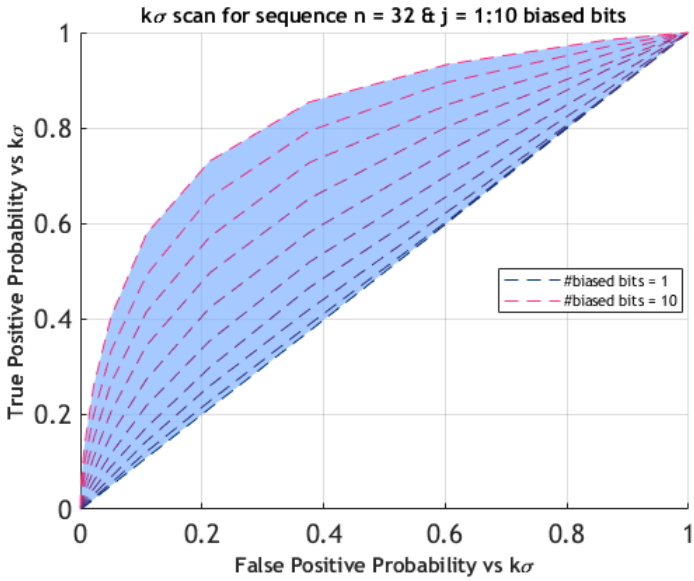


Figure 5.1: True Positive vs. False Positive probability for the Monobit test at varying confidence levels ($k\sigma$) for sequences of length $n = 32$ bits and a number of biased bits j ranging from 1 to 10. Each curve corresponds to a fixed j, with k scanned from 0 to 4. The parameter k determines the acceptance bounds for the test: sequences are flagged as anomalous if their count of 1-bits falls outside the interval $[n/2 - k\sigma, n/2 + k\sigma]$, where $\sigma = \sqrt{n}$. As k decreases from 4 to 0, the acceptance window tightens, increasing both the True Positive rate and the False Positive rate. This trade-off is traced out along each curve from bottom-left $k = 4$ to top-right $k = 0$. The shaded region illustrates the increasing sensitivity as the bias fraction grows, showing that for single-sequence detection a significant number of biased bits is required.

resource overhead. In the following sections, the qualification of the procedure and present the results obtained are discussed, as they were published in [50] and [3].

5.1 Monobit

The diagnostic power of the Monobit test (described in Section 4.1.1) on individual bit-strings can be expressed as its ability to detect anomalies in the number of ones, n_1, within a sequence of n random bits. Anomalies arise when the absolute value of S_n exceeds $k\sigma_{S_n}$, where k defines the confidence level. This condition can be written as:

$$\begin{aligned} S_n \geq k\sigma &\implies (2n_1 - n) \geq k\sigma \\ &\implies n_1 \geq \tfrac{1}{2}(k\sqrt{n} + n). \end{aligned} \tag{5.1}$$

To assess the diagnostic power of the Monobit test under controlled conditions, we introduce artificial biases by forcing a specified number of bits to one. If j bits are forced to one, the alarm triggers when the number of ones in the remaining $n - j$ unbiased portion of the string, n_1^{n-j}, satisfies

$$n_1^{n-j} \geq \tfrac{1}{2}(k\sqrt{n} + n) - j. \tag{5.2}$$

The tail probability of this distribution represents the True Positive Probability (TPP) for Monobit failures, while the False Positive Probability (FPP) corresponds to the same tail probability for an unbiased sequence. Their comparison for $n = 32$ bits at confidence levels ranging from 0 to 4σ is shown in Figure 5.1. Sensitivity remains close to 50% unless the number of biased bits becomes a significant fraction of n. However, when a series of S_n values is analyzed instead of single outcomes, the TPP can be enhanced, enabling anomaly detection with single-bit precision.

This reasoning extends naturally to all the tests discussed in the previous section, shifting the focus from assessing individual sequences to the statistical analysis of a series of N sequences of n bits. The objective is to distinguish systematic failures, which might indicate a persistent bias in the source, from occasional ones due to statistical fluctuations.

The key observable is $\overline{S_n}$, the average of S_n across N sequences, which (by the Central Limit Theorem) is expected to follow a Gaussian distribution.

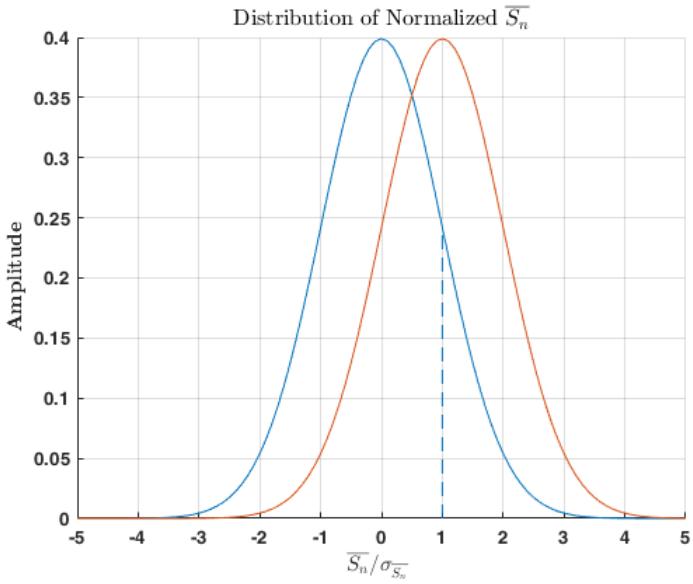


Figure 5.2: Normalized $\overline{S_n}$ distributions for unbiased (blue) and biased (red) sequences. The bias shifts the mean and alters the tail probability above a fixed threshold (dashed line), providing two indicators of systematic deviation.

Introducing a bias by forcing j bits to one, two potential indicators emerge (see Figure 5.2): (i) a shift in the mean $\overline{S_n}$ and (ii) a change in the fraction of samples in the distribution tails. Assuming all sequences are biased, the expected mean shift depends linearly on j:

$$S_n = 2n_1 - n = 2(n_1^{n-j} + j) - (n^{n-j} + j) = (2n_1^{n-j} - n^{n-j}) + j, \qquad (5.3)$$

hence $E[\overline{S_n}] = j$, scaled by the fraction of biased sequences. Conversely, the fraction of tail events depends non-linearly on j, being given by the integral of the normalized $\overline{S_n}$ distribution above threshold k.

Figure 5.3 shows the sensitivity of both estimators for $n = 32$ as j varies. The biasing frequency spans from one biased sequence per set to one in a thousand across $N = 2^5$ samples. Results demonstrate that the $\overline{S_n}$ mean-shift estimator outperforms tail counting. At $k = 3$ (99.9% CL), the presence

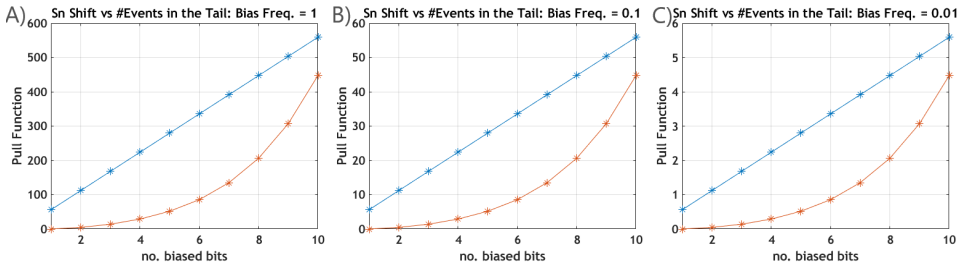


Figure 5.3: Comparison of bias sensitivity estimators for the Monobit test. The mean shift of $\overline{S_n}$ (blue) and the variation in tail-event counts above the 3σ threshold (orange) are shown as functions of the number of biased bits. Panels (A)–(C) correspond to biasing frequencies of 1, 0.1, and 0.01, respectively. The $\overline{S_n}$ shift consistently exhibits higher sensitivity across all conditions.

of a single biased bit is detectable if at least one in ten sequences is affected; when the bias frequency drops to one in a hundred, about five biased bits are required for detection. Sensitivity degrades at lower frequencies unless a larger N is used.

The $\overline{S_n}$ distribution forms the basis of the on-line quality assessment of the bitstream. A warning is issued when $\overline{S_n}$ exceeds the 3σ threshold, chosen as a compromise between sensitivity and false alarm rate. A lower threshold increases false positives, while a higher one risks missing genuine malfunctions. With $k = 3$, the false alarm probability is about 10^{-3} per series; two consecutive uncorrelated warnings have a combined probability of 10^{-6}, and three in a row 10^{-9}, thus effectively negligible in unbiased conditions.

A 1 Gb dataset was divded into series of $N = 2^{17}$ sequences of $n = 32$ bits. As shown in Figure 5.4, the $\overline{S_n}$ trace reveals no catastrophic failures, apart from a single warning attributable to statistical fluctuation. This demonstrates the method's ability to detect systematic degradations in entropy quality during on-line operation and promptly alert the user.

Finally, a retrospective analysis was conducted on the distribution of the

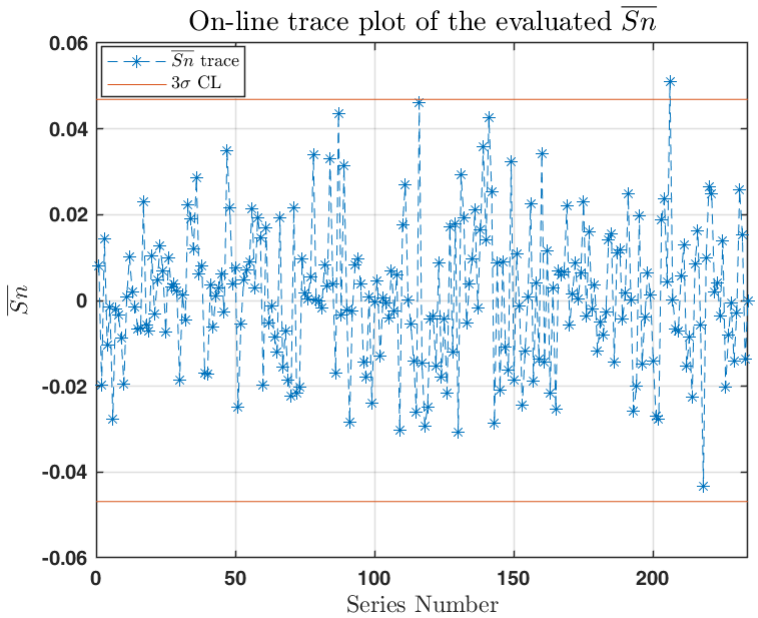


Figure 5.4: On-line monitoring of the Monobit test mean $\overline{S_n}$ over successive series of $N = 2^{17}$ sequences of $n = 32$ bits, for a total of 1 Gb of data. The orange lines mark the $\pm 3\sigma$ thresholds: values of $\overline{S_n}$ beyond these limits trigger warnings, while three consecutive warnings indicate a potential systematic failure of the entropy source.

Inter-failure Sequence Number (ISN), defined as the number of individual sequences between two consecutive test failures. Unlike the online monitoring approach described above, the ISN analysis evaluates each sequence independently against the $k\sigma$ threshold.

For each individual sequence, the test statistic S_n is computed and compared to the acceptance bounds $[\pm k\sigma]$. A failure occurs when $|S_n| > k\sigma$. The ISN is then the count of consecutive passing sequences between two such failures. This analysis was performed on the same 1 Gb dataset used for Figure 5.4, but with $k = 1$ rather than $k = 3$ to ensure sufficient failure events for statistical characterization of the distribution. The observed

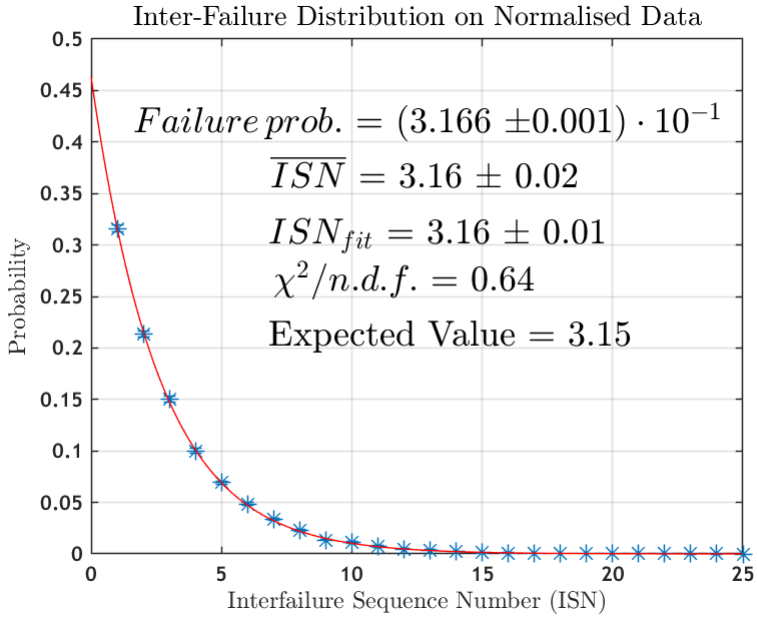


Figure 5.5: Inter-failure sequence number (ISN) distribution for the Monobit test on normalized data. The measured $\overline{ISN}$ and fitted ISN_{fit} values are both consistent with the theoretical expectation for an unbiased source ($\overline{ISN}_{\text{exp}} = 3.15$). The data closely follow the geometric model $P(X = x) = p(1-p)^{x-1}$, with a fitted failure probability $p = (3.166 \pm 0.001) \times 10^{-1}$ and a reduced chi-squared of 0.64, confirming statistical consistency with random behavior.

distribution, shown in Figure 5.5, follows

$$P(X = x) = p(1-p)^{x-1}, \tag{5.4}$$

which describes the probability of the next failure occurring after x sequences, given failure probability p. The results are statistically consistent with an unbiased source, as the total number of failures in 1 Gb of data (74.6 ± 2.6) matches the expected value of 77 within uncertainty.

k	***2-tailed failure probability***	$\overline{ISN}$
1	0.317	3.15
3	2.7×10^{-3}	370.4
5	5.7×10^{-7}	1.7×10^{6}
7	2.56×10^{-12}	3.9×10^{11}

Table 5.1: Expected $\overline{ISN}$ as a function of $k\sigma$ confidence level for a Normal distribution.

Sequence length n	***Avg.*** σ	$\sigma/\sqrt{N}$	ΔR_{limit} : $5\sigma/\sqrt{N}$	$(\Delta R/\sigma)_{limit}$: $5/\sqrt{N}$
32	2.695	0.009	0.043	0.016
64	3.906	0.012	0.062	0.016
128	5.590	0.018	0.088	0.016
256	7.953	0.025	0.126	0.016

Table 5.2: Sensitivity of the RUNS test at 5σ confidence level for ΔR variations in the number of runs. Results were obtained by analyzing 10^5 unbiased sequences and computing the average σ for each sequence length.

5.2 RUNS

As with the Monobit test, a collection of RUNS statistics can be used to diagnose biases and systematic failures by analyzing samples of N sequences of n bits. It is reasonable to assume that the same considerations outlined for the Monobit apply here; hence, the analysis focuses on the shift of the average measured number of runs from its expected mean.

Since the expected number of runs depends on the number of ones n_1 in each sequence, a normalized quantity is introduced. For each sequence, the measured number of runs R_m is converted to a z-score:

$$z = \frac{R_m - \overline{R}}{\sigma}, \tag{5.5}$$

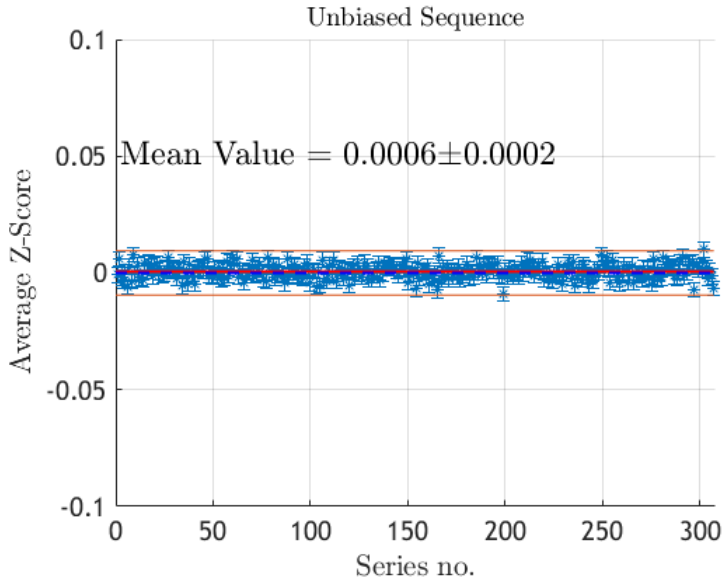


Figure 5.6: Trace plot of the average z-score for the RUNS test computed on-line during production over 307 series of $N = 2^{17}$ sequences of $n = 32$ unbiased bits. The red dashed line marks the expected mean value for a random bitstream, while the orange lines indicate the $\pm 3\sigma$ thresholds. With this configuration, approximately 0.3% of the series are expected to raise warnings; the single observed event among 307 series is consistent with this expectation, confirming the unbiased behavior of the source.

which is expected to follow a Gaussian distribution by virtue of the Central Limit Theorem. For unbiased series of N sequences, $\overline{z}$ is therefore centered around zero.

Following the same approach used for the Monobit test, the sensitivity of the RUNS test is investigated by introducing controlled biases in the

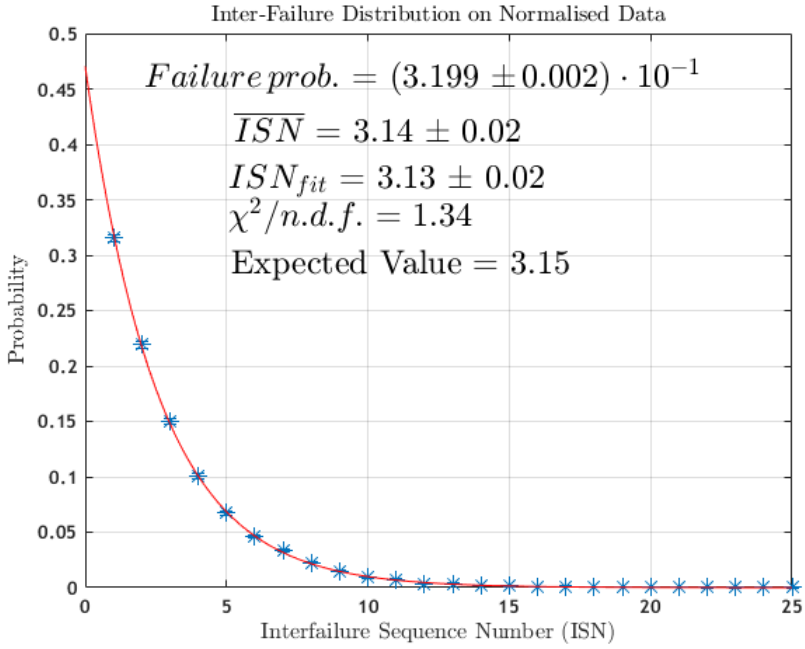


Figure 5.7: Inter-failure sequence number (ISN) distribution for the RUNS test on normalized data. The measured $\overline{ISN}$ and fitted ISN_{fit} values agree with the theoretical expectation for an unbiased source ($\overline{ISN}_{\text{exp}} = 3.15$). The distribution follows the geometric model $P(X = x) = p(1-p)^{x-1}$ with a fitted failure probability $p = (3.199 \pm 0.002) \times 10^{-1}$ and a reduced chi-squared of 1.34, confirming statistical consistency with random behavior.

number of runs. Defining

$$\begin{aligned} \overline{z} &= \frac{1}{N}\sum_{i=1}^{N} z_i, \\ \sigma_{\overline{z}} &= \frac{1}{\sqrt{N}}, \end{aligned} \tag{5.6}$$

a systematic change of ΔR in the number of runs induces a variation in the mean z-score of

$$\Delta\overline{z} = \frac{\Delta R}{\sigma_{\overline{z}}}, \tag{5.7}$$

which becomes statistically significant when $|\Delta\overline{z}| \geq k \cdot \sigma_{\overline{z}}$, where k defines the desired confidence level. Consequently, the detection threshold for the shift in the number of runs is

$$|\Delta R| \geq k \cdot \frac{1}{\sqrt{N}}. \tag{5.8}$$

A shift in the mean z-score occurring with a bias frequency of one in ten sequences can be detected with single-bit precision at a confidence level up to 5σ. As shown in Table 5.2, this sensitivity remains consistent for sequences up to 128 bits, while for longer sequences (or equivalently, for lower bias frequencies) the required deviation ΔR must be larger to achieve the same detection confidence.

Experimental results from the on-line evaluation of the z-score are shown in the trace plot of Figure 5.6. In the absence of catastrophic failures, all computed values remain within the expected confidence bounds, apart from normal statistical fluctuations.

A further analysis was conducted by measuring the Inter-failure Sequence Number (ISN) using the same dataset and parameters as for the Monobit test. The resulting ISN distribution, shown in Figure 5.7, is statistically consistent with the hypothesis of an unbiased source.

Overall, these results confirm that, like the Monobit test, the RUNS test is a robust tool for the on-line detection of systematic failures during bitstream production. Deviations from the expected number of runs manifest as measurable shifts in the z-score distribution and can be detected reliably even with moderate sample sizes.

H	α	C
2	2^{-20}	12
2	2^{-40}	22
4	2^{-20}	6
4	2^{-40}	11

Table 5.3: Cut-off thresholds C for the Repetition Count Test as a function of the entropy H and target failure probability α, computed for $m = 16$. These thresholds define the maximum number of consecutive identical symbols allowed before triggering a failure condition.

5.3 Repetition Count Test

The Repetition Count Test (RCT) was analyzed within a statistical framework by examining the properties of the Inter-failure Sequence Number (ISN) distribution, with the goal of assessing the bitstream quality in terms of the entropy H. Due to the large dataset required, this analysis was performed retrospectively rather than on-line; however, it incurs no additional computational cost, as it relies on data already collected during the NIST-mandated DRBG validation procedures.

Following NIST recommendations, the failure probability α is set to 2^{-20}. Given that the architecture under analysis produces 4-bit symbols, this leads to a threshold of $C = 6$ for the maximum number of consecutive identical symbols, as shown in Table 5.3. Within a defined observation window, failure occurrences are expected to follow a Poisson process, resulting in exponentially distributed ISN values.

A dataset of 100 Gb of random bits generated by four silicon-based QRNG boards was analyzed, with data from each board divided into three independent sets. To avoid floating-point precision issues related to very small exponential values, all measures were normalized with respect to the expected $ISN = 1/\alpha = 2^{6\cdot 4 - \log_2(15)}$, obtained from equation (4.6) using

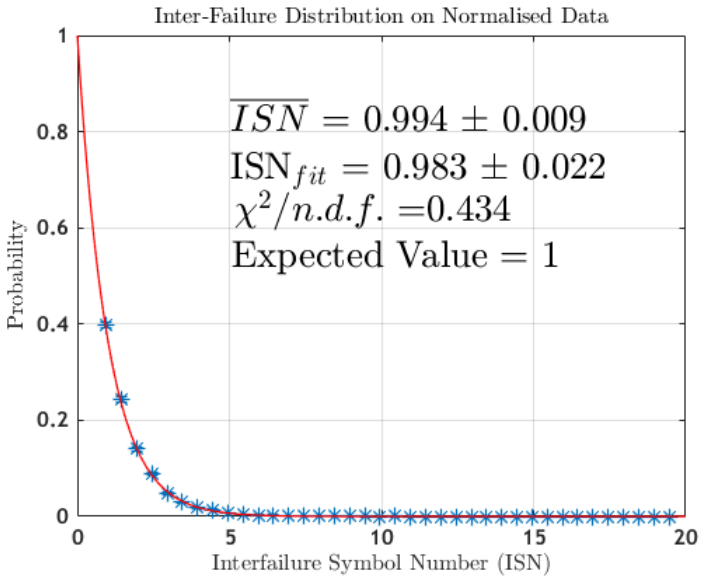


Figure 5.8: Normalized Inter-failure Sequence Number (ISN) distribution for the Repetition Count Test (RCT) computed on a 100 Gb dataset from one QRNG board. The data follow the expected exponential model, with the fitted parameter ISN_{fit} and the measured mean $\overline{ISN}$ in close agreement with the theoretical expectation ($\overline{ISN}_{\text{exp}} = 1$). The reduced χ^2 value confirms statistical consistency with the hypothesis of an unbiased entropy source.

$C = 6$ and $H = 4$. The observed ISN distributions, normalized over the total number of failures, follow the exponential model closely, as confirmed by the fits shown in Figure 5.8. The estimated parameters of the exponential distribution consistently fall within the 99.7% confidence interval of the theoretical expectation for an unbiased source.

The measured entropy H and its lower bound, the min-entropy, were derived from the average $\overline{ISN}$. Since C is fixed, the measured entropy is obtained by substituting the observed average failure rate $\overline{\alpha} = 1/\overline{ISN}$ into (4.6), and the uncertainty is propagated as

$$\sigma_H = \frac{1}{C \ln 2} \frac{1}{\sigma_\alpha}. \tag{5.9}$$

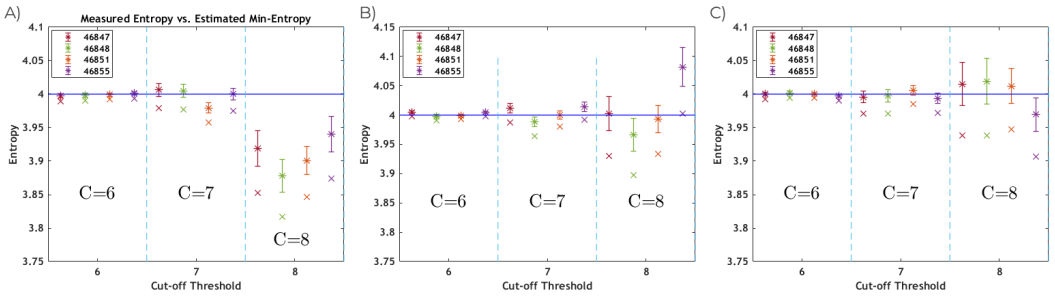


Figure 5.9: Measured entropy ($*$) and corresponding min-entropy estimates ($\times$) obtained from the Repetition Count Test (RCT) on 100 Gb of data generated by four SGBs: #46847 (red), #46848 (green), #46851 (orange), and #46855 (purple). Each board's dataset is divided into three subsets (panels A–C), and results are shown for three cut-off thresholds C. The blue horizontal line indicates the theoretical limit $H = 4$ for a perfectly 4-bit entropy symbol source. For $C = 6$, the estimated min-entropy consistently lies between 3.989 and 3.998, while higher thresholds yield larger uncertainties due to the reduced number of observed failures.

Given the large number of observed failures, $\overline{ISN}$ is assumed to follow a Gaussian distribution with standard deviation $\sigma_{\overline{ISN}} = \sqrt{n_{\text{fails}}}$, where n_{fails} denotes the total number of failure events in the sample. To obtain a conservative lower bound on the min-entropy, a $3\sigma_{\overline{ISN}}$ shift to the right of the mean is applied, yielding the maximum plausible true value of α (denoted $\tilde{\alpha}$) consistent with the observed data at 99.7% confidence:

$$\frac{1}{\tilde{\alpha}} = \widetilde{\overline{ISN}} = \overline{ISN} - 3\sigma_{ISN}. \tag{5.10}$$

Substituting this adjusted value into (4.6) provides a lower limit on the entropy.

The results, shown in Figure 5.9, consider several cut-off thresholds ($C = 6, 7, 8$). As expected, the measured entropy exhibits fluctuations that occasionally exceed physical constraints, reflecting the statistical variability in the number of observed failures (as shown in Table 5.4). The derived min-entropy represents the minimum entropy consistent with measurements,

obtained by applying the 3σ correction. For $C = 6$, this lower bound consistently falls between 3.989 and 3.998 across all boards and subsets analyzed. As C increases, failure events become rarer, leading to larger uncertainties in the estimated entropy.

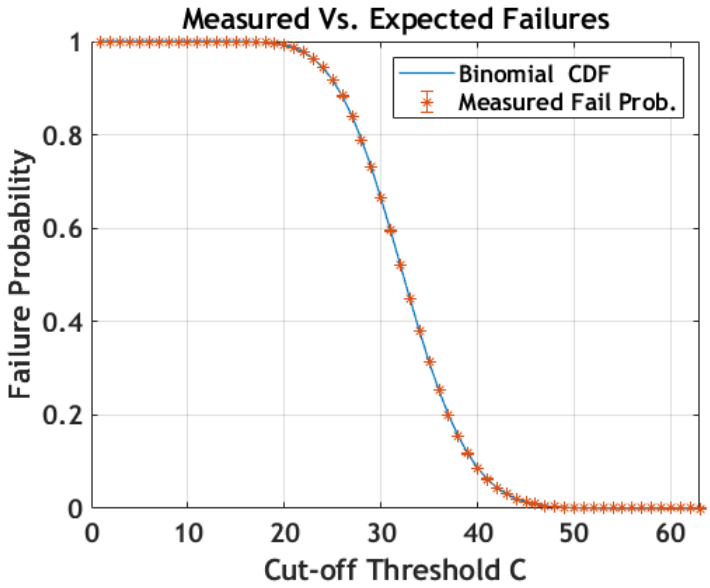


Figure 5.10: Comparison between measured and expected failure probabilities for the Adaptive Proportion Test (APT) as a function of the cut-off threshold C. The measured values (orange markers) lie within 3σ of the theoretical binomial cumulative distribution (blue line), in agreement with the expected behavior for an unbiased bit-stream.

5.4 Adaptive Proportion Test

The analysis of the Adaptive Proportion Test (APT) follows the same statistical framework adopted for the Repetition Count Test (RCT) to estimate the min-entropy. According to NIST guidelines, the data are partitioned into sequences of $n = 512$ symbols. For each sequence, the number of occurrences of the first symbol is counted, and the test fails if this count exceeds a defined threshold C. The corresponding failure probability follows a binomial cumulative distribution function (CDF):

$$P(k \geq C) = B_{\text{cdf}}\big(C - 1, N - 1, p\big), \tag{5.11}$$

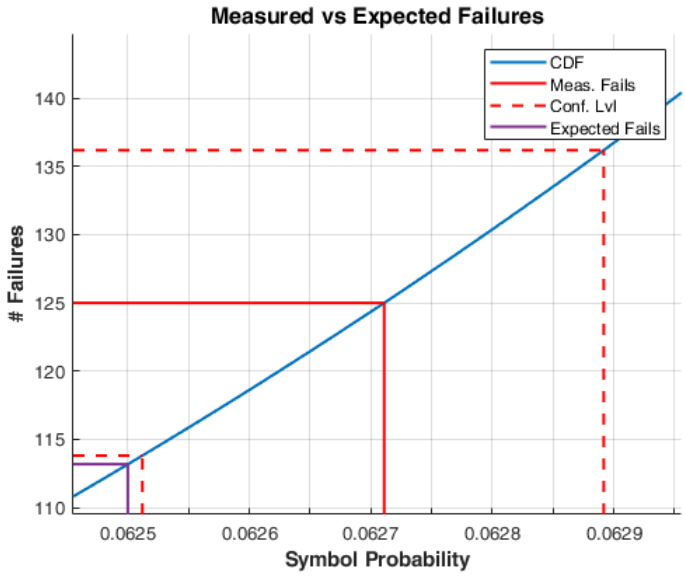


Figure 5.11: Comparison between measured and expected failure probabilities for the Adaptive Proportion Test (APT) as a function of the cut-off threshold C. The measured values (orange markers) lie within 3σ of the theoretical binomial cumulative distribution (blue line), in agreement with the expected behavior for an unbiased bit-stream.

where p denotes the probability of occurrence of the selected symbol, and the -1 terms account for the first symbol, which is fixed and already counted. For an unbiased bit-stream, $p = \frac{1}{16}$. Cases where the number of occurrences falls below a lower threshold are also considered; in such instances, the corresponding cumulative binomial probability is added to equation (5.11) to obtain the overall failure probability.

Given the failure probability $\alpha = 2^{-20}$ and $m = 16$, the test is deemed unsuccessful if the number of occurrences of the first symbol exceeds $C \geq 62$ or falls below $C \leq 8$. The observed distributions, measured for varying cut-off thresholds, are shown in Figure 5.10 and compared with the theoretical binomial CDF, exhibiting excellent agreement.

The failure probability α is directly related to the probability of occurrence p of a symbol, which determines the entropy of the bit-stream as $H = -\log_2(p)$. From the analysis, the estimated value $p = 0.0627 \pm 0.0002$ is obtained from the observed failures and their binomial uncertainties, as shown in Figure 5.11. The corresponding measured entropy values and uncertainties are summarized in Table 5.4. By computing the weighted average for each data set and board, the lower bounds on the entropy at 99.7% confidence are: $H_{\#46847} = 3.9917$, $H_{\#46848} = 3.9724$, $H_{\#46851} = 3.9842$, and $H_{\#46855} = 3.9824$.

The scope of the APT analysis can be further extended by evaluating the occurrence rate of every symbol in the alphabet. This generalization is supported by the observation that the minimum entropy is determined by the probability of the most frequent symbol. In this case, the failure probability for each symbol can still be modeled as binomial; however, potential correlations among symbols may alter the effective number of failures observed.

Sets\Boards	#46847		#46848		#46851		#46855	
	RCT	**APT**	**RCT**	**APT**	**RCT**	**APT**	**RCT**	**APT**
Set 1	3.997 ± 0.003	$4.007^{+0.014}_{-0.010}$	3.999 ± 0.003	$3.990^{+0.009}_{-0.011}$	3.999 ± 0.002	$4.004^{+0.010}_{-0.008}$	4.001 ± 0.003	$3.955^{+0.011}_{-0.009}$
Set 2	4.005 ± 0.003	$4.024^{+0.016}_{-0.011}$	3.998 ± 0.003	$3.990^{+0.008}_{-0.009}$	3.999 ± 0.002	$3.997^{+0.008}_{-0.007}$	4.004 ± 0.002	$4.004^{+0.010}_{-0.008}$
Set 3	4.000 ± 0.003	$4.006^{+0.012}_{-0.009}$	4.002 ± 0.003	$3.994^{+0.011}_{-0.009}$	4.000 ± 0.002	$3.992^{+0.008}_{-0.007}$	3.997 ± 0.002	$4.012^{+0.012}_{-0.009}$
Weighted Avg.	4.001 ± 0.002	4.011 ± 0.007	3.991 ± 0.002	3.992 ± 0.005	3.999 ± 0.001	3.997 ± 0.004	4.006 ± 0.002	4.003 ± 0.005
99.5% CL	3.996	3.993	3.987	3.978	3.996	3.986	3.997	3.991
min-H (NIST)	3.999		3.999		3.998		3.999	

Table 5.4: Measured entropy for the Repetition Count Test (RCT) and Adaptive Proportion Test (APT) at $\alpha = 2^{-20}$. All measured values are consistent with the theoretical entropy limit, considering expected statistical fluctuations for an unbiased source. The final row reports the minimum entropy estimated according to the NIST entropy source validation program [4]. Estimates derived from the proposed method are slightly lower, reflecting their more conservative uncertainty treatment.

5.5 FPGA Implementation of the Procedure

The direct implementation in FPGA of the procedure on firmware, as described in [3], enables the integration of the statistical tests within the bit generation logic, allowing on-line performance monitoring to occur concurrently with bit production. This architecture minimizes latency and preserves the generation rate, as the tests are executed in real time and provide early warning of anomalies without interrupting the bit-stream. The hardware-implemented blocks responsible for statistical test computations are structured as finite-state machines (FSMs), operating as sequential logic units in which each state depends on both the previous state and the current computational outcomes.

To enhance computational efficiency and minimize FPGA resource utilization, floating-point arithmetic was replaced with fixed-point operations. Fixed-point computation is inherently more suitable for FPGA architectures, as it requires fewer logic elements, reduces power consumption, facilitates pipelining, and simplifies algorithmic integration.

The core innovation of this implementation lies in the real-time monitoring of shifts in the mean of the failure distribution. This strategy lowers computational overhead while aligning more effectively with the goal of detecting long-term distribution drifts rather than isolated extreme deviations.

The implemented logic triggers an alert after three consecutive failures (configurable from 1 to 16). Assuming these failures are statistically independent and the threshold corresponds to a 3σ confidence level (false positive probability of 10^{-3}), the likelihood of three consecutive false alarms is $P = 10^{-9}$. At a 32 Mbit/s generation rate with 64 kbit test windows, this results in an expected false alarm roughly once every 69 days.

In terms of hardware resources, the APT and RCT modules together occupy approximately 300 Look-Up Tables (LUTs, 0.3% of the K26 FPGA),

90 registers ($< 0.1\%$), and 3 block RAM units (2.1%). The Monobit test requires 480 LUTs (0.4%), 260 registers (0.1%), and 1 block RAM (0.7%), while the RUNS test is comparatively more demanding, consuming 1730 LUTs (1.5%), 1900 registers (0.8%), and 3 block RAM units (2.1%).

5.5.1 RUNS Test

Among the statistical modules implemented in hardware, the RUNS test is the most computationally demanding. The test operates on configurable blocks of n bits, analyzing N consecutive sequences to provide a statistical assessment of stability over time.

As illustrated in Figure 5.12A, the hardware architecture is composed of a set of functional blocks managed by a FSM. The first stage, block (r1), counts the number of valid symbols processed, while block (r2) accumulates the number of ones in each sequence. Block (r3) identifies bit transitions by comparing consecutive values, thus computing the number of runs within the sequence. To correctly account for transitions that occur across symbol boundaries, the logic retains the last processed bit for comparison with the next symbol. These quantities are then combined to calculate the standardized test statistic z, defined in Eq. (5.5), which measures how far the observed data deviate from random expectation.

Unlike simpler tests such as the Monobit, APT, or RCT (which rely primarily on counting operations), the RUNS test requires variable mathematical operations at each iteration, preventing the use of precomputed constants and significantly increasing its computational load. To address this, two Xilinx IP cores are employed: a fixed-point CORDIC module to compute square roots iteratively, and a fixed-point divider for reciprocal evaluation. Although these introduce pipeline latency, they guarantee deterministic execution time and consistent throughput.

Where possible, arithmetic operations are optimized through fixed-point

scaling and bit-shifting, while lookup tables handle operations involving non-constant operands. The averaged test statistic is maintained by block (r4), while block (r5) continuously compares it against configurable statistical thresholds expressed as multiples of σ. If three consecutive averages exceed the confidence boundary, an anomaly flag is raised, enabling immediate system response.

The design supports flexible configurations, with sequence lengths ranging from 2^5 to 2^{11} bits and analysis windows from a few sequences up to over two million, allowing it to adapt to different operating conditions and sensitivity requirements.

5.5.2 Monobit Test

The Monobit test provides a more lightweight implementation, focused solely on verifying whether the proportion of ones in the bitstream remains statistically balanced. Its architecture follows the same general structure as the RUNS test but is considerably simpler in both logic and computational requirements.

As shown in Figure 5.12B, block (m1) counts the number of ones in each symbol, while block (m2) manages the FSM control logic, validating results once the entire n-bit sequence has been received. The statistic S_n, defined in Eq. (4.1), quantifies deviations from the expected 50% proportion of ones. After each sequence, the averaged value of S_n across N samples is compared to pre-established thresholds expressed as multiples of the standard deviation. When three consecutive averages exceed the confidence range, an anomaly flag is raised to signal a potential bias in the bit generation process.

The implementation supports sequence lengths between 2^2 and 2^{11} bits and can analyze up to several million sequences, with the additional constraint that n and N maintain the same parity to ensure numerical

symmetry in the thresholds.

5.5.3 Adaptive Proportion Test

In compliance with the NIST SP.800-90B guidelines, the implementation processes sequences of 512 symbols per test window.

As depicted in Figure 5.12C, the first block (a1) locks the initial symbol of each sequence as a reference. Subsequent symbols are compared against this reference by block (a2), which increments a counter whenever a match occurs. After all 512 symbols have been processed, block (a3) evaluates whether the number of matches lies within the acceptable range. Exceeding or falling below this threshold triggers an anomaly flag, indicating a potential bias toward one or more symbol values. The counter is initialized to one so that the reference symbol is included in the total count, consistent with the test's statistical formulation.

5.5.4 Repetition Count Test

The RCT operates at symbol level, allowing immediate anomaly detection without waiting for the completion of a full sequence, thus complementing the slower but more comprehensive RUNS test.

As illustrated in Figure 5.12D, block (c1) receives the symbol stream, while block (c2) stores the previous symbol to enable comparison with the current one. A repetition counter increments for each identical pair and resets to one when a change occurs or after every 512 samples, as prescribed by NIST specifications. Finally, block (c3) compares the current repetition count to a configurable limit. When the count exceeds this threshold, the module immediately asserts an anomaly flag, signaling the presence of a potential entropy degradation event.

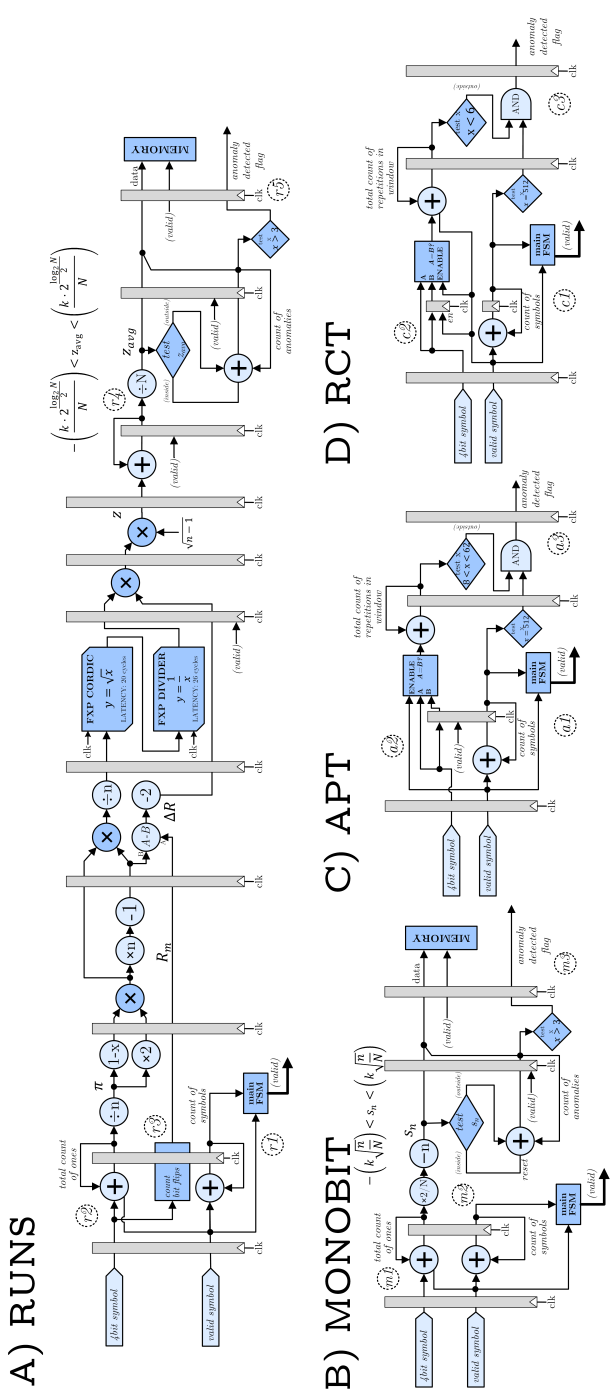


Figure 5.12: Block diagrams of the hardware-implemented statistical test accelerators. Each module is realized as a Finite-State Machine with Datapath. The RUNS accelerator is the most complex, as its outcome depends dynamically on the input bits, whereas the Monobit uses a simpler test statistic. The APT and RCT implementations require only counting and threshold comparison operations. In all cases, the p-value computation from the NIST reference design is omitted: the test statistics are directly compared against $\pm k\sigma$ thresholds derived from the Gaussian approximation of the expected distribution [3].

CHAPTER 6

Differential Privacy

As we've seen in previous chapters, randomness is fundamental to modern security and information systems. It's what allows the generation of encryption keys, transmission of data securely, and hiding sensitive information. We have explored how the quality of randomness directly determines whether cryptographic protocols actually work in practice, and why we need rigorous testing (both offline and online) to catch bias in our physical entropy sources.

However, the significance of randomness extends to other applications. It also plays a crucial role in privacy-preserving data analysis. In this case, randomness allows sharing meaningful statistical information about groups of people while protecting each individual's privacy.

Protecting sensitive information is a fundamental concern in communication and data management. Today, the challenge has evolved from protecting individual messages to safeguarding the personal data of millions of individuals stored in large digital datasets. For years, the typical approach was to remove direct identifiers (i.e., names, addresses, or social security numbers) from records before releasing data. However, as demonstrated by Narayanan and Shmatikov [51], even a few anonymous data points can be enough to re-identify people when combined with other publicly available information. The enormous amount of interconnected

data now circulating online has therefore made traditional anonymization methods ineffective, highlighting the need for mathematically rigorous forms of privacy protection [52].

To address this problem, *Differential Privacy (DP)* [53] stands out as a framework for its solid theoretical foundations and wide adoption. The core idea is that the output of an analysis should barely change whether any single individual's data is included or not. In practical terms, this means no one can determine if a specific person participated in a dataset, even when the analysis results are public. This property provides a strong, quantifiable form of privacy protection that remains valid even when an adversary possesses extensive background information.

Such guarantees are achieved by adding some carefully calibrated random noise to the results of computations. The amount of noise is governed by a parameter called the privacy loss (denoted by ε), which quantifies how much privacy is sacrificed for accuracy. A smaller ε means stronger privacy but noisier output data; a larger ε yields more accurate results but weaker privacy guarantees. In this way, randomness becomes the fundamental tool that balances these two competing needs, providing a probabilistic shield over individual contributions.

As emphasized in Chapters 3 and 5, the quality of randomness used in any security system is critical. Weak or biased entropy sources can compromise even well-designed algorithms. This concern extends naturally to DP mechanisms, which rely on random sampling to generate their protective noise. The distinction between TRNGs and PRNGs becomes particularly relevant here. While PRNGs are convenient and efficient, their deterministic nature raises an important question: could their use subtly weaken the theoretical guarantees of Differential Privacy?

This issue has gained particular attention after the U.S. Census Bureau adopted Differential Privacy to protect the 2020 census data [54]. The Bureau's *Disclosure Avoidance System (DAS)* [55] adds noise to demographic

counts before publication, marking one of the first large-scale governmental applications of the DP framework. In analyzing the mechanisms used for data release, it has been emphasized that using a PRNG to generate noise for sampling in Differential Privacy mechanisms used for census data release might not be sufficient to protect people's sensitive data. The algorithmic basis on which these numbers are generated could affect the actual quantification of privacy obtained. In fact, it is stated:

"The use of pseudo-random number generators is of special concern: using a pseudo-random number generator implies that the information-theoretic privacy-loss budget may be larger than claimed, as pseudo-random iterates are not independent in the strict sense required by information-theoretic definitions."

Similar concerns arise in the context of weak randomness models such as Santha–Vazirani sources [56], where each bit retains only partial unpredictability. A Santha–Vazirani (SV) source is a sequence of random bits in which the probability of each bit, conditioned on the outcomes of all previous bits, remains bounded within a fixed interval away from perfect unpredictability. Formally, for every bit X_i in the sequence and any history $x_1, \ldots, x_{i-1}$, there exists $0 < \gamma < \frac{1}{2}$ such that

$$\gamma \;\leq\; \Pr\big(X_i = 0 \mid X_1 = x_1, \ldots, X_{i-1} = x_{i-1}\big) \;\leq\; 1 - \gamma$$

for all possible histories [56]. Interestingly, it has been shown that DP can still be preserved with specific mechanisms under such imperfect randomness [57]. However, these claims have mostly remained theoretical, with limited empirical validation.

We therefore asked ourselves: despite proven analytical bounds, how sensitive are DP mechanisms to varying entropy quality in practice, and can we empirically measure this influence? To answer this, we conducted a systematic analysis measuring the empirical privacy guarantees achieved under different randomness sources. We established a rigorous experimental framework using open-source DP libraries, standard benchmark

parameters, and a controlled testing environment. We fixed common parameters (i.e., privacy budget, query types, and iteration counts) to ensure consistent comparisons. We then estimated the empirical privacy budget in the output relative to specified input parameters while systematically varying the underlying randomness source used for sampling. Our goal was to determine whether different quality levels of randomness produce measurable differences in output distributions and privacy guarantees.

The experimental methodology and results of this investigation are presented in detail in Chapter 7. In this chapter, we briefly present the theoretical background that defines Differential Privacy and the main mechanisms used to achieve it in practice, laying the groundwork for our analysis on the impact of entropy quality on privacy guarantees. We then examine a critical implementation challenge: how sampling procedures can be affected by floating point arithmetic, which in turn compromises the privacy guarantees of these mechanisms. This is an important addition because it shows that privacy guarantees can break down at multiple stages of the privatization process, not just at the level of randomness quality.

6.1 Theoretical Background

It is fundamental to have a mathematically rigorous privacy framework that overcomes the drawbacks of traditional approaches. We formally report the concept of Differential Privacy by first defining what a Randomized Algorithm is:

Definition 1 (Randomized Algorithm[53])**.** *A Randomized Algorithm $\mathcal{M}$ is an algorithm that employs a degree of randomness as part of its logic or procedure. Given input space $\mathcal{D}$ and output space $\mathcal{Y}$, the algorithm $\mathcal{M} : \mathcal{D} \to \mathcal{Y}$ outputs a sample from the distribution $M(\mathcal{D})$ over $\mathcal{Y}$.*

The algorithm is therefore non-deterministic in nature, as computations require harvesting some randomness to proceed, and the output probability

space is over the random choices made by the mechanism. This definition is fundamental because Differential Privacy achieves its guarantees through controlled randomization, where carefully calibrated noise is added to query results.

Given this preamble, we can now move on to formally define Differential Privacy:

Definition 2 (Differential Privacy[53])**.** *A randomized algorithm $\mathcal{M}$ is (ε, δ)-differentially private if, for all neighboring datasets $\mathcal{D}$ and $\mathcal{D}'$ (i.e., that differ by at most one element), and for all measurable subsets $\mathcal{S}$ of the output space $\mathcal{M}$:*

$$P(\mathcal{M}(\mathcal{D}) \in \mathcal{S}) \leq e^{\varepsilon} \cdot P(\mathcal{M}(\mathcal{D}') \in \mathcal{S}) + \delta \tag{6.1}$$

where ε is the privacy loss (or privacy budget) parameter and δ accounts for a small probability of failing the privacy guarantees.

Informally, this definition ensures that given two neighboring datasets fed as input of the mechanism, the distributions of the outputs are close, where this closeness is parameterized by the privacy loss ε. The randomization ensures that the presence or absence of any individual's data has minimal impact on the output distribution. A smaller ε provides stronger privacy guarantees by requiring the output distributions of neighboring datasets to be nearly identical, at the cost of reduced accuracy due to increased noise. Conversely, larger ε values allow for more accurate results but offer weaker privacy protection.

Intuitively, the interpretation of the definition is that by observing the result of the mechanism it is impossible to determine which dataset was given as input of the mechanism. The main idea is that by adding the right amount of noise to the results, it is possible to preserve statistical utility for aggregate analysis while protecting individual privacy.

The strongest form of this guarantee occurs when $\delta = 0$, which is referred to as Pure Differential Privacy, as the bounds are tight on ε. On the

other hand, when $\delta > 0$, we talk about Approximate Differential Privacy, which accounts for a small probability that the privacy guarantee might fail. δ captures failure events, and should be chosen small enough to make failures very unlikely (usually $\delta \ll 1/n$, where n is the number of samples in the dataset [58]). The trade-off is that with Approximate-DP, less noise is necessary to achieve the same privacy guarantees, yielding higher statistical utility to the data.

To better understand how Differential Privacy compares the output on neighboring datasets, it is useful to introduce a quantity that can capture the distance between two distributions. This measure is the *Privacy Loss Random Variable* (PLRV) [59, 60]. For a mechanism $\mathcal{M}$ and neighboring datasets $\mathcal{D}, \mathcal{D}'$, the privacy loss at output y is defined as

$$L_{\mathcal{M}}^{\mathcal{D},\mathcal{D}'}(y) = \ln\left(\frac{P(\mathcal{M}(\mathcal{D}) = y)}{P(\mathcal{M}(\mathcal{D}') = y)}\right).$$

This random variable measures how much more likely a given output is under $\mathcal{D}$ than under $\mathcal{D}'$. A mechanism is ε-DP exactly when the privacy loss is almost surely bounded by ε, and (ε, δ)-DP when this bound holds with probability at least $1 - \delta$. The PLRV therefore provides a precise tool to analyze privacy guarantees.

Before proceeding further, it is worth noting that there are two main deployment models for differential privacy: Global DP (also called Central DP) and Local DP [61]. In the global model, a trusted data curator collects raw data and applies privacy protection before releasing aggregate results. In the local model, each individual adds noise to their own data before sending it to an untrusted aggregator, providing stronger privacy guarantees since the aggregator never sees raw data, but at the cost of significantly reduced accuracy. Throughout this thesis, we focus exclusively on the global model, where noise is added to the final aggregate statistics after data collection.

It is interesting to note that differential privacy satisfies two fundamental properties that make it particularly robust and practical for real-world

applications: composition and post-processing immunity. These properties define the behavior of differential privacy when multiple queries are performed on the same dataset, or when combined with other operations.

When multiple queries are performed on the same dataset, the privacy guarantees degrade. The composition property formalizes this behavior giving exact bounds on the privacy loss. Under basic composition, if we have k mechanisms each satisfying ε-DP, their sequential composition satisfies $k\varepsilon$-DP, meaning that the privacy budget is consumed additively with each query. Approximate-DP provides tighter bounds with advanced composition theorems, showing that the privacy cost grows roughly at a $\sim \sqrt{k}$ rate rather than linearly, offering significant improvements for applications requiring many queries. Composition property shows that privacy degrades predictably when multiple computations are performed, and must be taken into account in real world applications.

The post-processing property ensures that any computation performed on the output of a differentially private mechanism cannot weaken the privacy guarantees, so long as the next computation is independent of the original data. Formally, if a mechanism $\mathcal{M}$ satisfies (ε, δ)-DP, then for any function g, the composition $g \circ \mathcal{M}$ also satisfies (ε, δ)-DP. This means that once the data has been made private, analysts can perform any kind of operation on differentially private results without compromising privacy, providing robustness against adversaries with auxiliary information.

6.2 Differential Privacy in Practice

Given the theoretical foundations of Differential Privacy, we now focus on presenting how it can be achieved in practice. The most commonly used mechanisms for implementing differentially private numerical queries are the Laplace mechanism and Gaussian mechanism.

The Laplace mechanism operates by adding noise sampled from a

Laplace distribution.

Definition 3 (Laplace Mechanism[53]). *Given a function $f : \mathcal{D} \to \mathbb{R}$, the Laplace mechanism for f with scale λ is defined as*

$$\mathcal{M}(u) = f(u) + Y;\ where\, Y \sim Lap(\lambda) \tag{6.2}$$

where $Lap(\lambda)$ denotes the Laplace probability distribution centered at zero, defined by the probability density function

$$g(\lambda; x) = \frac{1}{2\lambda} e^{-\frac{|x|}{\lambda}} \tag{6.3}$$

Specifically, for a query function f, the mechanism returns

$$f(\mathcal{D}) + Lap(\Delta f/\varepsilon) \tag{6.4}$$

where Δf represents the sensitivity of the function f, which measures the maximum change in a function's output when one person's data in the input is changed

$$\Delta f = \sup_{D,D'} |f(D) - f(D')| \tag{6.5}$$

for all neighboring datasets $\mathcal{D}$ and $\mathcal{D}'$. This sensitivity analysis is crucial because it determines the amount of noise required: queries with higher sensitivity need proportionally more noise to achieve the same level of privacy protection. What makes the Laplace mechanism appealing is how simple it is, yet it comes with strong privacy guarantees. It's a proven way to ensure ε-differential privacy while providing a practical framework for privatizing real-valued statistical queries.

We report the Theorem and proof exactly as described in [53], as this bound will be a core part of our analysis in Chapter 7.

Theorem 6.1. *The Laplace mechanism preserves ε-differential privacy.*

Proof. Let $x \in \mathbb{N}^{|\mathcal{X}|}$ and $y \in \mathbb{N}^{|\mathcal{X}|}$ be such that $\|x - y\|_1 \leq 1$, and let $f(\cdot)$ be some function $f : \mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^k$. Let p_x denote the probability density

function of $\mathcal{M}_L(x, f, \varepsilon)$, and let p_y denote the probability density function of $\mathcal{M}_L(y, f, \varepsilon)$. We compare the two at some arbitrary point $z \in \mathbb{R}^k$:

$$\begin{aligned}
\frac{p_x(z)}{p_y(z)} &= \prod_{i=1}^{k} \left(\frac{\exp\left(-\frac{\varepsilon|f(x)_i - z_i|}{\Delta f}\right)}{\exp\left(-\frac{\varepsilon|f(y)_i - z_i|}{\Delta f}\right)} \right) \\
&= \prod_{i=1}^{k} \exp\left(\frac{\varepsilon(|f(y)_i - z_i| - |f(x)_i - z_i|)}{\Delta f} \right) \\
&\leq \prod_{i=1}^{k} \exp\left(\frac{\varepsilon|f(x)_i - f(y)_i|}{\Delta f} \right) \\
&= \exp\left(\frac{\varepsilon \cdot \|f(x) - f(y)\|_1}{\Delta f} \right) \\
&\leq \exp(\varepsilon),
\end{aligned}$$

where the first inequality follows from the triangle inequality, and the last follows from the definition of sensitivity and the fact that $\|x - y\|_1 \leq 1$. That $\frac{p_x(z)}{p_y(z)} \geq \exp(-\varepsilon)$ follows by symmetry. □

Similarly to the Laplace mechanism, as the name suggests, the Gaussian mechanism adds noise sampled from a normal distribution. Since it is a proven procedure to achieve Approximate-DP, it is particularly well-suited for scenarios where more relaxed privacy guarantees are acceptable.

Definition 4 (Gaussian Mechanism[53])**.** *Given a function $f : \mathcal{D} \to \mathbb{R}$, the Gaussian mechanism for f with scale σ^2 is defined as*

$$\mathcal{M}(u) = f(u) + Y;\ where\, Y \sim \mathcal{N}(0, \sigma^2) \tag{6.6}$$

where $\mathcal{N}(0, \sigma^2)$ denotes the normal distribution with mean zero and variance σ^2.

For the Gaussian mechanism to satisfy (ε, δ)-DP, the noise scale must be set appropriately. Specifically, the mechanism returns

$$f(\mathcal{D}) + \mathcal{N}(0, \sigma^2) \tag{6.7}$$

where $\sigma^2 \geq \frac{2\Delta f^2 \ln(1.25/\delta)}{\varepsilon^2}$ and Δf is the L2-sensitivity of the function, defined as

$$\Delta f = \sup_{D,D'} \|f(D) - f(D')\|_2. \tag{6.8}$$

The key advantage of the Gaussian mechanism is that it often requires less noise than the Laplace mechanism for the same privacy level when $\delta > 0$, leading to an improved statistical utility. Additionally, the Gaussian mechanism is especially useful when making several queries, since it holds up better behavior under advanced composition theorems.

6.2.1 Discrete Mechanisms

While the Laplace and Gaussian mechanisms are typically introduced using continuous distributions, real-world implementations of Differential Privacy often use discrete versions instead. This is relevant for several practical reasons: query outputs are frequently integer-valued (like counts or histogram bins), floating-point sampling can introduce numerical errors, and some systems operate entirely on integer arithmetic.

Discrete Laplace Mechanism

The Discrete Laplace mechanism (known also as two-sided geometric mechanism [62]) replaces the continuous Laplace noise with an integer-valued random variable. The distribution's probability mass function is defined as:

Definition 5 (Discrete Laplace Distribution [63])**.** *Given the parameter $p = e^{-1/\sigma}$, where $\sigma > 0$ is known as the scale, a random variable $\mathcal{Y}$ has the discrete Laplace distribution with parameter $p \in (0, 1)$, denoted by DL(p), if*

$$f_p(k) = P(\mathcal{Y} = k) = \frac{1-p}{1+p}\, p^{|k|}, \qquad k \in \mathbb{Z} = \{0, \pm 1, \pm 2, \ldots\} \tag{6.9}$$

The definition of Discrete Laplace mechanism directly follows:

Definition 6 (Discrete Laplace Mechanism)**.** *Given a function $f : \mathcal{D} \to \mathbb{Z}$, privacy parameter ε, and sensitivity Δs, the discrete Laplace mechanism outputs*

$$\mathcal{M}(u) = f(u) + Y, \tag{6.10}$$

where $Y \sim DL(\Delta s/\varepsilon)$.

The discrete Laplace mechanism enjoys the same privacy guarantees as its continuous counterpart, and it satisfies pure ε-DP. Its main advantage is that the added noise is guaranteed to be integer-valued, avoiding potential precision loss from floating–point sampling. This makes it especially suitable for count queries and for systems that require exact reproducibility of results [64].

Discrete Gaussian Mechanism

Analogously, the Discrete Gaussian mechanism replaces continuous Gaussian noise with a distribution on the integers.

Definition 7 (Discrete Gaussian [65])**.** *Let $\mu, \sigma \in \mathbb{R}$ with $\sigma > 0$. The discrete Gaussian distribution with location μ and scale σ is denoted $\mathcal{N}_{\mathbb{Z}}(\mu, \sigma^2)$. It is a probability distribution supported on the integers and defined by*

$$\forall x \in \mathbb{Z}, \quad \mathbb{P}_{X \leftarrow \mathcal{N}_{\mathbb{Z}}(\mu,\sigma^2)}[X = x] = \frac{e^{-(x-\mu)^2/2\sigma^2}}{\sum_{y \in \mathbb{Z}} e^{-(y-\mu)^2/2\sigma^2}}, \tag{6.11}$$

where the definition of the mechanism simply follows as

Definition 8 (Discrete Gaussian Mechanism)**.** *Given a query $f : \mathcal{D} \to \mathbb{Z}$, the discrete Gaussian mechanism returns*

$$\mathcal{M}(u) = f(u) + Y, \tag{6.12}$$

where Y is sampled from the discrete Gaussian distribution with parameter σ.

The privacy guarantees of the discrete counterpart of the Gaussian mechanism are very similar to those of the continuous Gaussian [65].

Overall, discrete mechanisms offer a practical, numerically reliable alternative to their continuous counterparts, while maintaining the same privacy guarantees.

While discrete mechanisms address practical concerns about integer-valued outputs and numerical stability, they don't fully resolve the challenges of implementing DP on real computers. Even when mechanisms are mathematically correct, the gap between mathematical ideals and computational reality introduces subtle vulnerabilities. In particular, finite-precision floating-point arithmetic creates fundamental limitations that can compromise privacy guarantees.

6.3 Floating-Point Vulnerabilities in Differential Privacy

The definition of Differential Privacy is purely mathematical, defined over the real numbers. Moving to practical implementation on digital computers immediately introduces the fundamental challenge given by the discrete nature of computer arithmetic. Real numbers must be approximated using floating-point representation, typically following the IEEE 754 standard [66]. While this standard has enabled portability and predictable behavior across platforms, it creates a gap between mathematical theory and computational reality. This gap creates vulnerabilities that can compromise privacy guarantees if not properly addressed [67].

The core problem is that floating-point numbers are not uniformly spaced across their range. As defined by the IEEE 754 standard, floating-point representation provides higher density of representable values near zero and progressively sparser towards 1. For instance, there are more floats in the interval $[0, 0.5]$ than in $[0.5, 1]$. This creates a gap between mathematical theory and computational reality.

An additional challenge arises from integer-to-float conversion in sampling procedures. As demonstrated in [68], when using naive mappings of random integers to the floating-point interval [0,1], certain target values are over-represented: while only one integer input maps to 0.0, there are 257 different integer inputs that round to 0.5 (which has a single canonical IEEE-754 encoding). Combined with the finite precision of floating-point arithmetic, this means that when sampling noise for DP mechanisms, some floating-point values appear more frequently than others, while certain values may be entirely unreachable. This is simply because there are more possible real values in [0,1] than there are representable floats. The question of whether it's even possible to generate floating-point numbers uniformly in [0,1], understood as first sampling a real number, then rounding to the nearest representable value, has been a subject of ongoing research [69].

These concerns take on critical importance for Differential Privacy in light of Mironov's influential work [70]. The paper demonstrates that floating-point arithmetic actively leaks information through its limited precision. Moreover, floating-point operations like the natural logarithm further reduce numerical granularity, creating gaps in the output space. An attacker can exploit these gaps to break privacy guarantees: certain outputs become uniquely traceable to specific inputs, effectively making the noise addition procedure invertible. By observing only the outputs, an adversary can work backward to recover information about the original data, fundamentally undermining the privacy protection DP is meant to provide.

Mironov proposed a "Snapping Mechanism" that rounds outputs to counter floating-point vulnerabilities, but this approach proves computationally expensive and complex to implement.

Holohan et al. [71] proposed a faster alternative by exploiting the infinite divisibility of Laplace and Gaussian distributions. Their approach addresses both the inverse transform sampling vulnerability and makes

it computationally hard to recover original noise values. They extended this work in [72] with Mantissa Bit Manipulation, which manipulates the least significant bits of released values to prevent information leakage while maintaining high performance for applications like machine learning.

Subsequent research by Haney et al. [73] revealed that approaches based on sampling without rounding (including the method of Holohan et al.) remain vulnerable to a new class of precision-based attacks. These attacks exploit a property of IEEE 754 floating-point arithmetic: the unit in the last place (ulp) varies with the magnitude of a number. When adding noise to different input values, the precision of possible outputs depends on the input's magnitude. For instance, when adding Laplace noise to the value 1, all outputs are constrained to be multiples of 2^{-53}. In contrast, noise added to 0 is exactly the value of that noise. This discrepancy creates distinguishing events: an attacker observing an output that is not a multiple of 2^{-53} can deduce that the true input was not 1. The authors note that for Laplace noise with scale 1, approximately 25% of outputs constitute such distinguishing events.

To address these vulnerabilities, Haney et al. proposed interval refining, a technique that safely implements sampling by iteratively narrowing intervals until all values within the interval round to the same floating-point number, providing provable privacy guarantees equivalent to sampling from the real-valued distribution and rounding to the nearest float.

The challenge of constraining Differential Privacy implementations to the limits of finite computers, such as the limited precision of floating-point arithmetic and the difficulty of generating continuous random numbers, is at the center of the recommendations of the NIST SP 800-226 document [74]. The document emphasizes the importance of sampling techniques, discretization, and controlled approximations to ensure that the theoretical privacy properties are effectively maintained in practice, noting that an imprecise or systematically biased implementation can compromise privacy

guarantees.

This perspective aligns with the framework proposed by Balcer and Vadhan [64], which provides methods for transforming ideal privacy mechanisms into efficient, implementable algorithms. Both contributions emphasize that practical DP systems must be simultaneously computationally sustainable and formally correct from a privacy standpoint.

Finally, comprehensive recommendations on secure noise generation for DP applications are provided in [75], providing practical guidance for implementing systems that maintain both security and performance in real-world deployments.

These implementation challenges, from floating-point precision to secure random sampling, motivate our empirical investigation into how randomness quality affects DP mechanisms in practice. While the floating-point vulnerabilities described above can compromise privacy through numerical artifacts, our focus is on a complementary question: how does the quality of the underlying entropy source itself influence the privacy guarantees achieved by DP mechanisms?

6.4 Differential Privacy Libraries: addressing Finite Computer Constraints

The floating-point vulnerabilities and finite precision challenges discussed above are not merely theoretical concerns. They represent fundamental obstacles that any practical DP implementation must address. Given this premises, several open-source libraries were developed while carefully considering the gap between mathematical theory and computational reality with secure noise generation, discrete sampling, and the practical limitations of floating-point arithmetic.

As a preliminary step in our analysis, we surveyed the most widely adopted differential privacy libraries. This was necessary to ensure our

findings would be relevant across different implementation approaches and could reveal whether certain design choices make mechanisms more or less sensitive to randomness quality. The selection criteria was focused on adoption, maintenance, theoretical foundations, and documentation available, which ultimately led us to identify three libraries as the current state-of-the-art: OpenDP, Google's Differential Privacy, and IBM's DiffPrivLib.

6.4.1 OpenDP

OpenDP is a community-based, open-source project aimed at building a flexible and extensible platform for both differential privacy research and real-world applications [76]. The library employs a framework to define and verify privacy guarantees through a compositional type system, allowing users to construct complex privacy-preserving analyses by composing together several building blocks.

OpenDP provides implementations of core differential privacy primitives, including standard statistical queries and noise mechanisms such as Laplace and Gaussian. The core library is written in Rust, a language that provides strong memory safety guarantees and precise control over numerical operations, with Python and R bindings available [77].

Floating-point precision vulnerabilities are addressed with the implementation of discrete Laplace and Gaussian mechanisms that provide equivalent privacy guarantees to continuous noise while ensuring exact sampling [65].

6.4.2 IBM's DiffPrivLib

IBM's DiffPrivLib focuses mostly on machine learning applications and tight integration with Python's data science ecosystem [78]. The library is designed to work with scikit-learn and other standard tools, allowing data scientists to incorporate differential privacy into existing workflows with

minimal code changes.

The library implements secure Laplace and Gaussian sampling methods based on the work of Holohan et al. [71], which addresses both floating-point vulnerabilities and the invertibility of naive sampling procedures. The infinite divisibility properties of these distributions allow DiffPrivLib to construct sampling algorithms that are computationally efficient while being cryptographically hard to invert. It also implements the discrete Laplace (two-tailed Geometric [62]) for discrete differentially private operations.

6.4.3 Google's Differential Privacy Library

Google's differential privacy library is an open-source project designed to help developers and organizations analyze sensitive data while protecting individual privacy at scale [79]. Built on rigorous mathematical guarantees, the library supports both pure and approximate differential privacy and includes comprehensive tools for data preprocessing, aggregation, and statistical analysis.

The library is implemented in multiple languages: `C++`, `Java`, `Go`, and `Python`, and it also includes extensions such as a differentially private `SQL` interface, enabling privacy-preserving queries on large databases without requiring analysts to understand the underlying cryptographic details.

The library includes detailed documentation on sampling algorithms for Laplace and Gaussian distributions that explicitly address the limitations of standard floating-point implementations [80]. The library's noise generation procedures are designed to avoid the information leakage patterns using techniques that account for the discrete, non-uniform nature of representable floating-point values.

6.4.4 Design Philosophy

The common baseline around these libraries is the recognition that implementing differential privacy requires more than just translating mathematical definitions into code. The discreteness of computer arithmetic, non-uniform floating-point representations, and the impossibility of sampling true real numbers all demand careful attention. Each library addresses these constraints in their own, documented way.

While these libraries have carefully addressed floating-point and sampling challenges, they all depend on an underlying source of randomness. Our investigation explores a complementary question: regardless of how carefully the sampling procedure is designed, how much does the quality of the entropy source itself matter? Can weak randomness undermine privacy guarantees even when all other implementation details are correct? Sampling procedures expect that the underlying source of randomness is uniform. We want to measure how sensitive they are to a degradation of this uniformity.

These different implementation strategies provide us with an opportunity to test whether certain approaches are more or less sensitive to entropy quality degradation. In the following sections, we describe our experimental methodology for systematically testing these libraries under varying randomness conditions to empirically measure how entropy quality influences differential privacy guarantees in practice.

CHAPTER 7

Randomness Quality in Differential Privacy

As anticipated in Chapter 6, the objective of this research is to measure empirically the influence of entropy quality on the privacy guarantees provided by Differential Privacy mechanisms. While theoretical analyses have established bounds showing that DP can withstand certain types of imperfect randomness, these claims have remained largely untested in practice. The U.S. Census Bureau's concerns about using PRNGs in their DAS highlighted the gap between mathematical guarantees and real-world implementation uncertainties. To address this, we designed a systematic experimental framework that isolates the effect of randomness quality by testing two of the three major open-source DP libraries (i.e., OpenDP and IBM's DiffPrivLib) under controlled conditions. The Differential Privacy library developed by Google was excluded from the experimental analysis because timing constraints made it infeasible to perform a sufficiently thorough evaluation. We fixed common parameters including privacy budgets, query types, and iteration counts, while systematically varying the underlying entropy source. This chapter presents the results of these experiments, revealing how different levels of randomness quality affect the empirical privacy loss observed in practice and whether these effects are detectable through statistical analysis of output distributions.

The code used to perform the statistical analysis on the generated data is available on a dedicated Github repository [1].

7.1 Experimental Setup

This section describes the common framework we used across all experiments. By standardizing our computational environment, data sources, and testing parameters, we can make fair comparisons and isolate the effect of randomness quality from other variables that might confound our results.

7.1.1 Computation Environment

All experiments ran on a high-performance server with dual Intel Xeon Gold 5416S processors, providing 64 logical cores (32 physical cores with hyper-threading) at frequencies between 800 MHz and 4.0 GHz. The system has 128 GB of RAM organized in a NUMA (Non-Uniform Memory Access) architecture with two nodes for efficient memory allocation.

These resources were necessary given our experimental demands: millions of iterations for each combination of library, parameter set, and randomness source. The multi-core setup enabled parallel execution, while the large memory prevented bottlenecks when processing and storing the generated data. Samples were gathered with the currently most up-to-date version of each library (IBM's diffprivlib v.0.6.6; OpenDP v.0.14.0). All post-processing and statistical analysis were performed in MATLAB.

7.1.2 Randomness Sources

We used the same entropy sources described in Chapter 3: bits from a well-calibrated QRNG (`REF`), a cryptographically secure PRNG (`CSPRNG`), a miscalibrated QRNG (`MIS`), and manipulated versions of the well-calibrated

[1] https://github.com/grlcsr/dp_analysis

QRNG with varying degrees of `CORRELATION` introduced (`M4, M8, M16, M32, M64`), as defined in Section 3.4.3. As shown in Table 3.2, these sources have varying values of min-entropy, ranging from $H_{min} \geq 2.99$ for the manipulated entropy sources to $H_{min} \geq 3.99$ for our reference.

All three libraries were tweaked so that we could feed them our chosen randomness source, read directly from the binary files, instead of harvesting the randomness from their native RNG.

7.1.3 Query Selection and Datasets

To ensure fair comparisons across libraries, we designed a common query framework based on a fundamental differential privacy operation: counting the number of individuals in a dataset (length query). The choice was made due to the simplicity of the query and it's property of having a sensitivity always equal to 1. While each library uses its own syntax, we ensured the underlying mathematics remained consistent across implementations. This approach let us evaluate whether libraries produce theoretically equivalent results and how different randomness sources affect the output privacy loss both within and across libraries.

We selected two publicly available datasets: the Adult Income dataset [81] and the US 1990 Census dataset [82]. These represent common use cases in differential privacy applications. However, since our query simply counts individuals, the specific dataset content is actually irrelevant. What matters is demonstrating that such analyses can be conducted with readily available public data.

7.1.4 Parameters

We tested privacy budgets ranging from $\varepsilon = 0.001$ to $\varepsilon = 1$, covering both high-privacy (small ε) and more permissive regimes. This range reflects realistic privacy requirements in practical applications while providing

enough granularity to observe how ε interacts with different randomness sources.

Most of our analysis focuses on $\varepsilon = 0.1$, which offers a good trade-off between a reasonable sample size and achieving reliable accuracy while maintaining sufficient precision (granularity of 1) in the output distribution.

For our neighboring datasets, we used a fixed size of 10,000 individuals ($\mathcal{D}1$) and 10,001 individuals ($\mathcal{D}2$), representing the standard neighboring dataset definition where the datasets differ by exactly one person.

We ran 1 million iterations for each combination of parameters and randomness source (i.e., fixed $\varepsilon = 0.1$, $|\mathcal{D}1| = 10000$ and $|\mathcal{D}2| = 10001$, we perform 10^6 queries on $\mathcal{D}1$ and $\mathcal{D}2$ each). This large sample size ensures robust empirical estimation of output distributions and provides sufficient statistical power to detect meaningful differences between randomness sources that might otherwise be hidden by sampling noise.

All experiments use the Discrete Laplace mechanism, which provides pure differential privacy while also avoiding floating point operations. This choice lets us analyze how entropy variations affect the tightest possible bounds on ε.

7.1.5 Metrics

Measuring the influence of biased randomness on differential privacy requires careful consideration of what we mean by "influence". Our analysis focuses on two fundamental questions: first, how much bias is needed to produce measurable deviations from the theoretical model in terms of the Privacy Loss Random Variable? Second, even if the measured output distribution deviates from the expected Laplacian model, does this necessarily mean the privacy guarantee is broken?

Since Laplace sampling procedures expect uniformly distributed random input, we need statistical tools that can detect when this assumption is

violated and quantify the resulting impact on output distributions. Several metrics were considered during the design of our experimental framework. For example, we initially evaluated Spearman's rank correlation coefficient (a variant of Pearson's correlation for non-linear relationships), but this measure proved inadequate for our objectives: it assumes monotonic behavior between variables, which does not capture the kinds of distributional deviations we are interested in detecting.

After evaluating various alternatives, we settled on three complementary tests to assess whether the empirical distribution deviates significantly from the theoretical model:

- **Sign test**: A non-parametric test that detects systematic directional biases. The sign test identifies whether the output consistently skews in one direction relative to the theoretical model;

- $k\sigma$ **residual exceedance test**: We standardize the residuals $z_i = (O_i - E_i)/\sqrt{\sigma_i}$ between the empirical PLRV (observed, O_i) and theoretical distribution (expected, E_i) for each bin i. Under the hypothesis that our samples follow the theoretical model, the residuals should be approximately Gaussian. We then compute the fraction of bins where $|z_i| > k$ and compare this to the expected rate from a normal distribution for varying k (i.e., $k = 1, 2, 3$). Significant exceedance in terms of $k\sigma$ indicates poor fit of the distribution. These PLRV values are where the privacy guarantee might fail more likely, which means a higher risk of outputs that reveal more than the stated privacy budget should permit;

- χ^2 **goodness-of-fit test**: Evaluates whether the observed PLRV distribution matches the expected theoretical distribution. This test is sensitive to overall shape differences and provides a global measure of fit quality across all bins of the distribution [83].

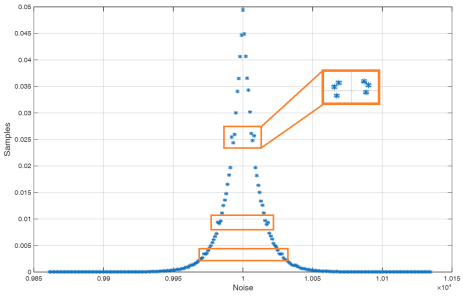
Together, these metrics allow us to quantify how well the empirical PLRV fits the theoretical model and determine the minimum amount of bias, according to our degradation models (`REF`, `CSPRNG`, `MIS`, `M4-M64`), needed to produce statistically significant deviations. Importantly, detecting such deviations does not automatically imply privacy failure; rather, it indicates where the gap between theory and practice becomes measurable, allowing us to assess the robustness margins of differential privacy mechanisms under realistic entropy conditions.

Moreover, the Kolmogorov-Smirnov test [84] was used to aid the analysis in certain cases. This test exploits the fact that the cumulative distribution function of any continuous random variable, when applied to samples from that distribution, will be uniformly distributed. The resulting empirical CDF can therefore be compared against the expected uniform distribution.

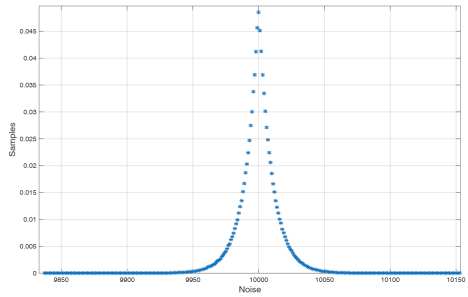
7.1.6 Methodological Considerations

This standardized setup allows us to make unbiased comparisons that isolate the effect of implementation choices from other factors that could affect the results. By maintaining consistency in data inputs, query formulations, and parameter settings, along with computation environment (which eliminates hardware-related variations), we can focus on the observation of potential differences in output distributions due to the selected randomness sources.

Furthermore, this approach allows us to evaluate also practical performance characteristics, including computational efficiency and numerical stability.



(a) OpenDP's Discrete Laplace with visible artifacts.



(b) IBM's diffprivlib Discrete Laplace.

Figure 7.1: Comparison of empirical Laplace distributions from OpenDP and IBM DiffPrivLib ($\varepsilon = 0.1$, $N = 10^6$ samples, bin size = 1, source of randomness: `REF`).

7.2 Model Validation

7.2.1 Reconstructing the Empirical Distribution

Our first step was to reconstruct the empirical Laplace distributions from the collected samples by building histograms. Recall that our query simply counts the number of individuals in the dataset, which has the useful property of always having sensitivity $\Delta s = 1$. Throughout this analysis, we focus on results for $\varepsilon = 0.1$. Given these two parameters, the scale of the Laplace distribution becomes $b = 1/\varepsilon = 10$.

Figure 7.1 shows the reconstructed histograms from $N = 10^6$ samples using a bin size of 1. An immediate difference appears between the two libraries. OpenDP's samples, shown in Figure 7.1a, exhibit evenly spaced artifacts that suggest a systematic implementation issue. These artifacts seemed to be independent of the source of randomness used be the sampling algorithm, as they were reproducible even by using the 2 integrated sources of the library itself. In contrast, IBM's discrete mechanism (Figure 7.1b) appears to reconstruct the discrete Laplace distribution accurately.

The choice of unitary bin size is critical here. Larger bins smooth over

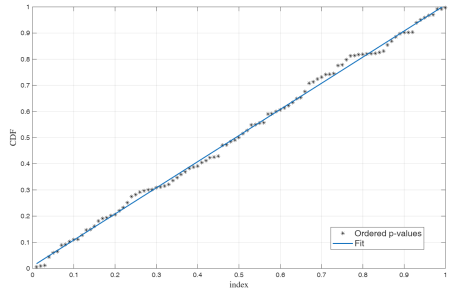


Figure 7.2: Cumulative distribution of p-values from 100 independent χ^2 tests shows the expected linear trend ($a = 0.999 \pm 0.005, b = 0.008 \pm 0.003$), confirming that IBM's samples follow the theoretical discrete Laplace distribution.

these artifacts and hide the problems we observed. Moreover, these patterns appeared consistently across different scale parameters, indicating a fundamental issue rather than a statistical fluctuation. We return to this anomaly and its resolution in Chapter 8.

7.2.2 Goodness-of-Fit Testing

To quantify how well these empirical distributions match the theoretical model, we performed χ^2 goodness-of-fit tests against the discrete Laplace PDF given in Equation 6.10. We repeated this test 100 times independently for each library to ensure our findings were robust.

The results confirmed what the histograms suggested. OpenDP consistently failed the χ^2 test across all 100 iterations, indicating a significant deviation from the theoretical distribution. In contrast, IBM's DiffPrivLib consistently produced reduced χ^2 values near 1, with p-values distributed as shown in Figure 7.2. The linear trend in the cumulative distribution of p-values confirms that the samples conform to the expected model, a conclusion further supported by a Kolmogorov-Smirnov test against the uniform distribution ($p = 0.93$).

Given these findings, we focus the remainder of our analysis on IBM's

differential privacy mechanism, which demonstrates reliable conformity to the theoretical model.

7.2.3 Computing the Privacy Loss Random Variable

To proceed with our analysis of entropy effects, we use the definition of differential privacy directly. We construct neighboring datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ centered on 10,000 and 10,001 individuals, respectively. For each dataset, we execute N queries and build the corresponding output histograms.

We then compute the empirical PLRV as the ratio of bin populations between these histograms. Formally, for each bin i, we compute:

$$PLRV_i = \frac{n_{1,i}}{n_{2,i}} \tag{7.1}$$

where $n_{1,i}$ and $n_{2,i}$ are the number of samples in bin i from queries on $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively. Note that we work directly with the ratio rather than its logarithm to simplify the uncertainty analysis while preserving the essential information about the relative likelihood of outputs under the two neighboring datasets.

7.2.4 Statistical Significance and Uncertainty Quantification

Before analyzing how different entropy sources affect differential privacy, we needed to establish that our measurement procedure itself is statistically sound. This required validating our uncertainty model and confirming that the observed variability in repeated experiments matches what we expect from sampling statistics.

Uncertainty Model

Our uncertainty analysis begins with the empirical PLRV, computed as the ratio of bin populations from the two histograms (rather than the logarithm of this ratio, which simplifies error propagation). For each bin, let n_1 denote

the number of samples from queries on $\mathcal{D}_1$ and n_2 the number from $\mathcal{D}_2$, with total sample size $N = 10^6$. Assuming binomial statistics for the histogram populations, the variances are $\sigma^2_{D_1} = n_1(1-n_1/N)$ and $\sigma^2_{D_2} = n_2(1-n_2/N)$.

The uncertainty on the PLRV for each bin is then computed by error propagation:

$$\sigma^2_{PLRV} = \frac{1}{n_2^2}\sigma^2_{D_1} + \frac{n_1^2}{n_2^4}\sigma^2_{D_2} \tag{7.2}$$

To validate this analytical error estimate, we repeated each experiment $M = 100$ times and computed the empirical standard deviation σ_{100} of the PLRV values across these repetitions. We then compared this empirical spread to our analytical prediction σ_{PLRV} by computing their ratio for each bin:

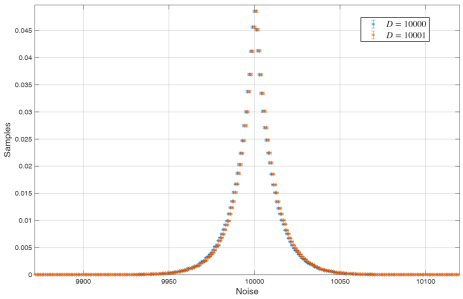
$$R = \frac{\sigma_{PLRV}}{\sigma_{100}} \pm \sigma_R \tag{7.3}$$

The uncertainty on this ratio, σ_R, was again computed by error propagation. The uncertainty on σ_{PLRV} comes from propagating uncertainties in n_1 and n_2:

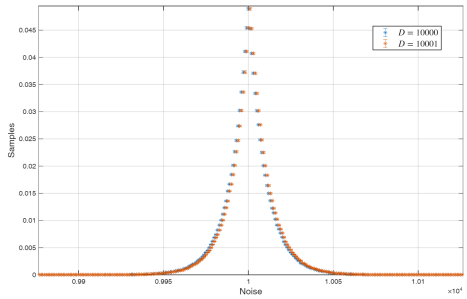
$$\begin{aligned}
\sigma^2_{\sigma^2_{PLRV}} &= \sum_{i\in\{1,2\}} \left[\frac{\partial \sigma^2_{PLRV}}{\partial n_i}\sigma_{D_i}\right]^2 \\
&= \left\{\frac{\partial}{\partial n_1}\left[\frac{n_1}{n_2^2}\left(1-\frac{n_1}{N}\right) + \frac{n_1^2}{n_2^3} - \frac{n_1^2}{N n_2^2}\right]\right\}^2 \sigma^2_{D_1} \\
&\quad + \left\{\frac{\partial}{\partial n_2}\left[\frac{n_1^2}{n_2^2}\left(\frac{1}{n_1}+\frac{1}{n_2}-\frac{2}{N}\right)\right]\right\}^2 \sigma^2_{D_2} \\
&= \left(\frac{-4n_1n_2 + 2n_1N + n_2N}{n_2^3 N}\right)^2 \sigma^2_{D_1} + \\
&\quad + \left(\frac{n_1(4n_1n_2 - 3n_1N - 2n_2N)}{n_2^4 N}\right)^2 \sigma^2_{D_2}
\end{aligned} \tag{7.4}$$

The standard error on σ_{100} follows from the Gaussian approximation:

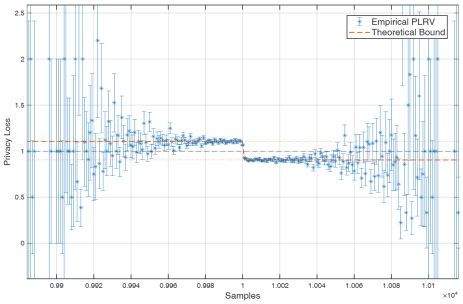
$$\text{se}(\sigma_{100}) = \frac{\sigma_{100}}{\sqrt{2(M-1)}} \tag{7.5}$$



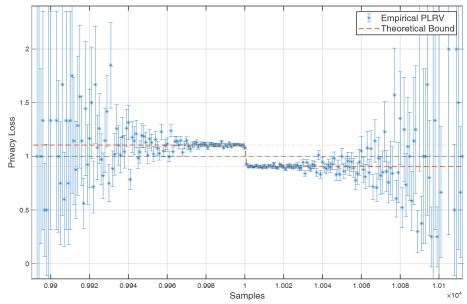
(a) Histograms with `REF` entropy source.



(b) Histograms with `CSPRNG` entropy source.



(c) PLRV computed from `REF` entropy.



(d) PLRV computed from `CSPRNG` entropy.

Figure 7.3: Top row: Output histograms from queries on $\mathcal{D}_1$ (blue, centered at 10,000) and $\mathcal{D}_2$ (orange, centered at 10,001) using frozen entropy. The histograms are nearly identical, differing only by the unit shift. Bottom row: Corresponding empirical PLRV distributions (blue: empirical data with error bars corresponding to 1σ CI; orange dashed lines: theoretical bound at $e^{\pm\varepsilon}$). Left column shows results with `REF` entropy, right column with `CSPRNG` entropy.

If our binomial uncertainty model is correct, we expect $R \approx 1$ for most bins, with deviations confined to regions where sampling noise dominates (primarily in the distribution tails where bin populations are small).

Frozen Entropy Validation

To establish a baseline for comparison, we first ran experiments with frozen initial conditions. By feeding identical entropy sequences to both the $\mathcal{D}_1$

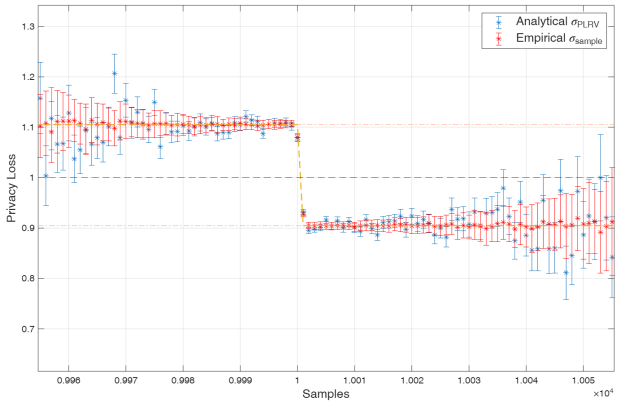


Figure 7.4: Ratio of analytical uncertainty (σ_{PLRV}, blue) to empirical standard deviation (σ_{100}, orange) across bins. Values near 1 confirm the validity of the binomial error model. Larger deviations appear in the tails where bin populations are small and statistical noise dominates.

and $\mathcal{D}_2$ queries, we produced two identical histograms shifted by exactly one bin. This controlled setup isolates the uncertainty purely from finite sampling, removing any variability from the randomness source itself.

Figure 7.3 shows these frozen-entropy results for both `REF` (our QRNG) and `CSPRNG` (OpenSSL). The top row displays the nearly overlapping histograms, while the bottom row shows the computed PLRV. As expected for pure differential privacy with $\varepsilon = 0.1$, the privacy loss is bounded within $\pm\varepsilon$ in the central region where most samples concentrate. The tails become noisier due to lower bin populations, reflected in the larger error bars. This effect could potentially be mitigated by increasing bin sizes in the tails (for example, by enforcing a minimum expected bin population), which may improve the statistical significance of tail estimates.

Figure 7.4 shows the ratio $R = \sigma_{PLRV}/\sigma_{100}$ across all bins for the frozen entropy case. The blue points represent the analytical binomial error σ_{PLRV}, while the orange points show the empirical standard deviation σ_{100} from 100 independent repetitions. The ratios cluster around 1, confirming that our

Source	Bins beyond 2σ	Percentage	p-value
`REF` (frozen)	9/181	4.97%	0.31
`CSPRNG` (frozen)	7/183	3.85%	0.59

Table 7.1: Statistical validation with frozen entropy. The fraction of bins exceeding 2σ deviation is consistent with random fluctuations (expected $\approx 5\%$), confirming the binomial uncertainty model.

Source	Bins beyond 2σ	Percentage	p-value
`REF`	12/181	6.63%	0.07
`CSPRNG`	9/183	4.93%	0.32

Table 7.2: Statistical validation with independent entropy sampling. Results remain consistent with the binomial model, showing that entropy variability does not introduce additional systematic uncertainty beyond what sampling statistics predict. In particular, for the observed number of bins, values in the range of a few percent around 5% are consistent with binomial variability and only statistically significant excesses would indicate additional systematic uncertainty.

binomial uncertainty model accurately describes the sampling variability.

To quantify the agreement to the model, we counted how many bins show ratios deviating from 1 by more than 2σ. Under the null hypothesis that our model is correct, we expect roughly 5% of bins to exceed this threshold by chance. Table 7.1 shows the results.

Both entropy sources show exceedance rates consistent with the expected 5%, with p-values well above conventional significance thresholds. This validates our uncertainty quantification approach.

Independent Sampling Validation

Having established the baseline with frozen entropy, we repeated the validation with unfrozen conditions, allowing each of the 100 experimental repetitions to sample independently from the entropy source. This repre-

sents the realistic scenario where each query draws fresh randomness.

Table 7.2 shows these results. Again, both `REF` and `CSPRNG` sources produce exceedance rates compatible with statistical fluctuations, confirming that the observed spread across experiments matches our binomial model even when entropy varies between runs.

The consistency between frozen and unfrozen results demonstrates two important points. First, our uncertainty model based on binomial statistics is sound. Second, high-quality entropy sources (both quantum TRNG and cryptographic PRNG) produce statistically indistinguishable results under these conditions, establishing a validated baseline for comparing against degraded entropy sources in the following sections.

7.3 Entropy Quality Impact Analysis

Having established the statistical validity of our measurement framework, we now evaluate how different entropy sources affect the empirical privacy loss. We begin with high-quality sources (`REF` and `CSPRNG`) before examining the impact of degraded entropy (`MIS` and `M4`–`M64` manipulations).

To move forward with the analysis, we need to make several considerations. First, bins in the distribution tails are inherently less reliable due to smaller populations and larger statistical errors. Second, we must choose binning carefully. While unitary binning (bin size = 1) is natural for discrete mechanisms, larger bins require that the midpoint between the two histogram peaks falls on a bin edge to avoid mixing data from opposite sides of the boundary. Finally, we need to exclude empty bins from all computations to avoid division by zero, infinities, and undefined logarithms.

7.3.1 Sign Test

The sign test provides a simple yet powerful non-parametric approach to detecting systematic biases in the PLRV distribution [85]. Unlike the χ^2 test,

which is sensitive to the size of deviations, the sign test focuses purely on whether deviations from the theoretical model are symmetric around zero or show a directional bias.

For each bin i where both neighboring dataset histograms are non-zero, we compute the deviation between the empirical privacy loss ratio and its theoretical value. Under the null hypothesis that the mechanism provides ε-DP with high-quality randomness, positive and negative deviations should occur with equal probability. The number of positive deviations k_{pos} out of n bins therefore follows approximately a Binomial$(n, 0.5)$ distribution. The sign test checks the symmetry of privacy loss fluctuations without assuming any particular distribution or relying on amplitudes.

We apply this test in two ways: first, across the entire PLRV histogram considering all adequately populated bins; second, focusing on the central 68% of the distribution (corresponding to 1σ from the mean) around the main peak to reduce noise from the noisier tails. Both analyses use bin size equal to 1, which provides maximum resolution for detecting asymmetries. We also report the p-value of the fraction of bins that violate the ε bound both in the linear and log-space and computing $\ln(\text{ratio})$, testing if it significantly violates the 0.5 assumption.

Baseline Entropy Sources

Tables 7.3 and 7.4 present sign test results for the unmanipulated entropy sources. All three sources (`REF`, `CSPRNG`, and `MIS`) yield p-values above conventional significance thresholds (typically between 0.16 and 0.8), indicating no detectable asymmetry in the privacy loss deviations.

Restricting the analysis to the central 68% of the histogram reduces the number of bins but does not change this conclusion. The sign test cannot distinguish between `REF`, `CSPRNG`, and `MIS` at bin size = 1, consistent with all three sources providing adequate randomness quality for the differential

Table 7.3: Sign test results on full PLRV histogram (bin size 1) for all entropy sources. n is the number of bins, k_{pos} is the number of bins with positive deviations, p tests the fraction of bins violating the ε-band, $p_{\log}$ tests centering at $\pm\varepsilon$ in log-space, and $H_{\min}$ is the min-entropy.

(a) Baseline entropy sources

Source	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
REF	209	96	0.2684	0.8900	3.9995
CSPRNG	213	98	0.2729	0.5836	3.9995
MIS	214	99	0.3052	0.6323	3.9954

(b) CLEAR

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	102	84	2.257e-11	0.1375	–
M8	170	115	4.870e-06	0.9389	–
M16	203	106	0.5746	0.0170	–
M32	210	105	1.0000	0.3006	–
M64	208	102	0.8353	0.2983	–

(c) SET

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	333	99	9.577e-14	0.0007	–
M8	236	81	1.673e-06	0.0009	–
M16	225	78	4.941e-06	0.0329	–
M32	222	95	0.0372	0.2539	–
M64	218	96	0.0902	0.2496	–

(d) FLIP

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	274	94	2.266e-07	0.0611	2.9998
M8	213	87	0.0091	0.1001	3.4149
M16	209	87	0.0185	0.0269	3.6780
M32	218	89	0.0081	0.1549	3.9021
M64	213	102	0.5837	0.5836	3.9722

(e) ALTERNATE

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	153	64	0.0520	0.5178	3.4148
M8	202	87	0.0572	0.0092	3.6779
M16	212	91	0.0462	0.6307	3.8301
M32	217	99	0.2217	0.6838	3.9503
M64	212	107	0.9453	0.5365	3.9861

(f) CORRELATION

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	125	100	8.085e-12	0.5915	3.1926
M8	208	112	0.2983	1.0000	3.7518
M16	207	98	0.4871	0.4042	3.9124
M32	214	101	0.4522	0.9455	3.9719
M64	210	93	0.1123	0.4478	3.9913

Table 7.4: Sign test results on central 68% of PLRV histogram (bin size 1) for all entropy sources. This analysis excludes the noisiest tail bins to focus on the central region.

(a) Baseline entropy sources

Source	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
REF	142	73	0.8013	0.9331	3.9995
CSPRNG	145	66	0.3190	0.5065	3.9995
MIS	146	64	0.1592	0.8039	3.9954

(b) CLEAR

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	69	59	1.377e-09	0.0533	2.9998
M8	116	80	5.400e-05	0.5157	3.4151
M16	138	79	0.1055	0.0333	3.7518
M32	143	79	0.2416	0.0944	3.8399
M64	141	74	0.6135	0.1296	3.9123

(c) SET

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	226	55	4.900e-15	0.0139	2.4149
M8	160	53	2.361e-05	0.0091	2.9127
M16	153	51	4.554e-05	0.0753	3.5739
M32	151	69	0.3288	0.4158	3.8403
M64	148	69	0.4595	0.4594	3.9229

(d) FLIP

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	186	55	2.462e-08	0.1236	2.9998
M8	145	54	0.0027	0.0967	3.4149
M16	142	55	0.0090	0.4501	3.6780
M32	148	66	0.2174	0.2176	3.9021
M64	145	77	0.5066	0.5065	3.9722

(e) ALTERNATE

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	104	37	0.0042	0.6239	3.4148
M8	137	58	0.0871	0.3053	3.6779
M16	144	68	0.5598	0.8026	3.8301
M32	148	73	0.9345	0.6811	3.9503
M64	144	76	0.5598	0.5597	3.9861

(f) CORRELATION

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	85	76	2.405e-14	0.8284	3.1926
M8	141	82	0.0635	0.2384	3.7518
M16	141	70	1.0000	0.8662	3.9124
M32	146	68	0.4565	0.8039	3.9719
M64	143	71	1.0000	0.4030	3.9913

privacy mechanism under these conditions.

Manipulated Entropy Sources

When we examine entropy-manipulated datasets, the picture changes. Tables 7.3 and 7.4 show results for all manipulation types across different entropy levels.

Strong manipulations (`M4` and `M8`) produce extremely small p-values (often $p < 10^{-5}$) across most manipulation types, signaling clear and highly significant directional bias. This pattern appears consistently in both full-histogram and central-68% tests, showing that the asymmetry is a global feature rather than an artifact of noisy tail bins. The `CLEAR`, `SET`, `FLIP`, and `CORRELATION` manipulations are particularly detectable, with the sign test rejecting the null hypothesis decisively.

For moderate manipulations (`M16`), detection becomes less reliable. Some manipulation types still result in significant p-values, while others are more ambiguous. This reflects the fact that these manipulations are less impactful on the bit-stream, and their effects on the PLRV distribution become more subtle.

Mild manipulations (`M32` and `M64`) remain mostly undetected by the sign test. Most p-values return to the non-significant range, suggesting that the directional bias introduced by these sparse manipulations is insufficient to produce detectable asymmetry at our sample size of $N = 10^6$. This establishes a clear sensitivity gradient: the sign test reliably detects strong, frequent manipulations but loses power for sparse or subtle perturbations.

The `ALTERNATE` manipulation behaves differently than the rest. Even at `M4`, it produces borderline p-values (around 0.05) rather than the decisive rejection seen with other manipulation types. This suggests that alternating-pattern manipulations may partially cancel in their effect on directional bias, making them harder to detect with a purely sign-based test.

To check the robustness of our findings, we repeated the sign test using a larger bin size of 5. Increasing the bin width naturally reduces the total number of bins and smooths out some of the local fluctuations in the data, which can slightly decrease the test's statistical power. However, this change also provides a valuable consistency check to confirm our results. Overall, the conclusions remain largely unchanged: baseline entropy sources (`REF`, `CSPRNG`, and `MIS`) still show non-significant p-values, indicating no directional bias; strong manipulations (`M4` and `M8`) continue to be detected clearly across most types; and weaker manipulations (`M16`, `M32`, and `M64`) become harder to identify, as expected due to the coarser binning. Notably, the `ALTERNATE` manipulation remains borderline even at higher manipulation intensities. Full results for the bin size of 5 are available in Appendix C.

7.3.2 Sigma Exceedance Test

The sigma exceedance test is very similar to the sign test (count statistic of how many bins fluctuate significantly from theoretical model) but also considers the amplitude relative to the expected statistical uncertainty. Rather than simply asking whether a bin's empirical privacy-loss ratio sits above or below the theoretical plateau, we measure how many standard deviations away from the plateau it lies.

For each populated bin i, we compute a z-score by comparing the empirical privacy-loss ratio (or its logarithm) to the theoretical value predicted by the ε-DP model:

$$z_i = \frac{\text{estimate}_i - \text{theory}_i}{\sigma_i},$$

where σ_i represents the uncertainty propagated from the binomial counting errors in the two neighboring histograms. If the mechanism truly satisfies ε-DP, these z-scores should behave like standard normal fluctuations: we expect roughly 31.7% of bins to exceed 1σ and about 4.6% to exceed 2σ purely by chance. Given n bins, the number of exceedances k_{out} at a

particular threshold should follow a binomial distribution. We compute one-sided binomial p-values to test whether the observed fraction of outliers is compatible with these expected rates.

We apply this test in multiple configurations. The main results appear in Tables 7.5to 7.10 and Tables D.1 to D.6, which report exceedance rates for bin size 1 across the full histogram and the central 68% region. Additional results with bin size 5 are provided in Appendix D. For each configuration, we perform the test both in the original ratio space and in log-space by replacing ratios with $\ln(\text{ratio})$.

Linear vs. Log-Space Analysis

We run the sigma exceedance test in two ways: directly on the privacy-loss ratios (linear space) and on their logarithms (log-space). These aren't the same test. The logarithm is nonlinear, so it compresses large ratios and changes how the z-scores behave. Even though both variants probe the same underlying DP condition, they can give you different p-values.

It is easy to notice is that for the baseline entropy sources, the log-space test looks adherent to the model: both 1σ and 2σ exceedance rates sit close to the expected 31.7% and 4.6%, with p-values above the significance level. The linear-space test, on the other hand, systematically inflates the number of 2σ exceedances and produces some significant deviations ($p \ll 0.01$). This might signify the Gaussian error model works better in log-space than in the linear ratio domain.

For strongly manipulated datasets, this distinction doesn't matter much, as both tests detect violations clearly, but for borderline cases, the choice matters. We'll focus primarily on the linear-space results (which are more conservative) but report log-space values in the tables for completeness.

Table 7.5: $k\sigma$ exceedance test results on full PLRV histogram (bin size 1) for baseline entropy sources. For each test level, we report the observed exceedance rate, its p-value, and the p-value in log-space.

Source	**1σ exceedance**			**2σ exceedance**		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`REF`	0.306	0.603	0.660	0.067	0.056	0.245
`CSPRNG`	0.300	0.673	0.443	0.070	0.035	0.175
`MIS`	0.308	0.578	0.349	0.051	0.270	0.642

Baseline Entropy Sources

Table 7.5 presents sigma exceedance test results for the baseline entropy sources (`REF`, `CSPRNG`, `MIS`) on the full histogram. For all three, the fraction of 1σ exceedances remains close to the expected 31.7%, with p-values typically between 0.16 and 0.8, indicating statistical fluctuations consistent with theoretical model.

At the 2σ level, a slight increase is observed in some configurations, notably for `CSPRNG`, where exceedance fractions slightly exceed the nominal 4.6% and corresponding p-values fall below 0.05. However, this effect is modest and inconsistent across tests. The miscalibrated QRNG (`MIS`) exhibits no significant outlier excess at either threshold, maintaining p-values above significance levels.

Limiting the analysis to the central 68% of the histogram (Table D.1) reduces bin count but does not alter overall results. None of the baseline sources display systematic deviations from expected exceedance rates. Results for bin size 5 (Appendix D) are similar. Overall, the sigma exceedance test confirms that baseline entropy sources yield PLRV distributions consistent with the ε-DP model.

Table 7.6: $k\sigma$ exceedance test results on full PLRV histogram (bin size 1) for `CLEAR` manipulations.

Level	1σ exceedance			2σ exceedance		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.696	9.93e-16	9.50e-19	0.441	3.56e-34	6.05e-33
`M8`	0.353	0.140	0.059	0.147	6.80e-08	1.70e-08
`M16`	0.291	0.769	0.813	0.099	4.28e-04	0.046
`M32`	0.338	0.234	0.192	0.105	1.01e-04	0.058
`M64`	0.312	0.526	0.526	0.067	0.054	0.608

Table 7.7: $k\sigma$ exceedance test results on full PLRV histogram (bin size 1) for `SET` manipulations.

Level	1σ exceedance			2σ exceedance		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.471	1.68e-09	4.24e-10	0.258	1.25e-40	4.74e-33
`M8`	0.407	0.002	2.31e-04	0.174	4.42e-14	9.62e-10
`M16`	0.360	0.075	0.097	0.107	4.08e-05	0.002
`M32`	0.338	0.232	0.232	0.099	2.30e-04	0.214
`M64`	0.344	0.178	0.068	0.064	0.075	0.535

Manipulated Entropy Sources

The sensitivity of the sigma exceedance test is most evident when analyzing entropy-manipulated datasets, as summarized in Tables 7.6 to 7.10 for the full histogram and Tables D.2 to D.6 for the central 68% region.

For strong manipulations (`M4`), both 1σ and 2σ exceedance rates claerly increase. Within the central 68% region, over 80% of data points exceed the 1σ threshold, and up to 50% exceed 2σ. This pattern occurs consistently across manipulation types (`CLEAR`, `SET`, `FLIP`, `CORRELATION`) and remains

Table 7.8: $k\sigma$ exceedance test results on full PLRV histogram (bin size 1) for `FLIP` manipulations.

Level	**1σ exceedance**			**2σ exceedance**		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.409	5.73e-04	8.86e-05	0.197	1.98e-20	3.00e-16
`M8`	0.347	0.154	0.041	0.117	5.47e-06	0.004
`M16`	0.330	0.316	0.370	0.086	0.003	0.159
`M32`	0.330	0.312	0.218	0.073	0.023	0.196
`M64`	0.315	0.502	0.331	0.056	0.175	0.636

Table 7.9: $k\sigma$ exceedance test results on full PLRV histogram (bin size 1) for `ALTERNATE` manipulations.

Level	**1σ exceedance**			**2σ exceedance**		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.386	0.030	0.005	0.183	7.03e-11	1.63e-09
`M8`	0.386	0.016	0.132	0.074	0.023	0.315
`M16`	0.255	0.972	0.996	0.061	0.106	0.372
`M32`	0.304	0.631	0.517	0.088	0.002	0.041
`M64`	0.349	0.143	0.050	0.075	0.018	0.372

Table 7.10: $k\sigma$ exceedance test results on full PLRV histogram (bin size 1) for `CORRELATION` manipulations.

Level	**1σ exceedance**			**2σ exceedance**		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.608	6.61e-12	7.49e-10	0.344	2.76e-27	3.63e-23
`M8`	0.293	0.747	0.898	0.087	0.003	0.349
`M16`	0.329	0.334	0.390	0.072	0.028	0.343
`M32`	0.294	0.739	0.635	0.047	0.383	0.761
`M64`	0.324	0.388	0.445	0.057	0.163	0.250

present when increasing the bin size, indicating the effect is characteristic of the PLRV distribution rather than noise in the tails.

At moderate levels (M8), the test remains highly sensitive. The fraction of 2σ exceedances surpasses the nominal 4.6%, with p-values often between 10^{-5} and 10^{-10}; 1σ exceedances are also high across manipulation types. These deviations are visible in both full and central histograms, and consistent in linear and logarithmic scales. Compared to the sign test, which detects directional imbalance, the sigma exceedance test also highlights changes in privacy-loss magnitude inconsistent with the ideal model.

At M16 manipulations, detection becomes subtler. Several manipulation types still produce significant excess 2σ outliers, while 1σ exceedance rates generally fall within expected ranges. This reflects the milder impact at this level.

Mild manipulations (M32 and M64) typically fall below the test's sensitivity at $N = 10^6$. Exceedance rates fluctuate near nominal values, with mostly non-significant p-values. Some M32 cases show borderline 2σ excesses at bin size 1, suggesting this is the test's sensitivity limit. At M64, exceedance rates align with unmanipulated data.

The ALTERNATE manipulation shows less clear detection levels than CLEAR, SET, or FLIP for comparable entropy loss, having borderline p-values even at M4, implying that alternating patterns partially cancel out in their influence on PLRV magnitudes, reducing detectability via threshold exceedance.

7.3.3 χ^2 Goodness-of-Fit Test

The final test applied is the χ^2 goodness-of-fit, which quantifies how well the observed PLRV histogram fits the theoretical Laplace distribution from

the ε-DP model. The reduced chi-squared statistic is computed as

$$\chi^2_{\text{red}} = \frac{1}{n_{\text{dof}}} \sum_{i=1}^{n_{\text{bins}}} \frac{(\text{observed}_i - \text{expected}_i)^2}{\sigma_i^2},$$

where n_{dof} is degrees of freedom. Under the null hypothesis, χ^2_{red} should approximate 1, supported by the p-value.

Analysis is performed in both linear and log-space. Key results appear in Tables 7.11 and 7.12 (bin size 1), with additional bin size 5 results in Appendix E.

Baseline Entropy Sources

Baseline results in linear space show χ^2_{red} slightly above 1; for example, `REF` reports 1.19 ($p = 0.027$), formally rejecting the null due to sensitivity to noisy histogram tails rather than model failure. In log-space, `REF` aligns with the model ($\chi^2_{\text{red}} = 0.98$, $p = 0.579$), and other sources follow similar trend.

Restricting to the central 68% reduces bins, reducing linear-space deviations: `REF` shows $p = 0.084$ and `MIS` $p = 0.074$, both non-significant. `CSPRNG` exhibits a slight deviation ($\chi^2_{\text{red}} \approx 1.32$, $p = 0.006$). Bin size 5 results show elevated p-values (0.1 to 0.9), indicating good model fit with smoother histograms.

With bin size 5 (Tables E.1 and E.2), all p-values rise substantially, typically ranging from 0.1 to 0.9. The coarser binning smooths out local fluctuations and noise in the tails, and the theoretical Laplace model describes the data very well at this resolution. For `CSPRNG` in particular, the full histogram with bin size 5 yields $\chi^2_{\text{red}} = 0.68$ and $p = 0.96$—an essentially perfect fit.

Thus, baseline sources are fundamentally consistent with the ε-DP model, particularly in log-space and wtih larger binning. The linear-space bin size 1 rejections reflect sensitivity to tail noise.

Table 7.11: χ^2 goodness-of-fit test results on full PLRV histogram (bin size 1) for all entropy sources. χ^2_{red} is the reduced chi-squared statistic, p is the p-value in linear space, $\chi^2_{\text{red,log}}$ is the reduced chi-squared in log-space, and $p_{\log}$ is the corresponding p-value in log-space.

(a) Baseline entropy sources

Source	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
REF	1.193	0.027	0.978	0.579
CSPRNG	1.156	0.059	1.108	0.134
MIS	1.086	0.185	1.066	0.243

(b) CLEAR

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	9.903	0.000	9.987	0.000
M8	2.789	0.000	2.812	0.000
M16	1.691	0.000	1.286	0.004
M32	1.550	0.000	1.134	0.089
M64	1.598	0.000	1.092	0.174

(c) SET

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	14.410	0.000	12.391	0.000
M8	5.936	0.000	5.305	0.000
M16	3.203	0.000	2.740	0.000
M32	1.817	0.000	1.548	0.000
M64	1.487	0.000	1.227	0.012

(d) FLIP

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	13.325	0.000	11.183	0.000
M8	3.220	0.000	2.849	0.000
M16	1.805	0.000	1.595	0.000
M32	1.404	0.000	1.171	0.043
M64	1.369	0.000	1.089	0.177

(e) ALTERNATE

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	10.230	0.000	9.213	0.000
M8	2.582	0.000	2.176	0.000
M16	1.363	0.000	1.193	0.029
M32	1.390	0.000	1.163	0.050
M64	1.708	0.000	1.246	0.009

(f) CORRELATION

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	16.830	0.000	18.549	0.000
M8	1.524	0.000	1.291	0.003
M16	1.223	0.016	1.012	0.439
M32	1.103	0.145	0.983	0.557
M64	1.186	0.034	1.049	0.300

Table 7.12: χ^2 goodness-of-fit test results on central 68% of PLRV histogram (bin size 1) for all entropy sources.

(a) Baseline entropy sources

Source	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
REF	1.164	0.084	1.128	0.136
CSPRNG	1.320	0.006	1.318	0.006
MIS	1.174	0.074	1.149	0.105

(b) CLEAR

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	13.857	0.000	14.153	0.000
M8	3.679	0.000	3.804	0.000
M16	1.581	0.000	1.541	0.000
M32	1.276	0.014	1.261	0.019
M64	1.249	0.024	1.185	0.067

(c) SET

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	20.308	0.000	17.786	0.000
M8	7.965	0.000	7.389	0.000
M16	3.867	0.000	3.700	0.000
M32	2.030	0.000	1.936	0.000
M64	1.531	0.000	1.495	0.000

(d) FLIP

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	18.704	0.000	16.126	0.000
M8	3.951	0.000	3.713	0.000
M16	1.991	0.000	1.951	0.000
M32	1.344	0.003	1.315	0.006
M64	1.329	0.005	1.319	0.006

(e) ALTERNATE

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	14.514	0.000	13.159	0.000
M8	2.946	0.000	2.780	0.000
M16	1.429	0.001	1.406	0.001
M32	1.395	0.001	1.347	0.003
M64	1.445	0.000	1.387	0.001

(f) CORRELATION

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	24.241	0.000	26.956	0.000
M8	1.587	0.000	1.564	0.000
M16	1.094	0.211	1.108	0.179
M32	1.162	0.087	1.141	0.117
M64	1.189	0.061	1.157	0.096

Manipulated Entropy Sources

Table 7.11 and 7.12 show a very strong sensitivity to with manipulation strength.

Strong manipulations (`M4`, `M8`) cause severe rejections, for example χ^2_{red} reaches extremely high values at `M4`, with effectively zero p-values across configurations, including the log-space. `CORRELATION` manipulation in particular shows $\chi^2_{\text{red}} \approx 97$ in the central 68% region. `M8` values fall, yet remain very high, confirming inconsistency with the theoretical model.

At `M16`, bit-based manipulations continue showing clear rejections (χ^2_{red} between 1.5–4), while `ALTERNATE` is weaker but similar. `CORRELATION` manipulations at `M16` show milder deviations: linear space reports $\chi^2_{\text{red}} = 1.22$, $p = 0.016$, but log-space and bin size 5 analyses yield non-significant results ($p > 0.4$), indicating low-frequency correlation patterns produce PLRV distributions that remain compatible with the theoretical Laplace model despite degraded entropy.

Mild manipulations (`M32`, `M64`) present variable sensitivity. `M32` bit-based manipulations reject the null in linear space ($p < 10^{-4}$), even though log-space is more permissive. With bin size 5, some cases become marginal or compatible, especially when considering the central part of the histogram only. `M32 CORRELATION` is mild and treated as "good" entropy by the test. At `M64`, most manipulations are indistinguishable from baseline, especially with bin size 5 and log-space, where p-values often exceed 0.1, demonstrating PLRV shapes consistent with ε-DP guarantees.

Overall, the χ^2 test in log-space balances avoiding false rejections from tail noise in baselines and strong detection of manipulations. Most notably, bit-biasing manipulations distort the PLRV distribution more than low-frequency correlations.

The interpretation of statistically detectable deviations and their relationship to concrete privacy loss is discussed in Section 7.4.

7.4 Discussion

Taken together, the sign test, sigma exceedance test, and χ^2 goodness-of-fit test provide a comprehensive framework for evaluating entropy quality and PLRV adherence to the ε-DP model.

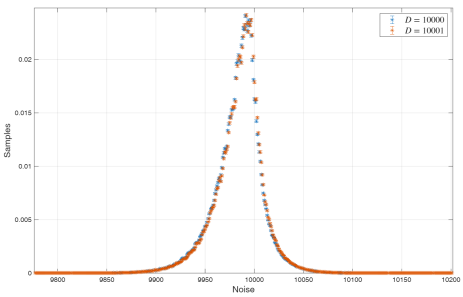
The sign test effectively detects directional biases in privacy loss and is highly sensitive to strong manipulations (`M4`, `M8`). However, it lacks sensitivity to amplitude changes and can miss subtler effects such as alternating or balanced perturbations seen at higher manipulation levels.

The sigma exceedance test complements the sign test by capturing both the frequency and magnitude of exceedances beyond expected noise levels. It identifies strong and moderate manipulations, detects tail anomalies at `M16`, but begins to lose sensitivity near `M32`. Comparing linear and log-space analysis, the log-space generally better matches baseline behavior.

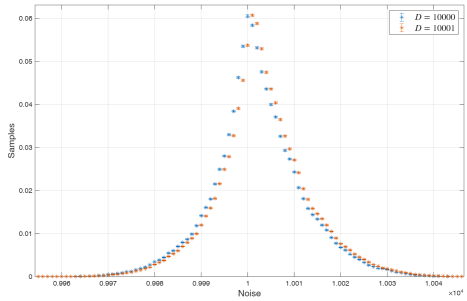
The χ^2 test assesses how well the data matches the shape of the theoretical distribution, strongly rejecting the Laplace model under strong and moderate manipulations with overwhelming significance. At `M16`, bit-bias manipulations remain clearly detectable, while correlation-based perturbations often appear compatible with the model, especially in log-space or coarser binnings. Mild manipulations become indistinguishable from baseline under these conditions.

In practice, these tests should be interpreted together. Bin size, scale choice (linear or log), and larger errors in the tails influence test sensitivity. Bit-level biases more strongly distort the bit-stream and are easier to detect, while temporal correlations affect sequence structure more subtly.

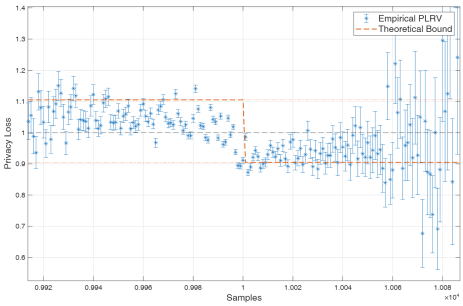
While the presented metrics rigorously assess adherence to the theoretical model, statistical deviations do not, by themselves, imply that the privacy guarantee is violated, as we anticipated in Section 7.1.5. In particular, these tests quantify the minimum bias required for statistically detectable deviations from the ideal Laplacian mechanism, but do not imply that ev-



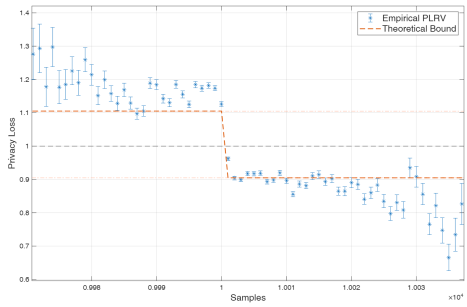
(a) Distribution for `M4 SET` manipulation.



(b) Distribution for `M4 CLEAR` manipulation.



(c) PLRV values for `M4 SET` manipulation.



(d) PLRV values for `M4 CLEAR` manipulation.

Figure 7.5: Output distributions and PLRV values for the `M4 SET` and `CLEAR` manipulations, with $\varepsilon = 0.1$, $N = 10^6$, bin size $= 1$. (a) and (b) show the empirical distributions for `SET` and `CLEAR`, both diverging from the theoretical Laplace. (c) and (d) present the PLRV values: `CLEAR` shows clear violations of privacy bounds, while `SET`, despite failing the tests, maintains PLRV values within bounds.

ery such deviation corresponds to an effective violation of the differential privacy bounds. This distinction is evident when comparing the effects of `M4 CLEAR` and `SET` manipulations in Figure 7.5b and Figure 7.5a. Both manipulations diverge significantly from the Laplace output distribution and are decisively rejected by the tests, yet the underlying privacy guarantee seems to differ.

As shown in Figure 7.5d, `CLEAR` produces PLRV values that violate the

expected privacy bounds, indicating genuine loss of privacy. In contrast, Figure 7.5c demonstrates that the `SET` manipulation, though non-Laplacian in distribution, yields PLRV values within the limits. These examples illustrate that a detected anomaly in the output distribution demands further scrutiny, as actual privacy failure depends on whether the empirical PLRV surpasses the theoretical threshold.

These findings reveal an important asymmetry in how different types of entropy degradation affect differential privacy mechanisms. DP appears much more resilient to correlations introduced by mis-calibrations (`MIS`) and similarly subtle manipulations such as low-frequency correlations (`CORRELATION`) and symmetric perturbations (`ALTERNATE`), which often remain undetected or produce PLRV values within acceptable bounds even when the output distribution deviates from the ideal Laplace shape. In contrast, bit-level biases (`CLEAR`, `SET`, `FLIP`) distort the sampling process more directly, producing distributions that not only fail statistical tests but can also violate the privacy guarantee itself.

Our experiments demonstrate that entropy quality does measurably affect differential privacy mechanisms. The statistical framework we developed is able to reliably detect deviations when approximately 1 bit in every 8 to 16 is manipulated; beyond this threshold, detection becomes increasingly ambiguous, particularly for correlation-based perturbations. Strong manipulations, however, alter the Laplace distribution to the point where the output no longer resembles the theoretical model.

From a practical point of view, these results suggest that while differential privacy mechanisms are robust enough to imperfect randomness, users who require strict adherence to the mathematical bounds should ensure their entropy sources are of high quality. The gap between statistical detectability and actual privacy failure provides some margin of safety, but this margin shrinks rapidly as manipulation intensity increases. For applications where the privacy budget must be spent precisely as specified,

it is advisable to invest in high quality entropy sources.

CHAPTER 8

OpenDP Artifact Analysis and Resolution

As discussed in Section 7.2.1, we observed systematic artifacts when reconstructing histograms from samples generated by iteratively calling OpenDP's discrete Laplace mechanism (Figure 7.1a). These artifacts exhibit a clear symmetry with respect to the distribution center and a periodic pattern whose spacing depends on the Laplace scale. They appear regardless of which unbiased entropy source feeds the library, including OpenDP's native source, which strongly suggests that the issue lies in the internal sampling logic rather than in the external randomness.

While the global Laplace shape is preserved, the artifacts are clearly visible. A single DP query might still satisfy the claimed (ε, δ) guarantees, but the presence of structured deviations becomes problematic for iterated queries and for all higher-level mechanisms that build on the discrete Laplace sampler, such as the discrete Gaussian mechanism.

8.1 Implementation Structure

According to the documentation [77], the implementation follows the construction described by Canonne, Kamath, and Steinke [65]. The paper describes a family of exact samplers and provides proofs for each component used to generate discrete Laplace (and discrete Gaussian) samples.

In the OpenDP codebase, this logic is decomposed into a set of nested primitives that call each other in a tree-like fashion:

- `sample_discrete_laplace`: top-level sampler for Lap$\sim(0, \texttt{scale})$, which draws a random sign via `sample_standard_bernoulli` and a magnitude via `sample_geometric_exp_fast`; "fast" geometric sampler with parameter $\exp(-x)$, which repeatedly calls `sample_uniform_ubig_below` and `sample_bernoulli_exp`;
- `sample_geometric_exp_slow`: reference geometric sampler defined as a sequence of independent Bernoulli$(\exp(-x))$ trials;
- `sample_standard_bernoulli`: Bernoulli(0.5) implemented by extracting a single random bit;
- `sample_bernoulli_exp`: Bernoulli$(\exp(-x))$ for general $x \geq 0$, implemented by repeatedly invoking `sample_bernoulli_exp1` on the integer and fractional parts of x;
- `sample_bernoulli_exp1`: Bernoulli$(\exp(-x))$ restricted to $x \in [0, 1]$, based on an alternating Taylor series;
- `sample_bernoulli_rational`: Bernoulli(p) for rational $p = a/b$;
- `sample_uniform_ubig_below`: rejection sampler for a uniform big integer on $\{0, \ldots, \texttt{upper} - 1\}$ based on repeated calls to the underlying byte-level RNG.

8.2 Diagnostic Methodology

To localize the source of the artifacts, we adopted a top-down approach. We implemented a traced version of OpenDP's CKS20 sampler that, for each call to the discrete Laplace mechanism, records one representative output from every primitive in the call chain. Concretely, a single call to

the traced `sample_discrete_laplace` returns both the final discrete Laplace sample and a diagnostic row containing: the first uniform value drawn at the beginning of the cascade; the first value produced by `sample_uniform_ubig_below`; the first outcomes of `sample_bernoulli_standard`, `sample_bernoulli_rational`, `sample_bernoulli_exp1`, and `sample_bernoulli_exp`; the first magnitudes returned by `sample_geometric_exp_slow` and `sample_geometric_exp_fast`; and finally the discrete Laplace sample itself.

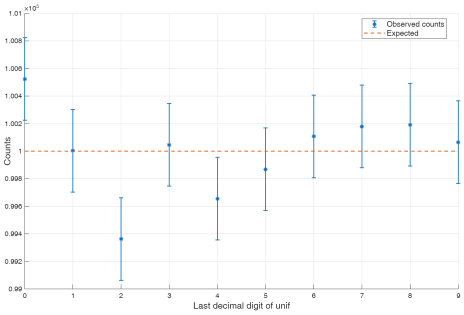
By iterating $N = 10^6$ times, we saved these cascades of samples, all generated from the same underlying entropy source. On these empirical distributions we then performed goodness-of-fit tests against the corresponding theoretical distributions: χ^2 tests and z-tests. Because all columns are generated from the same random bytes, any deviation must originate from one of the deterministic transformations. The idea behind this top-down structure is that as soon as one primitive starts to distort the distribution, the discrepancy becomes visible in its own column (and in all downstream columns), allowing us to pinpoint the earliest possibly faulty step in the sampling pipeline.

As our analysis so far has focused on $\varepsilon = 0.1$, which translates to a Laplace scale of $b = 10$, results reported in this section use the same parameter choice.

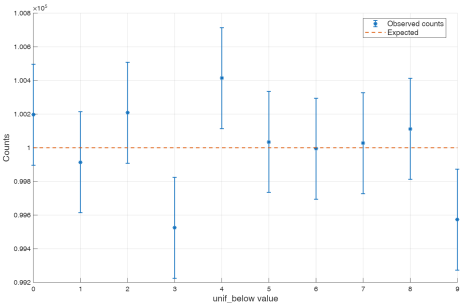
8.3 Layer-by-Layer Analysis

8.3.1 Uniform Sampling

While already confident the bit-stream would be uniform (as established in Chapter 3), we started the analysis by testing uniformity on the sampled bits, followed by testing for uniformity on the rejection sampler function `sample_uniform_ubig_below`. Results are presented in Figure 8.1. Both distributions yield $\chi^2_{\text{red}} \approx 1$ and p-values of $p = 0.433$ (uniform bit sam-



(a) Uniformity fit on the last digit of sampled values.



(b) `sample_uniform_ubig_below` uniformity fit with upper bound 10.

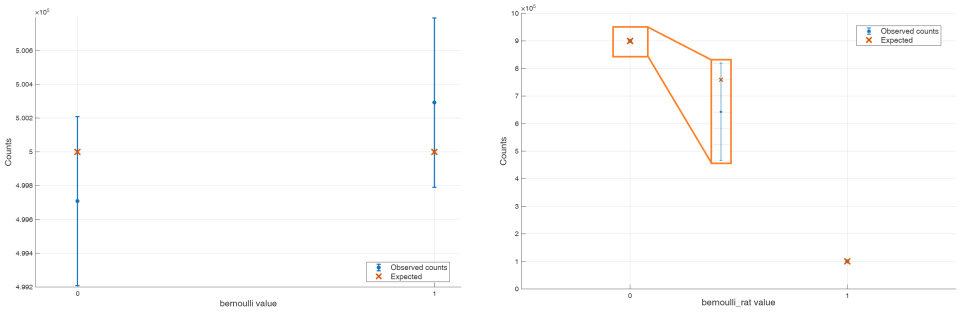
Figure 8.1: Uniformity tests on base sampling primitives. Left: last digit distribution of sampled uniform values ($\chi^2 = 9.04$, dof = 9, $p = 0.433$). Right: `sample_uniform_ubig_below` with upper bound 10 ($\chi^2 = 6.83$, dof = 9, $p = 0.654$). Both tests show $\chi^2_{\text{red}} \approx 1$ and $p > 0.4$, confirming the entropy source operates correctly.

pling) and $p = 0.654$ (thresholded uniform sampler) when fit against the theoretical uniform model. These results confirm that the entropy source and the uniform integer sampling layer operate correctly.

8.3.2 Bernoulli Samplers

The next layer consists of the Bernoulli samplers, namely `sample_standard_bernoulli`, `sample_bernoulli_exp`, `sample_bernoulli_exp1`, and `sample_bernoulli_rational`. While the standard Bernoulli is called only by the Laplace sampler to determine the sign and draws directly from the uniform entropy source, the other three call each other sequentially in a nested environment, with only the rational sampler harvesting bytes from the entropy source.

To test conformity to the theoretical models, we performed both a χ^2 goodness-of-fit test and a z-test on the empirical success probability $\hat{p} = k_1/n$ against the theoretical value p_{th}, using the normal approximation to



(a) Standard Bernoulli: $z = 0.58$, $p = 0.56$. **(b)** Rational Bernoulli: $z = 0.66$, $p = 0.51$.

Figure 8.2: Bernoulli samplers that draw directly from the entropy source. Left: standard Bernoulli with $p = 0.5$ ($\hat{p} = 0.5003$, $p_{\text{th}} = 0.5000$, $\chi^2 = 0.339$, $p = 0.561$). Right: rational Bernoulli with $p = 0.1$ ($\hat{p} = 0.1002$, $p_{\text{th}} = 0.1000$, $\chi^2 = 0.431$, $p = 0.511$). Both pass goodness-of-fit tests with $\chi^2_{\text{red}} < 1$ and $p > 0.5$. The rational Bernoulli subplot includes an inset showing the zoomed region around the expected value, with error bars representing 1σ confidence intervals, demonstrating the close agreement between observed and theoretical counts.

the binomial. The test statistic

$$z = \frac{\hat{p} - p_{\text{th}}}{\sqrt{p_{\text{th}}(1 - p_{\text{th}})/n}}$$

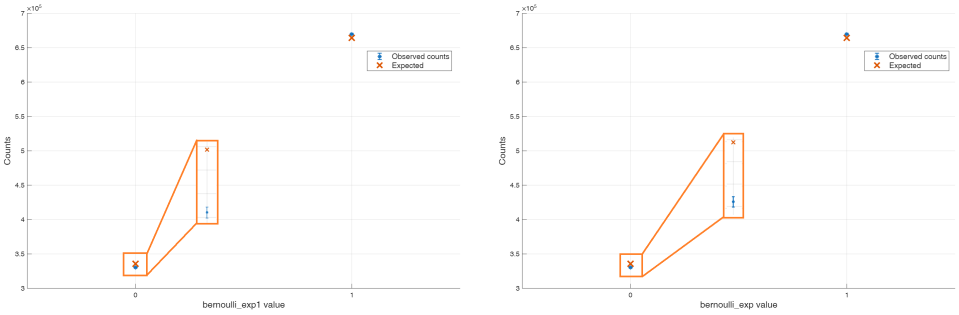
is compared to a standard normal distribution to obtain the p-value.

The standard Bernoulli was tested against $p = 0.5$. The rational Bernoulli, according to the documentation, was tested against $p = 1/b$ where b is the Laplace scale, so $p = \varepsilon = 0.1$. Both exponential functions were tested against the expected probability

$$p = \mathbb{E}[e^{-U/b}] = \frac{1}{b}\sum_{u=0}^{b-1} e^{-u/b} = \frac{1}{b} \cdot \frac{1 - e^{-1}}{1 - e^{-1/b}},$$

where U is uniformly sampled from $\{0, \ldots, b-1\}$.

The results reveal a clear division. The `sample_standard_bernoulli` and `sample_bernoulli_rational` functions pass both tests. The

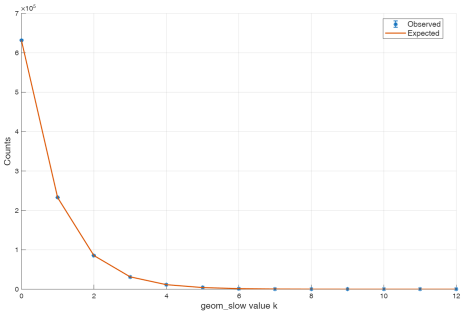


(a) `bernoulli_exp1`: $z = 11.3$, $p \approx 10^{-29}$. **(b)** `bernoulli_exp`: $z = 11.3$, $p \approx 10^{-29}$.

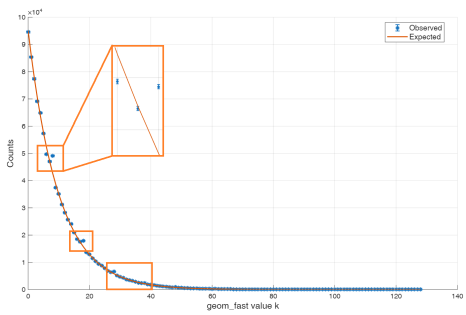
Figure 8.3: Exponential Bernoulli samplers showing significant deviations from the theoretical model. Left: `bernoulli_exp` ($\hat{p} = 0.6696$, $p_{\text{th}} = 0.6643$, $\chi^2 = 128$, $p = 0$). Right: `bernoulli_exp1` ($\hat{p} = 0.6696$, $p_{\text{th}} = 0.6643$, $\chi^2 = 128$, $p = 0$). Both fail goodness-of-fit tests with $\chi^2 \approx 128$ and $p \approx 0$. The insets on the plots show a zoomed region around the expected value, with error bars representing 1σ confidence intervals, showing how the observed values are far beyond 3σ from the expected ones.

standard Bernoulli returns $z = 0.58$ with $p = 0.56$, and the rational Bernoulli returns $z = 0.66$ with $p = 0.51$. In both cases, the z-value sits well within 1σ of the expected value, with correspondingly high p-values. These results are shown in Figure 8.2.

The exponential functions, on the other hand, fail the tests. The `bernoulli_exp1` function returns $z = 11.30$ with $p \approx 0$ and $\chi^2 = 127.7$. The `bernoulli_exp` function shows identical statistics, which is expected given the calling hierarchy. Since the order of function calls runs from `bernoulli_exp` to `bernoulli_exp1` to bernoulli_rational (the most nested), the failing function is most likely `bernoulli_exp1`, with the error propagating upward to `bernoulli_exp`. These results are shown in Figure 8.3.



(a) Geometric slow: passes with good fit to the theoretical model.



(b) Geometric fast: fails with visible periodic artifacts.

Figure 8.4: Geometric sampler distributions. Left: slow implementation (mean $z = 0.96$, $p = 0.34$; $\chi^2 = 15$, $p = 0.26$) passes with good fit to the theoretical model. Right: fast implementation (mean $z = 3.99$, $p = 6.6 \times 10^{-5}$; $\chi^2 = 1.9 \times 10^3$, $p = 0$) fails with visible periodic artifacts similar to those observed in the final Laplace distribution. The insets on the plot show a zoomed region around the expected value, with error bars representing 1σ confidence intervals, showing how the observed values are far beyond 3σ from the expected ones.

8.3.3 Geometric Samplers

Moving to the geometric samplers layer, we immediately notice a divergence. The geometric slow distribution reports seemingly good test results, while the geometric fast sampler clearly exhibits artifacts similar to those seen in the final Laplacian output, and it fails the statistical tests.

We performed both z-tests and χ^2 tests against the geometric distribution

$$P(K = k) = (1 - e^{-x})\, e^{-xk}, \quad k = 0, 1, 2, \ldots$$

where $x = 1$ for `sample_geometric_exp_slow` and $x = 1/b$ for `sample_geometric_exp_fast`.

The reason we assume `sample_geometric_exp_slow` still appears correct, despite depending on the buggy `bernoulli_exp` is because this sampler is always called with parameter $x = 1$, which means it only ever

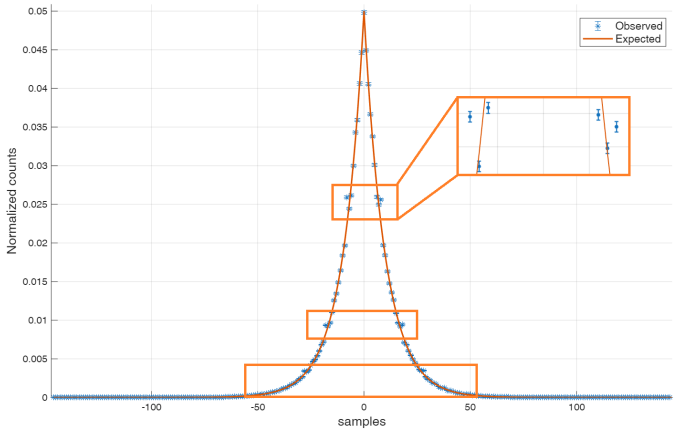


Figure 8.5: Discrete Laplace distribution generated by OpenDP's sampler (mean $z = -0.68$, $p = 0.49$; $\chi^2 = 2.1 \times 10^3$, $p = 0$), showing the characteristic periodic artifacts that closely match those in Figure 7.1a. While the mean is well-centered, the distribution fails goodness-of-fit testing due to systematic deviations. The insets on the plot show a zoomed region around the expected value, with error bars representing 1σ confidence intervals, showing how the observed values are far beyond 3σ from the expected ones.

uses a single Bernoulli with $p = e^{-1}$, implemented via `sample_bernoulli_exp1(1)`. As the parameter e^{-1} is sufficiently large, only a small bias is introduced. The function returns quickly enough that this small bias becomes negligible, producing a geometric distribution that passes the tests.

The fast geometric sampler, in contrast, repeatedly samples and uses an entire family of slightly biased coins with probabilities $e^{-u/b}$. This over-selects some samples while underselecting others, and distorts the entire geometric distribution, causing the tests to fail. The distributions are shown in Figure 8.4.

8.3.4 Final Laplace Distribution

Finally, the generated discrete Laplace distribution, shown in Figure 8.5, closely resembles the artifacts we first observed in Figure 7.1a. The top-

down analysis has successfully traced the bug from its visible manifestation in the final output back to its origin in the `bernoulli_exp1` function.

8.4 Bernoulli Exp1 Algorithm Analysis

The algorithm for `sample_bernoulli_exp1` presented in CKS20 [65] is mathematically correct. If every intermediate coin flip $\mathrm{Bernoulli}(x/k)$ behaved ideally, the procedure would return True with exactly probability $\exp(-x)$. The problem arises on finite computers. Our interpretation is that each of these coin flips is implemented using a separate finite-precision random comparison, which introduces a tiny deviation from the ideal probability that depends on k. Individually, these errors are negligible. However, the algorithm operates as a "repeat until failure" loop that composes many such coins together, and the small discrepancies accumulate across iterations. The net effect is a global success probability that differs measurably from $\exp(-x)$. When this biased sampler feeds into the fast geometric and discrete Laplace constructions, the error is amplified and manifests as the artifacts we observed in the output distributions.

8.5 Proposed Algorithm

Our alternative implementation is based on the proof of Proposition 33 in CKS20 [65]. The algorithm samples from a $\mathrm{Bernoulli}(\exp(-x))$ distribution by iteratively refining the partial sums of the Taylor series expansion until a decision can be made, without ever computing the exponential directly. The pseudocode is reported in Algorithm 1, and it is based on an alternating series Bernoulli sampler that compares a uniform random variate against successive brackets for e^{-x} defined by consecutive partial sums, as described in [86].

The first step is to sample a 128-bit uniformly random integer $U \in \{0, \ldots, 2^{128} - 1\}$ and interpret it as the rational $u = U/2^{128}$. In our imple-

Algorithm 1: Sample Bernoulli$(\exp(-x))$ via Taylor Series

Input: $x \in [0, 1]$

Output: True with probability $\exp(-x)$, False otherwise

if $x = 0$ **then**
  return *True*
Draw $U \sim$ Uniform$(\{0, 1, \ldots, 2^{128} - 1\})$
$u \leftarrow U/2^{128}$
$S_{\text{prev}} \leftarrow 1$
$S_{\text{curr}} \leftarrow 1 - x$
$k \leftarrow 1$
$t \leftarrow x$
while *True* **do**
  $L \leftarrow \min(S_{\text{prev}}, S_{\text{curr}})$ $H \leftarrow \max(S_{\text{prev}}, S_{\text{curr}})$
  if $u < L$ **then**
    return *True*
  if $u \geq H$ **then**
    return *False*
  $k \leftarrow k + 1$
  $t \leftarrow t \cdot x/k$
  $S_{\text{prev}} \leftarrow S_{\text{curr}}$
  if k *is even* **then**
    $S_{\text{curr}} \leftarrow S_{\text{curr}} + t$
  else
    $S_{\text{curr}} \leftarrow S_{\text{curr}} - t$

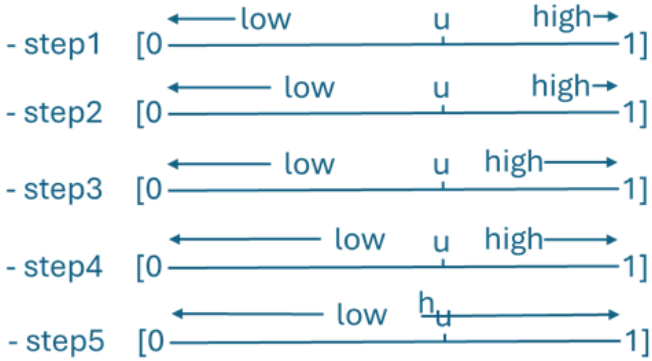


Figure 8.6: Iterative refinement of the Taylor series bounds. At each step, the interval $[L, H]$ tightens around the true value. The algorithm terminates when the uniform sample u falls outside the current bracket, returning True if $u < L$ or False if $u \geq H$.

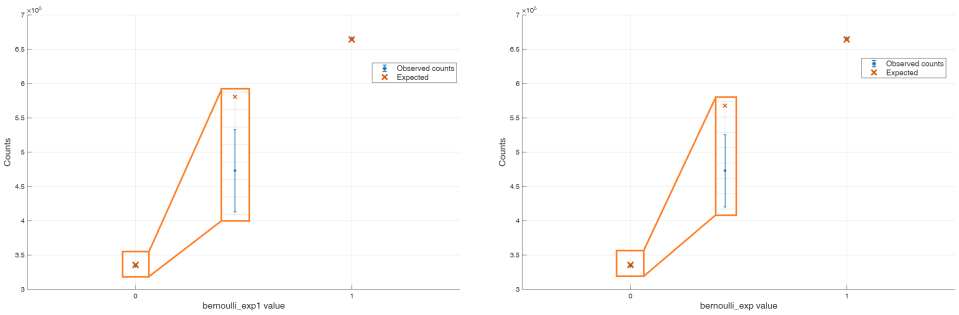
mentation, this fraction is stored as a pair of arbitrary-precision integers (numerator and denominator) using Rust's `dashu::RBig` type [87]. This representation avoids floating-point arithmetic entirely, ensuring that all subsequent comparisons are performed exactly.

The algorithm then exploits the Taylor series expansion of the exponential:

$$e^{-x} = \sum_{k=0}^{\infty} \frac{(-x)^k}{k!} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots$$

Because this is an alternating series with decreasing terms, the partial sums oscillate around the true value e^{-x}. This means that consecutive partial sums S_{k-1} and S_k always bracket the true value: one lies above e^{-x} and the other below.

The algorithm initializes $S_{\text{prev}} = 1$ and $S_{\text{curr}} = 1 - x$, corresponding to the first two partial sums. At each iteration, it computes bounds $low = \min(S_{\text{prev}}, S_{\text{curr}})$ and $H = \max(S_{\text{prev}}, S_{\text{curr}})$, then compares the uniform sample u against these thresholds. If $u < L$, we know with certainty that $u < e^{-x}$, so the algorithm returns True. If $u \geq H$, we know $u \geq e^{-x}$, so



(a) `bernoulli_exp1`: $z = 1.80$, $p = 0.073$. **(b)** `bernoulli_exp`: $z = 1.80$, $p = 0.073$.

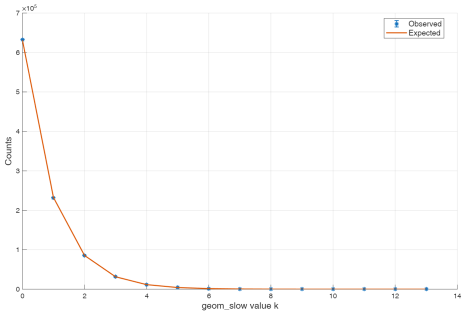
Figure 8.7: Exponential Bernoulli samplers with the new implementation. Left: `bernoulli_exp` ($\hat{p} = 0.6651$, $p_{\text{th}} = 0.6643$, $\chi^2 = 3.22$, $p = 0.0726$). Right: `bernoulli_exp1` ($\hat{p} = 0.6651$, $p_{\text{th}} = 0.6643$, $\chi^2 = 3.22$, $p = 0.0726$). Both now pass goodness-of-fit tests with $\chi^2_{\text{red}} \approx 3$ and $p > 0.07$, demonstrating correct sampling behavior. The insets on the plots show a zoomed region around the expected value, with error bars representing 1σ confidence intervals, showing how the observed values are within 3σ from the expected values.

it returns False. Otherwise, u falls within the undecided region, and the algorithm computes the next term of the Taylor series until convergence.

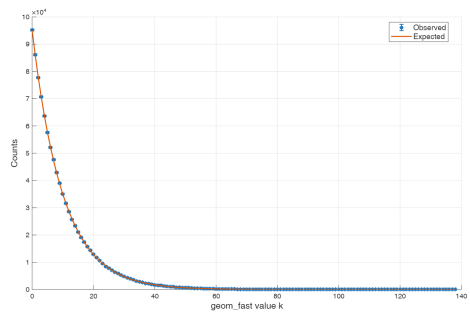
Figure 8.6 illustrates this process. At each step, the bounds L and H converge toward e^{-x} until u falls outside the shrinking interval, at which point a decision is made.

8.6 Results with new Implementation

The proposed algorithm and its evaluation focus only on the statistical properties of the distributions output. In this analysis, we did not benchmark the runtime performance or compare it with the original implementation. After replacing the original `bernoulli_exp1` function with our Taylor series implementation, we repeated the same diagnostic pipeline to verify that the artifacts had been resolved. The results confirm that all samples now correctly fit their theoretical distributions.



(a) Geometric slow: $\chi^2 = 9.74$, $p = 0.715$.



(b) Geometric fast: $\chi^2 = 145.5$, $p = 0.315$.

Figure 8.8: Geometric sampler distributions with the corrected implementation. Left: slow implementation (mean $z = -0.38$, $p = 0.71$; $\chi^2 = 9.7$, $p = 0.72$). Right: fast implementation (mean $z = -0.18$, $p = 0.86$; $\chi^2 = 1.5 \times 10^2$, $p = 0.31$). Both samplers now pass goodness-of-fit tests, with the fast sampler no longer exhibiting the periodic artifacts observed previously.

Starting with the Bernoulli samplers, Figure 8.7 shows that both `bernoulli_exp1` and `bernoulli_exp` now produce samples consistent with the theoretical model. From $n = 10^6$ samples, we observe an empirical success probability of $\hat{p} = 0.6651$ against a theoretical value of $p_{\text{th}} = 0.6643$. The z-test yields $z = 1.80$ with $p = 0.073$, and the χ^2 test gives $\chi^2 = 3.22$ with $p = 0.073$. While the p-values are somewhat low, the z-score remains well within 2σ, indicating acceptable statistical fluctuation rather than systematic bias. This stands in stark contrast to the original implementation, which produced $z \approx 11$ and $p \approx 0$. Moreover, results were confirmed by repeatedly iterating the procedure through independent experiments and with different parameter choice.

The improvement propagates through the sampling hierarchy. Figure 8.8 shows the geometric samplers, both of which now pass the goodness-of-fit tests. The slow geometric sampler, which already worked correctly, continues to perform well ($\chi^2 = 9.74$ with 13 degrees of freedom, $p = 0.715$). The fast geometric sampler now adheres to the theoretical distribution

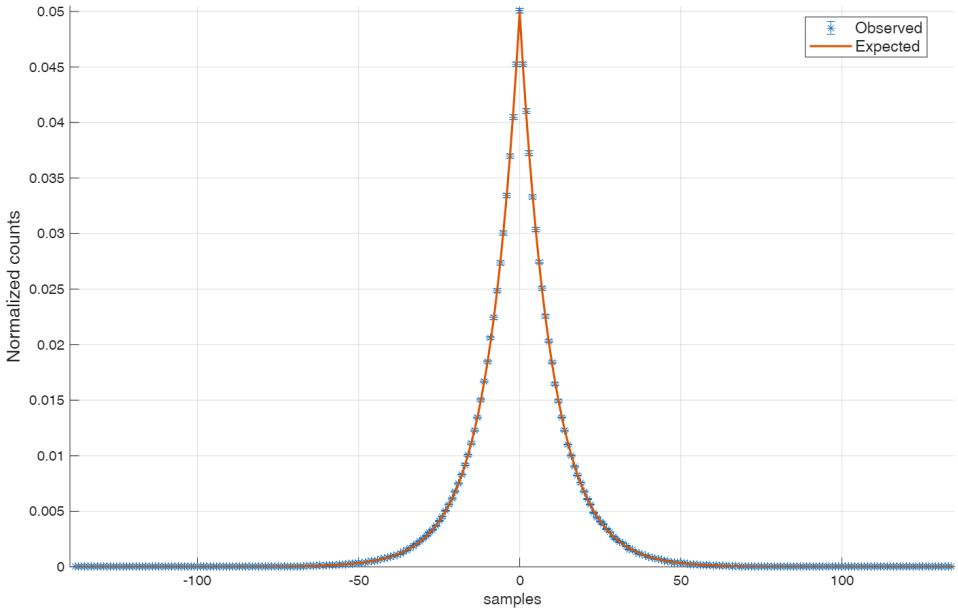


Figure 8.9: Discrete Laplace distribution with the corrected implementation (mean $z = -1.22$, $p = 0.22$; $\chi^2 = 2.8 \times 10^2$, $p = 0.5$). The periodic artifacts visible in Figure 8.5 have been eliminated, and the distribution now passes goodness-of-fit testing with $\chi^2_{\text{red}} \approx 1$ and $p = 0.5$.

($\chi^2 = 145.5$ with 138 degrees of freedom, $p = 0.315$), and empirical moments closely matching their theoretical values: $\hat{\mu} = 9.507$ against $\mu_{\text{th}} = 9.508$, and $\hat{\sigma}^2 = 99.94$ against $\sigma^2_{\text{th}} = 99.92$. The periodic artifacts are no longer visible.

Finally, Figure 8.9 shows the discrete Laplace distribution produced by the corrected sampler. The characteristic periodic artifacts that motivated this investigation (compare with Figure 8.5) have been completely eliminated. The distribution passes the χ^2 goodness-of-fit test with $\chi^2 = 275.4$ on 276 degrees of freedom ($p = 0.499$), indicating excellent agreement with the theoretical model. The empirical variance $\hat{\sigma}^2 = 199.90$ matches the theoretical value $\sigma^2_{\text{th}} = 199.83$, and the sample mean $\hat{\mu} = -0.017$ is statistically indistinguishable from zero ($z = -1.22$, $p = 0.224$).

These results confirm that the artifacts originated from the `bernoulli`

`_exp1` implementation and that our proposed fix seemingly resolves the issue. The corrected sampler produces discrete Laplace samples that are statistically indistinguishable from the theoretical distribution.

Conclusions

This thesis analyzes how the quality of random numbers affects differential privacy. By running experiments and measuring real impacts, it shows how degrading randomness affects the privacy guarantees that these algorithms actually deliver.

We began by developing and characterizing a quantum random number generator, covering the principles behind the generation of randomness, the hardware implementation, and the software framework needed to harvest the bit-stream. To qualify the entropy source, we established an offline testing framework based on well-known statistical test suites (NIST SP 800-22, TestU01) and implemented online health tests capable of rapidly detecting catastrophic generation failures. These online tests monitor the Monobit and RUNS statistics, flagging anomalies when the observed distribution shifts away from its expected mean. We also demonstrated an online min-entropy estimation method based on the inter-failure sequence of the Repetition Count Test and Adaptive Proportion Test.

We then focused towards an exemplary application of high-quality randomness: differential privacy. The motivation for this investigation came from two seemingly contradictory positions in the literature. On one hand,

the U.S. Census Bureau expressed concerns about using pseudorandom number generators, suggesting that entropy quality matters for privacy guarantees. On the other hand, theoretical work indicates that differential privacy can tolerate certain amounts of bias without compromising its guarantees. We set out to measure where, in practice, this boundary lies.

To do so, we developed a systematic experimental framework to measure empirical privacy loss using the definition of differential privacy directly. We tested three entropy conditions: a well-calibrated QRNG as our reference, a CSPRNG for comparison, and deliberately degraded streams including a mis-calibrated QRNG and manually manipulated bit-streams with controlled levels of bias and correlation. For each condition, we reconstructed the Privacy Loss Random Variable from millions of queries and assessed its conformity to the theoretical model using three complementary statistical tests: the sign test for directional bias, the sigma exceedance test for outlier frequency, and the χ^2 goodness-of-fit test for overall adherence to the theoretical model.

Our analysis relied primarily on IBM's DiffPrivLib, though we also examined OpenDP and Google's Differential Privacy library. OpenDP, in particular, exhibited unexpected artifacts in its discrete Laplace sampler that were visible even with high-quality entropy. We dedicated a portion of this work to tracing the source of these artifacts, and proposed an alternative implementation.

Several limitations of this study deserve mention. We focused exclusively on the discrete Laplace mechanism because it provides the tightest privacy guarantees (pure ε-DP) and avoids floating-point complications. Other mechanisms, such as the Gaussian mechanism used for approximate (ε, δ)-DP, involve additional numerical considerations that could make them more susceptible to entropy degradation even under normal conditions. Extending this analysis to other noise distributions remains an interesting direction for future work. Additionally, we tested only counting queries

with sensitivity equal to 1. Queries with higher sensitivity or more complex structure could interact differently with entropy quality.

In conclusion, Differential privacy mechanisms seems to be resilient to subtle forms of entropy degradation. Low-frequency correlations and symmetric perturbations often went undetected or produced privacy loss values that remained within acceptable bounds, even when the output distribution deviated measurably from the ideal Laplace shape. Bit-level manipulations, in contrast, had a more direct impact on the sampling process, distorting the output distribution and, in case of strong manipulations, violating the privacy guarantee itself. Our statistical framework was able to detect deviations when approximately 1 bit in every 8 to 16 was manipulated. Beyond this threshold, detection became increasingly ambiguous and inconclusive. Potentially, detecting `M32` or `M64` manipulations with high confidence would require sample sizes substantially larger than $N = 10^6$, likely by one or two orders of magnitude, making such detection impractical for most real-world monitoring scenarios.

This work highlights that, while randomness quality is of primary concern in cryptography, differential privacy seems to be more permissive. In cryptographic applications, even minor predictability in the random stream can be catastrophic, making high-quality entropy (ideally from quantum sources) essential. Differential privacy, by contrast, appears to tolerate a degree of imperfection before the guarantees fail in practice. This does not mean entropy quality is irrelevant for DP, but rather that the sensitivity threshold is higher than in a cryptographic setting.

Finally, we note that our analysis focused on what the statistical methods we adopted can detect. It remains possible that entropy degradation affects differential privacy in ways that our framework was not able to capture. Notably, while the artificially induced manipulations produced clear effects on privacy loss distributions, the miscalibrated QRNG (`MIS`) did not yield noticeable deviations despite its known hardware issues. This

suggests that real-world entropy degradation may manifest differently from our controlled perturbations, making the development of different or complementary methodologies necessary.

Appendices

APPENDIX A

Pseudo-code Listings

This appendix provides detailed pseudo-code implementations of the statistical tests used in NIST SP 800-90B's IID validation procedure, as described in Section 3.3. These algorithms represent the six categories of tests employed to evaluate whether a noise source produces independently and identically distributed outputs: excursion statistics, directional runs, collision tests, periodicity tests, covariance tests, and compression tests. Each algorithm is presented in a standardized format showing the input sequence, computational steps, and the resulting test statistic that is compared against the distribution obtained from permuted versions of the data. These implementations form the basis of the bootstrap methodology used throughout our entropy quality evaluation.

Algorithm 2: Excursion Test Statistic

Input: A sequence $S = (s_1, s_2, \ldots, s_L)$

Output: The statistic T

Compute the sample mean: $\bar{X} \leftarrow \frac{1}{L}\sum_{j=1}^{L} s_j$;

for $i \leftarrow 1$ **to** L **do**

  $d_i \leftarrow \left|\sum_{j=1}^{i}(s_j - \bar{X})\right|$;

$T \leftarrow \max(d_1, d_2, \ldots, d_L)$;

return T

Algorithm 3: Number of Directional Runs

Input: A sequence $S = (s_1, s_2, \ldots, s_L)$

Output: Test statistic T

Construct a new sequence $S' = (s'_1, s'_2, \ldots, s'_{L-1})$ such that:;

for $i \leftarrow 1$ **to** $L - 1$ **do**

  if $s_i > s_{i+1}$ **then**

    $s'_i \leftarrow -1$;

  else

    $s'_i \leftarrow +1$;

Initialize run counter: $T \leftarrow 1$;

for $i \leftarrow 2$ **to** $L - 1$ **do**

  if $s'_i \neq s'_{i-1}$ **then**

    Increment T;

return T

Algorithm 4: Average Collision Test

Input: A sequence $S = (s_1, s_2, \ldots, s_L)$

Output: Test statistic T

Initialize empty list $C \leftarrow [\,]$;

Set $i \leftarrow 1$;

while $i < L$ **do**

  Find the smallest $j \geq 1$ such that the subsequence $(s_i, s_{i+1}, \ldots, s_{i+j-1})$ contains a repeated value;

  if *no such j exists* **then**

    break;

  Append j to list C;

  Update $i \leftarrow i + j$;

Compute $T \leftarrow \text{average}(C)$;

return T

Algorithm 5: Periodicity Test

Input: A sequence $S = (s_1, s_2, \ldots, s_L)$ and a lag parameter $p < L$

Output: Test statistic T

Initialize $T \leftarrow 0$;

for $i \leftarrow 1$ **to** $L - p$ **do**

  if $s_i = s_{i+p}$ **then**

    Increment T;

return T

Algorithm 6: Compression Test Statistic

Input: A sequence $S = (s_1, s_2, \ldots, s_L)$

Output: Test statistic T

Encode S as a space-separated string of values;

e.g., $S = (144, 21, 139, 0, 0, 15) \rightarrow$ "144 21 139 0 0 15";

Compress the string using the `bzip2` compression algorithm (see [88]);

Let T denote the length of the compressed string in bytes;

return T

Algorithm 7: Permutation-based Procedure to test the IID Assumption

Input: A sequence $S = (s_1, s_2, \ldots, s_L)$

Output: Decision on the IID assumption

foreach *test* k **do**

  Initialize counters: $C_{0,k} \leftarrow 0$, $C_{1,k} \leftarrow 0$;

  Compute test statistic $T_{x,k}$ on S;

for $j \leftarrow 1$ **to** $n_{sequences}$ **do**

  Permute S using the Fisher–Yates shuffle algorithm;

  foreach *test* k **do**

    Compute test statistic T_i on the permuted data;

    if $T_i > T_{x,k}$ **then**

      Increment $C_{0,k}$;

    if $T_i = T_{x,k}$ **then**

      Increment $C_{1,k}$;

foreach *test* k **do**

  if $C_{0,k} + C_{1,k} \leq 0.0005 \cdot n_{sequences}$ ***or*** $C_{0,k} \geq 0.9995 \cdot n_{sequences}$ **then**

    return *Reject the IID assumption*;

return *Assume the noise source outputs are IID*

APPENDIX B

SET CLEAR Results

This appendix presents the complete statistical test results for the SET and CLEAR manipulation types introduced in Section 3.4.3. As discussed in Chapter 3, these manipulations proved to be particularly severe: by deterministically setting or clearing one bit in each symbol based on a deterministic rule, they drastically skew symbol distributions, causing some values to become over-represented while others are suppressed or eliminated entirely. The catastrophic nature of these manipulations resulted in consistent test failures across all batteries, even at the lowest manipulation frequencies.

The tables below document the IID validation outcomes, min-entropy estimates, and statistical test failures for both SET and CLEAR across all manipulation levels (M4 to M64). This data confirms the asymmetric behavior between forcing bits high versus low, and the results serve as a reference for understanding the extreme end of the entropy degradation and highlight the complementary nature of different statistical test suites in detecting various forms of bias.

Dataset	IID Validated (files)		Min-entropy (mean)	
REF	20/20		3.9995	
CSPRNG	20/20		3.9995	
MIS	20/20		3.9954	
	SET	**CLEAR**	**SET**	**CLEAR**
M4	19/20	19/20	2.4149	2.9998
M8	0/20	0/20	2.9127	3.4151
M16	0/20	0/20	3.5739	3.7518
M32	0/20	0/20	3.8403	3.8399
M64	3/20	19/20	3.9229	3.9123

Table B.1: IID validation outcomes and RaP min-entropy estimates when grouping bits into 4-bit symbols for SET and CLEAR manipulations. Baseline datasets (REF, CSPRNG, MIS) retain full IID validation and near-ideal entropy. Under injected bias, SET and CLEAR behave asymmetrically: some strongly biased configurations (e.g., M4) still pass IID, due to the "drugging" of certain symbols as explained in Section 3.4.3, while others only recover IID validation when the manipulation becomes sufficiently sparse (e.g., M64 CLEAR).

Table B.2: Statistical test results across datasets and manipulation types. Top: per-file average failures for NIST SP 800-22 and Rabbit. Bottom: aggregate failures for Crush and Alphabit batteries with set verdicts. Reference datasets (REF, CSPRNG) show minimal failures consistent with statistical fluctuations. MIS shows moderate failures. Manipulated datasets are split into SET and CLEAR variants, illustrating how forcing bits high or low can affect the detection of bias.

Dataset	STS Bad Fails (avg/file)		STS Stat. Fails (avg/file)		Rabbit Fails (avg/file)	
REF	0.00		0.35		0.25	
CSPRNG	0.20		0.30		0.10	
MIS	3.05		0.35		11.40	
	SET	**CLEAR**	**SET**	**CLEAR**	**SET**	**CLEAR**
M4	159.00	150.00	0.00	0.05	35.00	32.45
M8	155.80	148.75	1.70	3.60	33.25	32.40
M16	157.45	149.35	0.65	1.40	32.15	31.05
M32	140.50	133.40	3.70	8.45	31.85	31.50
M64	114.15	89.95	12.35	16.60	30.70	30.20

Dataset	Crush Total (set)		Alphabit Total (set)		Set Verdict	
REF	1		0		Silver	
CSPRNG	3		0		Silver	
MIS	257		12		Discarded	
	SET	**CLEAR**	**SET**	**CLEAR**	**SET**	**CLEAR**
M4	1169	1094	17	17	Discarded	Discarded
M8	1053	963	17	17	Discarded	Discarded
M16	929	868	17	17	Discarded	Discarded
M32	819	766	17	17	Discarded	Discarded
M64	729	675	17	17	Discarded	Discarded

APPENDIX C

Sign Test Results

This appendix presents sign test results for the empirical PLRV distributions obtained from differential privacy experiments described in Chapter 7. As introduced in Section 7.3.1, the sign test provides a non-parametric approach to detecting systematic directional biases in how the empirical distribution deviates from the theoretical model. Unlike tests that depend on the magnitude of deviations, the sign test focuses purely on symmetry.

These results complement the chi-squared goodness-of-fit tests by revealing directional patterns that might not be apparent from overall fit statistics. The systematic comparison across binning strategies and histogram regions demonstrates how analytical choices affect the detection of asymmetric biases. Results are discussed in Section 7.3.2.

Table C.1: Sign test results on full PLRV histogram (bin size 5) for all entropy sources. The larger bin size reduces the number of available bins and partially smooths local fluctuations compared to bin size 1.

(a) Baseline entropy sources

Source	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
REF	48	18	0.1114	0.8854	3.9995
CSPRNG	49	18	0.0854	0.5682	3.9995
MIS	50	21	0.3222	0.6718	3.9954

(b) CLEAR

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	21	18	0.0015	0.3833	2.9998
M8	37	30	0.0002	0.3240	3.4151
M16	46	27	0.3020	0.1038	3.7518
M32	47	25	0.7709	0.5601	3.8399
M64	46	23	1.0000	0.8830	3.9123

(c) SET

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	74	14	0	0.0474	2.4149
M8	52	13	0.0004	0.2116	2.9127
M16	50	14	0.0026	0.3222	3.5739
M32	49	17	0.0444	0.1524	3.8403
M64	48	14	0.0055	0.8854	3.9229

(d) FLIP

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	64	15	0	0.1686	2.9998
M8	48	16	0.0293	0.1934	3.4149
M16	47	14	0.0079	0.3817	3.6780
M32	49	18	0.0854	0.0854	3.9021
M64	47	18	0.1439	0.7709	3.9722

(e) ALTERNATE

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	32	12	0.2153	1.0000	3.4148
M8	45	18	0.2327	0.0725	3.6779
M16	48	23	0.8854	0.8854	3.8301
M32	48	15	0.0133	1.0000	3.9503
M64	47	20	0.3817	1.0000	3.9861

(f) CORRELATION

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	28	26	0	0.3449	3.1926
M8	48	25	0.8854	0.3123	3.7518
M16	48	20	0.3123	0.6655	3.9124
M32	47	21	0.5601	0.7709	3.9719
M64	48	20	0.3123	0.8854	3.9913

Table C.2: Sign test results on central 68% of PLRV histogram (bin size 5) for all entropy sources. The combination of larger bins and central region focus further reduces sensitivity to manipulations.

(a) Baseline entropy sources

Source	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
REF	33	12	0.1628	1.0000	3.9995
CSPRNG	33	12	0.1628	0.4869	3.9995
MIS	34	14	0.3915	0.6076	3.9954

(b) CLEAR

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	14	12	0.0129	0.4240	2.9998
M8	25	20	0.0041	1.0000	3.4151
M16	31	19	0.2810	0.7201	3.7518
M32	32	16	1.0000	0.5966	3.8399
M64	31	15	1.0000	1.0000	3.9123

(c) SET

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	50	6	0	0.1189	2.4149
M8	35	8	0.0019	1.0000	2.9127
M16	34	8	0.0029	1.0000	3.5739
M32	33	11	0.0801	0.2962	3.8403
M64	33	9	0.0135	0.7283	3.9229

(d) FLIP

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	44	8	0	0.2912	2.9998
M8	33	9	0.0135	0.2962	3.4149
M16	32	8	0.0070	1.0000	3.6780
M32	33	13	0.2962	0.2962	3.9021
M64	32	12	0.2153	1.0000	3.9722

(e) ALTERNATE

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	22	6	0.0525	1.0000	3.4148
M8	31	12	0.2810	0.4731	3.6779
M16	33	15	0.7283	1.0000	3.8301
M32	33	10	0.0351	1.0000	3.9503
M64	32	13	0.3771	0.8601	3.9861

(f) CORRELATION

Level	n	k_{pos}	p	$p_{\log}$	$H_{\min}$
M4	19	18	7.629e-05	1.0000	3.1926
M8	33	18	0.7283	0.2962	3.7518
M16	33	14	0.4869	1.0000	3.9124
M32	32	10	0.0501	1.0000	3.9719
M64	33	12	0.1628	1.0000	3.9913

APPENDIX D

Sigma Exceedance Test Results

The tables below show test results under different analysis configurations: central 68% versus full histogram, and bin size 1 versus bin size 5. For each entropy source we report exceedance rates for both 1σ and 2σ thresholds. For each threshold, we provide p-values testing whether the observed exceedance rate matches the expected rate from a normal distribution, computed in both linear space (p) and log-space ($p_{\log}$).

These results complement the sign tests and chi-squared tests by focusing specifically on the magnitude of deviations rather than their symmetry or overall distribution shape. Results are discussed in Section 7.3.2.

Table D.1: $k\sigma$ exceedance test results on central 68% of PLRV histogram (bin size 1) for baseline entropy sources. This analysis excludes the noisiest tail bins to focus on the central region.

Source	**1σ exceedance**			**2σ exceedance**		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`REF`	0.331	0.327	0.394	0.070	0.060	0.114
`CSPRNG`	0.352	0.164	0.210	0.076	0.034	0.016
`MIS`	0.329	0.346	0.483	0.062	0.130	0.347

Table D.2: $k\sigma$ exceedance test results on central 68% of PLRV histogram (bin size 1) for `CLEAR` manipulations.

Level	1σ exceedance			2σ exceedance		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
M4	0.812	3.84e-18	3.13e-20	0.594	1.42e-38	1.42e-38
M8	0.431	0.004	0.004	0.198	4.55e-10	1.31e-11
M16	0.319	0.444	0.373	0.094	0.004	0.011
M32	0.364	0.101	0.101	0.091	0.006	0.006
M64	0.348	0.194	0.246	0.057	0.194	0.313

Table D.3: $k\sigma$ exceedance test results on central 68% of PLRV histogram (bin size 1) for `SET` manipulations.

Level	1σ exceedance			2σ exceedance		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
M4	0.575	4.29e-16	4.64e-17	0.332	2.75e-44	2.75e-44
M8	0.456	8.73e-05	8.73e-05	0.200	3.06e-13	1.67e-12
M16	0.399	0.013	0.013	0.118	7.45e-05	7.45e-05
M32	0.377	0.049	0.049	0.093	0.004	0.044
M64	0.412	0.006	0.004	0.054	0.233	0.233

Table D.4: $k\sigma$ exceedance test results on central 68% of PLRV histogram (bin size 1) for `FLIP` manipulations.

Level	1σ exceedance			2σ exceedance		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
M4	0.505	3.48e-08	3.48e-08	0.258	8.57e-24	4.66e-22
M8	0.372	0.067	0.092	0.131	1.04e-05	3.47e-04
M16	0.324	0.394	0.265	0.085	0.013	0.029
M32	0.338	0.264	0.164	0.054	0.233	0.139
M64	0.379	0.047	0.047	0.048	0.340	0.340

Table D.5: $k\sigma$ exceedance test results on central 68% of PLRV histogram (bin size 1) for `ALTERNATE` manipulations.

Level	**1σ exceedance**			**2σ exceedance**		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.452	0.001	3.71e-04	0.231	6.17e-12	6.17e-12
`M8`	0.372	0.072	0.100	0.051	0.286	0.173
`M16`	0.278	0.823	0.903	0.056	0.211	0.211
`M32`	0.331	0.324	0.264	0.088	0.008	0.008
`M64`	0.375	0.059	0.041	0.056	0.211	0.333

Table D.6: $k\sigma$ exceedance test results on central 68% of PLRV histogram (bin size 1) for `CORRELATION` manipulations.

Level	**1σ exceedance**			**2σ exceedance**		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.741	2.10e-16	1.33e-15	0.471	4.09e-32	7.84e-31
`M8`	0.319	0.441	0.719	0.071	0.058	0.058
`M16`	0.362	0.112	0.194	0.064	0.110	0.194
`M32`	0.336	0.284	0.228	0.048	0.347	0.498
`M64`	0.357	0.136	0.178	0.056	0.205	0.205

Table D.7: $k\sigma$ exceedance test results on full PLRV histogram (bin size 5) for baseline entropy sources. The larger bin size reduces the number of available bins and partially smooths local fluctuations compared to bin size 1.

Source	**1σ exceedance**			**2σ exceedance**		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`REF`	0.354	0.238	0.095	0.062	0.174	0.374
`CSPRNG`	0.184	0.972	0.972	0.041	0.387	0.387
`MIS`	0.400	0.082	0.082	0.000	0.903	0.903

Table D.8: $k\sigma$ exceedance test results on full PLRV histogram (bin size 5) for `CLEAR` manipulations.

Level	1σ exceedance			2σ exceedance		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.810	4.85e-07	4.85e-07	0.714	5.52e-18	7.71e-20
`M8`	0.541	0.001	0.001	0.297	5.01e-08	4.91e-07
`M16`	0.326	0.381	0.504	0.109	0.017	0.057
`M32`	0.277	0.665	0.773	0.085	0.062	0.165
`M64`	0.304	0.504	0.837	0.000	0.883	0.883

Table D.9: $k\sigma$ exceedance test results on full PLRV histogram (bin size 5) for `SET` manipulations.

Level	1σ exceedance			2σ exceedance		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.595	2.52e-07	7.23e-08	0.351	8.35e-18	9.91e-17
`M8`	0.596	9.36e-06	2.89e-04	0.250	5.58e-08	2.95e-06
`M16`	0.420	0.046	0.046	0.200	1.24e-05	3.83e-04
`M32`	0.429	0.037	0.037	0.082	0.071	0.387
`M64`	0.292	0.582	0.341	0.083	0.066	0.174

Table D.10: $k\sigma$ exceedance test results on full PLRV histogram (bin size 5) for `FLIP` manipulations.

Level	1σ exceedance			2σ exceedance		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.422	0.029	0.004	0.234	2.03e-08	2.03e-08
`M8`	0.417	0.054	0.095	0.229	1.18e-06	2.79e-04
`M16`	0.447	0.022	0.043	0.170	2.36e-04	0.019
`M32`	0.327	0.379	0.271	0.061	0.183	0.183
`M64`	0.191	0.959	0.959	0.064	0.165	0.165

Table D.11: $k\sigma$ exceedance test results on full PLRV histogram (bin size 5) for `ALTERNATE` manipulations.

Level	**1σ exceedance**			**2σ exceedance**		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.531	0.004	0.004	0.250	9.01e-06	9.01e-06
`M8`	0.533	8.04e-04	0.005	0.133	0.004	0.147
`M16`	0.438	0.028	0.054	0.104	0.021	0.174
`M32`	0.292	0.582	0.699	0.021	0.648	0.648
`M64`	0.319	0.420	0.305	0.064	0.165	0.362

Table D.12: $k\sigma$ exceedance test results on full PLRV histogram (bin size 5) for `CORRELATION` manipulations.

Level	**1σ exceedance**			**2σ exceedance**		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.679	1.89e-05	1.89e-05	0.429	7.03e-11	7.03e-11
`M8`	0.333	0.341	0.699	0.104	0.021	0.021
`M16`	0.208	0.933	0.878	0.000	0.893	0.893
`M32`	0.340	0.305	0.305	0.000	0.888	0.888
`M64`	0.188	0.966	0.878	0.042	0.374	0.648

Table D.13: $k\sigma$ exceedance test results on central 68% of PLRV histogram (bin size 5) for baseline entropy sources. The combination of larger bins and central region focus further reduces sensitivity to manipulations.

Source	**1σ exceedance**			**2σ exceedance**		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`REF`	0.424	0.069	0.069	0.061	0.189	0.189
`CSPRNG`	0.212	0.868	0.868	0.061	0.189	0.189
`MIS`	0.471	0.020	0.020	0.000	0.795	0.795

Table D.14: $k\sigma$ exceedance test results on central 68% of PLRV histogram (bin size 5) for `CLEAR` manipulations.

Level	1σ exceedance			2σ exceedance		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.929	1.05e-07	1.05e-07	0.929	1.63e-19	1.63e-19
`M8`	0.560	0.003	0.003	0.320	8.77e-07	8.77e-07
`M16`	0.387	0.152	0.152	0.161	0.002	0.012
`M32`	0.281	0.589	0.589	0.094	0.056	0.056
`M64`	0.355	0.256	0.690	0.000	0.764	0.764

Table D.15: $k\sigma$ exceedance test results on central 68% of PLRV histogram (bin size 5) for `SET` manipulations.

Level	1σ exceedance			2σ exceedance		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.780	2.92e-12	2.32e-11	0.500	5.33e-22	1.17e-20
`M8`	0.629	3.77e-05	3.77e-05	0.286	2.61e-07	2.61e-07
`M16`	0.441	0.044	0.020	0.235	1.55e-05	1.55e-05
`M32`	0.485	0.014	0.014	0.061	0.189	0.189
`M64`	0.364	0.221	0.221	0.091	0.061	0.061

Table D.16: $k\sigma$ exceedance test results on central 68% of PLRV histogram (bin size 5) for `FLIP` manipulations.

Level	1σ exceedance			2σ exceedance		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.591	4.98e-05	4.98e-05	0.341	4.14e-11	4.14e-11
`M8`	0.394	0.130	0.221	0.212	9.15e-05	9.15e-05
`M16`	0.406	0.104	0.104	0.094	0.056	0.056
`M32`	0.273	0.634	0.634	0.030	0.447	0.189
`M64`	0.188	0.922	0.922	0.062	0.177	0.056

Table D.17: $k\sigma$ exceedance test results on central 68% of PLRV histogram (bin size 5) for `ALTERNATE` manipulations.

Level	**1σ exceedance**			**2σ exceedance**		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.636	4.87e-04	4.87e-04	0.318	3.30e-06	3.30e-06
`M8`	0.516	0.006	0.006	0.065	0.166	0.166
`M16`	0.394	0.130	0.130	0.030	0.447	0.447
`M32`	0.303	0.487	0.487	0.000	0.785	0.785
`M64`	0.375	0.185	0.185	0.062	0.177	0.177

Table D.18: $k\sigma$ exceedance test results on central 68% of PLRV histogram (bin size 5) for `CORRELATION` manipulations.

Level	**1σ exceedance**			**2σ exceedance**		
	Rate	p	$p_{\log}$	Rate	p	$p_{\log}$
`M4`	0.842	2.81e-07	2.81e-07	0.579	2.94e-12	2.94e-12
`M8`	0.364	0.221	0.221	0.121	0.016	0.016
`M16`	0.242	0.766	0.766	0.000	0.785	0.785
`M32`	0.406	0.104	0.053	0.000	0.775	0.775
`M64`	0.242	0.766	0.766	0.030	0.447	0.447

APPENDIX E

Chi-Square Test Results

This appendix presents χ^2 goodness-of-fit test results for the empirical PLRV distributions obtained from differential privacy experiments described in Chapter 7. The chi-squared test provides a complementary global measure of how well the entire empirical distribution matches the theoretical model.

The tables below show test results under different analysis configurations: full histogram versus central 68% of the distribution, bin size 1 versus bin size 5, and linear-space versus log-space analysis. For each entropy source we report the reduced chi-squared statistic and associated p-values.

Table E.1: χ^2 goodness-of-fit test results on full PLRV histogram (bin size 5) for all entropy sources. The larger bin size reduces the number of available bins and partially smooths local fluctuations compared to bin size 1.

(a) Baseline entropy sources

Source	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
REF	1.173	0.193	1.092	0.306
CSPRNG	0.677	0.959	0.733	0.918
MIS	0.976	0.522	1.018	0.438

(b) CLEAR

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	33.276	0.000	34.088	0.000
M8	8.782	0.000	8.874	0.000
M16	2.895	0.000	2.848	0.000
M32	1.916	0.000	1.230	0.134
M64	0.876	0.710	0.850	0.754

(c) SET

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	49.968	0.000	44.425	0.000
M8	21.422	0.000	19.581	0.000
M16	8.106	0.000	7.653	0.000
M32	2.557	0.000	2.350	0.000
M64	1.341	0.058	1.212	0.149

(d) FLIP

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	45.487	0.000	40.189	0.000
M8	9.167	0.000	8.421	0.000
M16	4.057	0.000	3.159	0.000
M32	1.675	0.002	1.132	0.244
M64	0.905	0.658	0.878	0.708

(e) ALTERNATE

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	30.307	0.000	28.283	0.000
M8	5.715	0.000	4.855	0.000
M16	2.284	0.000	1.635	0.004
M32	1.956	0.000	0.805	0.830
M64	1.103	0.290	0.919	0.630

(f) CORRELATION

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	66.579	0.000	73.590	0.000
M8	2.925	0.000	2.802	0.000
M16	0.725	0.923	0.770	0.876
M32	0.889	0.688	0.909	0.649
M64	1.213	0.148	0.761	0.887

Table E.2: χ^2 goodness-of-fit test results on central 68% of PLRV histogram (bin size 5) for all entropy sources. The combination of larger bins and central region focus further reduces sensitivity to manipulations.

(a) Baseline entropy sources

Source	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
REF	1.303	0.114	1.262	0.144
CSPRNG	0.783	0.808	0.786	0.804
MIS	1.008	0.455	1.017	0.439

(b) CLEAR

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	47.982	0.000	49.034	0.000
M8	11.914	0.000	12.299	0.000
M16	3.860	0.000	3.910	0.000
M32	1.451	0.048	1.456	0.046
M64	0.941	0.560	0.935	0.570

(c) SET

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	73.426	0.000	65.256	0.000
M8	30.805	0.000	28.462	0.000
M16	11.297	0.000	10.819	0.000
M32	3.111	0.000	3.043	0.000
M64	1.505	0.031	1.492	0.034

(d) FLIP

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	65.966	0.000	58.259	0.000
M8	12.221	0.000	11.612	0.000
M16	4.017	0.000	3.932	0.000
M32	1.058	0.377	1.091	0.329
M64	1.007	0.455	1.004	0.461

(e) ALTERNATE

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	43.577	0.000	40.485	0.000
M8	6.681	0.000	6.430	0.000
M16	1.742	0.005	1.703	0.007
M32	0.822	0.754	0.820	0.757
M64	1.044	0.399	1.031	0.418

(f) CORRELATION

Level	χ^2_{red}	p	$\chi^2_{\text{red,log}}$	$p_{\log}$
M4	97.286	0.000	107.862	0.000
M8	4.106	0.000	3.958	0.000
M16	0.805	0.778	0.809	0.773
M32	1.044	0.399	1.011	0.449
M64	0.862	0.693	0.845	0.720

Bibliography

[1] Chun-Heng You, Shuen-Ming Tsai, and Paul Chao. A novel clock recovering circuit to thwart clock glitch attacks on ring-oscillator-based trngs in edge devices like sensors. *Microsystem Technologies*, 29:1–9, 04 2023.

[2] M. Caccia, L. Malinverno, L. Paolucci, C. Corridori, E. Proserpio, A. Abba, A. Cusimano, W. Kucewicz, P. Dorosz, M. Baszczyk, M. Esposito, and P. Svenda. In-silico generation of random bit streams. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 980:164480, 2020.

[3] Cesare Gerolimetto Fabrello, Valeria Rossi, Kamil Witek, Alberto Trombetta, Mateusz Baszczyk, Piotr Dorosz, Wojciech Kucewicz, and Massimo Caccia. Entropy measurement and online quality control of bit streams by a true random bit generator. *Frontiers in Computer Science*, Volume 7 - 2025, 2025.

[4] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A. McKay, Mary L. Baish, and Mike Boyle. Recommendation for the Entropy

Sources Used for Random Bit Generation. NIST Special Publication 800-90B, National Institute of Standards and Technology, Gaithersburg, MD, January 2018. 84 pages.

[5] R. Gennaro. Randomness in cryptography. *Security & Privacy, IEEE*, 4:64– 67, 04 2006.

[6] G. Cowan. *Review of Particle Physics, 2016-2017*, chapter The Monte Carlo Techniques, page 537. Particle Data Group, 2016.

[7] Alexandra Wood, Micah Altman, Aaron Bembenek, Mark Bun, Marco Gaboardi, James Honaker, Kobbi Nissim, David R. O'Brien, Thomas Steinke, and Salil Vadhan. Differential privacy: A primer for a non-technical audience. *Vanderbilt Journal of Entertainment and Technology Law*, 21(1):209, 2020.

[8] Debian Security Team. [SECURITY] [DSA 1571-1] New openssl packages fix predictable random number generator. Debian Security Advisory, May 2008. DSA-1571-1, CVE-2008-0166.

[9] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: results from the 2008 debian openssl vulnerability. In Anja Feldmann and Laurent Mathy, editors, *Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference, IMC 2009, Chicago, Illinois, USA, November 4-6, 2009*, pages 15–27. ACM, 2009.

[10] J. Von Neumann. Various techniques used in connection with random digits. In *Applied Math Series*, volume 12, pages 36–38. National Bureau of Standards, 1951. Notes by G. E. Forsythe.

[11] George Marsaglia. Xorshift rngs. *Journal of Statistical Software*, 8(14):1–9, 2003.

[12] ScienceDirect. Mersenne twister – an overview. https://www.sciencedirect.com/topics/computer-science/mersenne-twister. Accessed: September 16, 2025.

[13] Wikipedia contributors. Randu. https://en.wikipedia.org/wiki/RANDU, 2025. Accessed: 2025-09-19.

[14] Andrew Rukhin and et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, 05 2001.

[15] Cryptomathic. How to choose the right cryptographic key generation algorithm. https://www.cryptomathic.com/blog/how-to-choose-the-right-cryptographic-key-generation-algorithm, February 2018. Accessed: September 16, 2025.

[16] Elaine Barker and John Kelsey. Recommendation for random number generation using deterministic random bit generators. Technical report, 2015-06-24 2015.

[17] The OpenSSL Project. *RAND – the OpenSSL random generator*. OpenSSL, 2018–2019. Accessed: September 16, 2025.

[18] TechTarget Contributors. What are random numbers and how are they used? https://www.techtarget.com/whatis/definition/random-numbers, 2020. Accessed: 2025-09-21.

[19] Fabio Frustaci, Fanny Spagnolo, Pasquale Corsonello, and Stefania Perri. A high-speed and low-power dsp-based trng for fpga implementations. *IEEE Trans. Circuits Syst. II: Express Briefs*, 71(12):4964–4968, 2024.

[20] R. Brown, H. Li, X. Fong, Y. Choi, J. Yao, J. Zhang, M. F. Chang, C. H. Kim, D. Jeon, and S. Yu. Random-telegraph-noise-enabled true random

number generator for hardware security. *Scientific Reports*, 10(1):16829, 2020.

[21] Ars Technica Staff. How a months-old amd microcode bug destroyed my weekend. *Ars Technica*, Oct 2019. Accessed: 2025-09-17.

[22] Linux Reviews. AMD Ryzen 3000 series CPUs can't do Random on boot causing Boot Failure on newer Linux distributions, July 2019. Published 2019-07-09, last edited 2019-07-30.

[23] Elie Noumon Allini. *Characterisation, evaluation and use of clock jitter as a source of randomness in data security*. PhD thesis, 09 2020.

[24] Miguel Herrero-Collantes and Juan Carlos Garcia-Escartin. Quantum random number generators. *Reviews of Modern Physics*, 89(1), Feb 2017.

[25] Palo Alto Networks. What is a quantum random number generator (qrng)? https://www.paloaltonetworks.com/cyberpedia/what-is-a-quantum-random-number-generator-qrng, 2025. Accessed: 2025-09-22.

[26] Smogon University. Rng manipulation guide - pokémon emerald, 2012. Accessed: 2025-09-17.

[27] Bulbapedia contributors. Rng manipulation. Bulbapedia, the community-driven Pokémon encyclopedia, 2025. Accessed: 2025-09-17.

[28] Ian Deakin, Walter Krawec, and William Trost. Implications of entropy on symmetric key encryption resilience to quantum. Technical Report ATIS-I-0000097, Alliance for Telecommunications Industry Solutions (ATIS), February 2023. Version 6.1.

[29] Morris Dworkin. Recommendation for block cipher modes of operation: Methods and techniques. NIST Special Publication 800-38A,

National Institute of Standards and Technology (NIST), December 2001.

[30] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 205–220, Bellevue, WA, August 2012. USENIX Association.

[31] g0tmi1k. Debian openssl predictable prng (cve-2008-0166). https://github.com/g0tmi1k/debian-ssh, 2008. GitHub repository.

[32] Bushing, Marcan, and Sven. Console hacking 2010 – ps3 epic fail. https://fahrplan.events.ccc.de/congress/2010/Fahrplan/events/4087.en.html, 2010.

[33] Cisco Systems. Cisco asa and firepower threat defense software secure boot and random number generation vulnerabilities. https://www.cisco.com/c/en/us/support/docs/csa/cisco-sa-20190501-asa-ftd-entropy.html, May 2019. Accessed: 2025-09-16.

[34] National Vulnerability Database. Cve-2019-15703: Insufficient entropy in prng vulnerability in fortinet fortios. https://nvd.nist.gov/vuln/detail/cve-2019-15703, 2019. Accessed: September 16, 2025.

[35] Dan Shumow and Niels Ferguson. On the possibility of a back door in the nist sp800-90 dual ec prng. In *CRYPTO 2007, Rump Session*, 2007. Available at https://rump2007.cr.yp.to/15-shumow.pdf.

[36] Susan Landau. Close the nsa's back doors. *The New York Times*, September 2013. Opinion piece, accessed 2025-09-17.

[37] Matt Williams. On the possibility of a back door in the nist sp800-90 dual ec prng, 2013. Stories by Williams blog, accessed 2025-09-17.

[38] National Institute of Standards and Technology (NIST). Nist recommends against the use of the dual_ec_drbg random number generator. https://csrc.nist.gov/news/2014/nist-recommends-against-dual-ec-drbg, 2014. NIST announcement.

[39] Massimo Luigi Maria Caccia, Lorenza Paolucci, and Luca Malinverno. Device and method for generating random bit sequences, April 2020. International application PCT/IB2019/058340, filed October 1, 2019.

[40] K. G. McKay. Avalanche breakdown in silicon. *Phys. Rev.*, 94:877–884, May 1954.

[41] Roland H. Haitz. Model for the electrical behavior of a microplasma. *Journal of Applied Physics*, 35(5):1370–1376, 05 1964.

[42] Roland H. Haitz. Mechanisms contributing to the noise pulse rate of avalanche diodes. *Journal of Applied Physics*, 36(10):3123–3131, 10 1965.

[43] Samuela Lomazzi. *Development of novel multi-SiPM systems for biomedical applications*. Phd thesis, Università degli Studi dell'Insubria, July 2021. 33. ciclo, Anno Accademico 2019/2020.

[44] Django Software Foundation. Django: The web framework for perfectionists with deadlines. https://www.djangoproject.com/, 2025. Accessed: 2025-09-26.

[45] TrustedFirmware.org. Op-tee: Open portable trusted execution environment. https://optee.readthedocs.io/en/latest/general/about.html, 2025. Accessed: 2025-09-26.

[46] Pierre L'Ecuyer and Richard J. Simard. Testu01: A C library for empirical testing of random number generators. *ACM Trans. Math. Softw.*, 33(4):22:1–22:40, 2007.

[47] E.B. Barker, J.M. Kelsey, National Institute of Standards, and Technology (U.S.). *Recommendation for random bit generator (RBG) constructions*. NIST special publication. U.S. Department of Commerce, National Institute of Standards and Technology, 2012.

[48] James V. Bradley. *Distribution-Free Statistical Tests*. Prentice-Hall, Englewood Cliffs, NJ, 1968.

[49] Juan Soto and Lawrence Bassham. Randomness testing of the advanced encryption standard finalist candidates, 2000-04-01 00:04:00 2000.

[50] Cesare Gerolimetto Fabrello, Valeria Rossi, Kamil Witek, Alberto Trombetta, and Massimo Caccia. On-line anomaly detection and qualification of random bit streams. In *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 897–904, 2024.

[51] Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the netflix prize dataset, 2007.

[52] Ferdinando Fioretto, Pascal Van Hentenryck, and Juba Ziani. Differential privacy overview and fundamental techniques, 2024.

[53] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, August 2014.

[54] Simson L. Garfinkel and Philip Leclerc. Randomness concerns when deploying differential privacy. In *Proceedings of the 19th Workshop on Privacy in the Electronic Society*, WPES'20, page 73–86, New York, NY, USA, 2020. Association for Computing Machinery.

[55] U.S. Census Bureau. Disclosure avoidance system for the 2020 census, end-to-end release. https://github.com/uscensusbureau/census2020-das-e2e, 2019. Accessed: 2025-11-16.

[56] Miklos Santha and Umesh V. Vazirani. Generating quasi-random sequences from semi-random sources. *Journal of Computer and System Sciences*, 33(1):75–87, 1986.

[57] Yevgeniy Dodis, Adriana Lopez-Alt, Ilya Mironov, and Salil Vadhan. Differential privacy with imperfect randomness. Cryptology ePrint Archive, Paper 2012/435, 2012.

[58] Clément Canonne. What is delta, and what difference does it make? DifferentialPrivacy.org, 03 2021. https://differentialprivacy.org/flavoursofdelta/.

[59] Cynthia Dwork, Guy N. Rothblum, and Salil Vadhan. Boosting and differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 51–60, 2010.

[60] Damien Desfontaines. The privacy loss random variable. https://desfontain.es/blog/privacy-loss-random-variable.html, 03 2020. Ted is writing things (personal blog).

[61] Damien Desfontaines. Local vs. central differential privacy. https://desfontain.es/blog/local-global-differential-privacy.html, 06 2019. Ted is writing things (personal blog).

[62] Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM Journal on Computing*, 41(6):1673–1693, 2012.

[63] Seidu Inusah and Tomasz J. Kozubowski. A discrete analogue of the laplace distribution. *Journal of Statistical Planning and Inference*, 136(3):1090–1102, 2006.

[64] Victor Balcer and Salil Vadhan. Differential privacy on finite computers, 2019.

[65] Clement Canonne, Gautam Kamath, and Thomas Steinke. Discrete gaussian for differential privacy. *Journal of Privacy and Confidentiality*, 12(1), July 2022.

[66] Ieee standard for binary floating-point arithmetic. *ANSI/IEEE Std 754-1985*, pages 1–20, 1985.

[67] Cynthia Dwork, Adam Smith, Thomas Steinke, and Jonathan Ullman. Exposed! a survey of attacks on private data. *Annual Review of Statistics and Its Application (2017)*, 2017.

[68] Daniel Lemire. How many floating-point numbers are in the interval [0,1]?, 2017.

[69] Taylor R. Campbell. Uniform random floats: How to generate a double-precision floating-point number in [0, 1] uniformly at random given a uniform random source of bits, 2014.

[70] Ilya Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 650–661, New York, NY, USA, 2012. Association for Computing Machinery.

[71] Naoise Holohan and Stefano Braghin. Secure random sampling in differential privacy, 2021.

[72] Naoise Holohan, Stefano Braghin, and Mohamed Suliman. Securing floating-point arithmetic for noise addition. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 1954–1966, New York, NY, USA, 2024. Association for Computing Machinery.

[73] Samuel Haney, Damien Desfontaines, Luke Hartman, Ruchit Shrestha, and Michael Hay. Precision-based attacks and interval refining: how to break, then fix, differential privacy on finite computers, 2022.

[74] Joseph Near, David Darais, Naomi Lefkovitz, and Gary Howarth. Guidelines for evaluating differential privacy guarantees. Special Publication NIST SP 800-226, National Institute of Standards and Technology, March 2025.

[75] Naoise Holohan. Random number generators and seeding for differential privacy, 2023.

[76] OpenDP Project. OpenDP: An Open-Source Project for Differential Privacy. https://opendp.org/, 2025. Accessed: 2025-07-24.

[77] Michael Hay, Marco Gaboardi, and Salil Vadhan. A programming framework for opendp. In *6th Workshop on the Theory and Practice of Differential Privacy (TPDP 2020)*, 2020.

[78] IBM. Differential privacy library. https://github.com/IBM/differential-privacy-library. Accessed: February 12, 2026.

[79] Google. Google differential privacy. https://github.com/google/differential-privacy, 2019. Accessed: 2025-07-24.

[80] Differential Privacy Team. Secure noise generation. Technical report, Google, June 2020.

[81] Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[82] Bo Meek Chris, Thiesson and Heckerman David. US Census Data (1990). UCI Machine Learning Repository, 2001. DOI: https://doi.org/10.24432/C5VP42.

[83] NIST. *1.3.5.15. Chi-Square Goodness-of-Fit Test*. National Institute of Standards and Technology. Accessed: September 16, 2025.

[84] NIST. *1.3.5.16. Kolmogorov-Smirnov Goodness-of-Fit Test*. National Institute of Standards and Technology. Accessed: September 16, 2025.

[85] Jean Dickinson Gibbons and Subhabrata Chakraborti. *Nonparametric Statistical Inference*. Chapman and Hall/CRC, 5 edition, 2010.

[86] Peter Occil. Bernoulli factory algorithms, 2025. Version dated 2025-11-06.

[87] Jacob Zhong. dashu: A library set of arbitrary precision numbers, 2024. Rust crate.

[88] http://www.bzip.org/.