entropy

MDPI

Article

Shannon Entropy Loss in Mixed-Radix Conversions

Amy Vennos * and Alan Michaels

Hume Center for National Security and Technology, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA; ajm@vt.edu
* Correspondence: avennos@vt.edu

Abstract: This paper models a translation for base-2 pseudorandom number generators (PRNGs) to mixed-radix uses such as card shuffling. In particular, we explore a shuffler algorithm that relies on a sequence of uniformly distributed random inputs from a mixed-radix domain to implement a Fisher–Yates shuffle that calls for inputs from a base-2 PRNG. Entropy is lost through this mixed-radix conversion, which is assumed to be surjective mapping from a relatively large domain of size 2^J to a set of arbitrary size n. Previous research evaluated the Shannon entropy loss of a similar mapping process, but this previous bound ignored the mixed-radix component of the original formulation, focusing only on a fixed n value. In this paper, we calculate a more precise formula that takes into account a variable target domain radix, n, and further derives a tighter bound on the Shannon entropy loss of the surjective map, while demonstrating monotonicity in a decrease in entropy loss based on increased size J of the source domain 2^J. Lastly, this formulation is used to specify the optimal parameters to simulate a card-shuffling algorithm with different test PRNGs, validating a concrete use case with quantifiable deviations from maximal entropy, making it suitable to low-power implementation in a casino.

Keywords: PRNG; Shannon entropy; mixed-radix conversion

check for updates

Citation: Vennos, A.; Michaels, A. Shannon Entropy Loss in Mixed-Radix Conversions. *Entropy* **2021**, *23*, 967. https://doi.org/10.3390/e23080967

Academic Editor: Yong Deng

Received: 17 May 2021
Accepted: 24 July 2021
Published: 27 July 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The residue number system (RNS), initially proposed in 1959, was derived from the third century Chinese remainder theorem [1]. RNS architectures are now applied in many growing fields such as cryptography [2], image-processing systems [3], and error-correction codes [4] due to its convenience in parallel computing. Parallel processing with RNS often involves replacing a typical base-2 system with a different number representation system built upon two or more coprime number bases, which we refer to as mixed-radix (MR) [5].

Research in MR calculations in RNS implementations has increased due to applications of circuits in which radix choice affects their speed, power, and area [6]. In some implementations, such as those using low-power devices that require some level of security, binary-number representation may result in poor implementation or may not be applicable at all, requiring the consideration of another radix. For example, this distinction is apparent in Reed–Muller expansions over Galois fields involving cryptographic circuits [7]. In this application, using a lower-order radix usually requires less computation, which decreases the circuit's area, but this technique allows for increased power consumption due to the large amount of interconnections. On the other hand, choosing a higher radix decreases power consumption, but increases the circuit area [8]. In the case that an optimal radix cannot be found, it is common to convert pseudorandomly generated words from one number base to another [9]. It is important to examine efficient ways to use MR techniques in processes such as Reed–Muller expansions. Researchers thus worked on independent MR conversion algorithms and techniques that have short conversion times [6,10].

MR techniques were also used for fast Fourier transform (FFT) pruning, which was designed to improve computational efficiency [11,12]. These algorithms have conventional applications, such as recording the flicker of voltage in smart homes [13]. Similar to the

general circuit application, the choice of radix is important, and some radices may be better disposed in different scenarios depending on operation conditions. For example, higher radices reduce latency for memory-shared FFT architectures [14]. This additional use of MR conversions has prompted additional research on MR [12,15,16].

Most digital logic and computing systems are base-2, but algorithms such as Cooley–Tukey [17] offer equivalent mixed-radix representations to achieve the same overall calculation, but as a parallel composition of many small RNS operations. In particular, most PRNG methods were designed to produce bistate binary values, and methods to quantify PRNG output randomness usually require their binary form [18]. Consequently, the most common output of a PRNG is a k-bit binary word that may be viewed as an element $r \in \mathbb{Z}_{2^k}$. Examples include linear feedback shift registers (LFSR) [19], the Mersenne Twister [20], and thermal entropy-based true random-number generators (TRNG) [21]. As such, the primary source domain for RNG values is typically on $\mathbb{Z}_{2^k}$, while other algorithms consuming random words may wish for nonbinary random words. The specific example of optimally shuffling a deck of cards is discussed later on in this paper.

To support this evaluation, we focus on quantifying a specific MR application: calculating the Shannon entropy loss of an onto map from a source domain of size 2^J to a target domain of arbitrary size n. Ref. [22] developed a lower bound on the Shannon entropy loss from mapping between an RNS-based source domain onto a $\mathbb{Z}_{2^k}$ target domain (a fixed n value), but this bound involves approximations that are only applicable for extraordinarily large source domains. In this paper, we calculate a more precise formulation that reveals the exact change in entropy of the onto map; our truncation of a Taylor series approximation offers a useable closed-form approximation that may be expanded by the reader. These calculations are referenced later to choose the correct parameter to reduce computations in our chosen application of card shuffling.

On the basis of these calculations, we apply MR conversions in a casino shuffling application with a base-2 PRNG component. As the casino industry is developing, casinos are relying on robotic card dealers to reduce the cost of hiring human card dealers and decrease the time that it takes to shuffle cards. Card-shuffling machines that randomize one deck while another is in use remove lost time and the possibility for fraudulent shuffles by a human dealer. These machines, which usually utilize a simple riffle shuffle, are rented for USD 500 per machine per month [23]. Additionally, a casino might benefit from the determinism of PRNG values to more precisely predict payouts; the same determinism becomes a potential avenue for dynamically skewing player odds [24], though the use of physics-based true RNGs can mitigate that risk.

Many prior works explored testing the quality of a card shuffle. More generally, card randomization is a popular problem in many fields such as statistics, combinatorics, and communications [25]. Markov proved results that analyzed card shuffles as early as 1906 using finite Markov chains [26]. More rigorous methods were also introduced, such as those using Fourier analysis and quantifying the entropy loss of riffle shuffling [27]. More recently, statistical tests for randomness, such as the FSU DIEHARD suite [28], were used to quantify the randomness of a permutation [29]. Additionally, entropy formulations, such as fiber entropy, were utilized to measure shuffle output randomness [30]. In our application, we determined the quality of the shuffle on the basis of Shannon entropy loss of the mapping process within the shuffle and simulating the shuffle algorithm using MATLAB with different PRNGs.

PRNGs and lightweight cryptographic primitives are useful in many applications. Though the option of using a TRNG in place of a lightweight primitive always provides a result with the highest entropy, utilizing PRNGs in these applications may be feasible and cost-efficient. In this paper, we demonstrate the feasibility of implementing a real-time, low-power card-shuffling algorithm with negligible entropy loss. The core algorithm is described in Section 2, followed by a complete derivation of Shannon entropy loss in Section 3. A simulation model is then presented in Section 4, followed by overall conclusions in Section 4.

2. Materials and Methods

To begin, we introduce a card-shuffling algorithm characterized by modulo arithmetic on RNG outputs and recurrent Fisher–Yates permutations [31]. The system, illustrated in Figure 1, shuffles a standard card deck as an ordered array $\mathcal{C} = [1|2|\ldots|52]$ by repeated Fisher–Yates-based shuffles that utilize random numbers derived from any base-2 RNG.

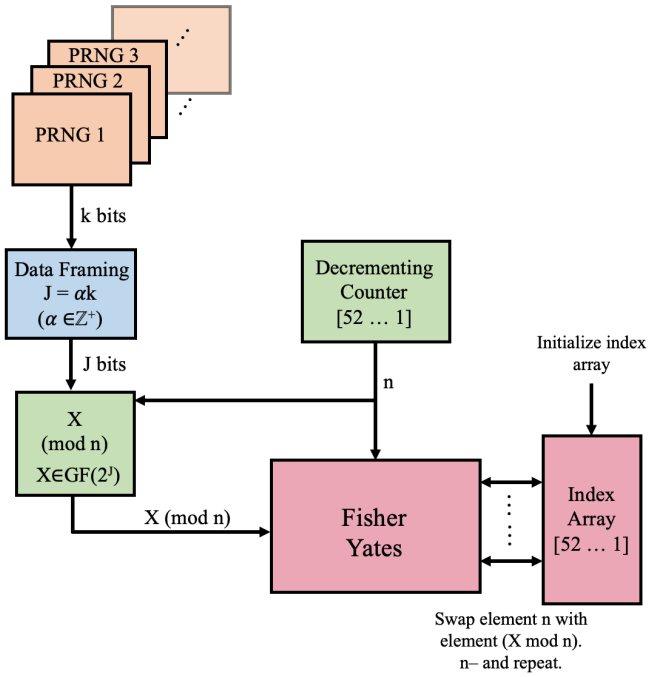

Figure 1. Shuffler diagram.

Let k denote the width of the RNG output. Each clock cycle, k bits are produced by the RNG and are sent to the data-framing stage, which involves concatenating the bits from α PRNG outputs to create a word X of length $J = \alpha \cdot k$. The product of this stage is a word X that has enough bits to perform future modulo operations in the algorithm. For example, an 8-bit RNS with $J = 64$ is framed collecting eight successive outputs and concatenating them together. Figure 2 demonstrates how X is framed in this step.

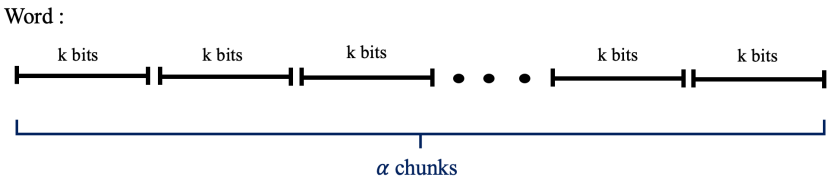

Figure 2. Data-framing stage concatenates α words, each composed of k bits, to create a J-bit string.

Once X is created, the J-bit string is mapped from a source domain of $\mathbb{Z}_{2^J}$ to a dynamic target domain, n (starting at $n = 52$), effectively producing a near-ideal uniformly distributed random value on $\mathbb{Z}_n$. These modulo residuals are used in a Fisher–Yates permutation on $\mathcal{C}$ for each of the 51 iterations of the modulo calculation. Once the last Fisher–Yates shuffle is completed, deck $\mathcal{C}$ is fully shuffled.

One RNG may be better suited for this application than others are. in this case, "better" refers to both how closely the chosen RNG approaches maximal entropy, whether successive samples offer an opportunity to reverse-engineer, and the associated costs (money or computational complexity). As such, a TRNG would not contribute to distortions in output randomness, while utilizing a low-complexity PRNG may result in a noticeably

nonrandom shuffle. However, the consumer of this algorithm may prefer the cost effectiveness of low-power PRNG instead of TRNG. This paper attempts to answer the underlying question of how to optimize the needs of both randomness and cost effectiveness.

2.1. Mixed-Radix Conversion Entropy

The shuffler's deviation from maximal entropy occurs both through the PRNG algorithms that produce random input numbers and the onto mapping process of the mixed-radix conversion. In this section, we provide a background on Shannon entropy, the metric we used to measure entropy loss, and we explain how to calculate the Shannon entropy loss of a mixed-radix conversion from $\mathbb{Z}_{2^k} \to \mathbb{Z}_n$.

The Shannon definition of entropy quantifies the memoryless predictability in an event. In particular, the Shannon entropy of a discrete alphabet of possible events of size L is defined by

$$H = -\sum_{i=1}^{L} p_i \log_n p_i. \tag{1}$$

We utilize Equation (1) to measure the entropy of modulo reductions $A \pmod B$, which can be represented by a surjective map $\mathcal{A} \to \mathcal{B}$, where $|\mathcal{A}| = A$ and $|\mathcal{B}| = B$. The mapping process can be visualized as placing the values $1, \ldots, A$ into bins that represent their residual modulo B, creating a histogram of values as illustrated in Figure 3.

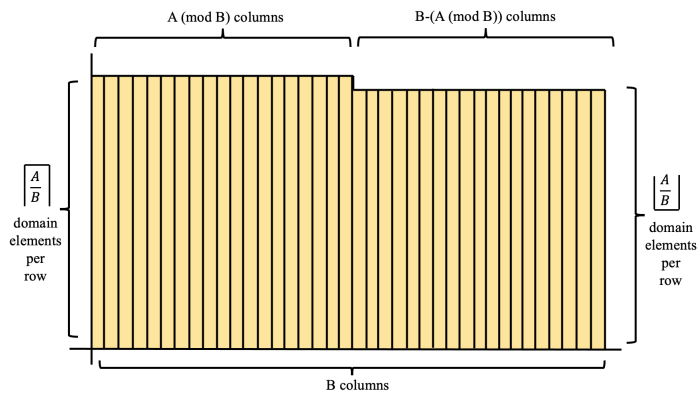

Figure 3. Histogram of residuals resulting from surjective mapping $\mathcal{A} \to \mathcal{B}$, where $|\mathcal{A}| = A$ and $|\mathcal{B}| = B$.

Splitting up the histogram on the basis of column height, we could calculate the Shannon entropy of onto map $\mathcal{A} \to \mathcal{B}$. From this splitting, we tailor the Shannon entropy formula in Equation (1) to obtain

$$\begin{aligned} H \quad = \quad & -(A \pmod B)\left[\frac{1}{A}\left\lceil\frac{A}{B}\right\rceil \log_B \frac{1}{A}\left\lceil\frac{A}{B}\right\rceil\right] \\ & -(B - A \pmod B)\left[\frac{1}{A}\left\lfloor\frac{A}{B}\right\rfloor \log_B \frac{1}{A}\left\lfloor\frac{A}{B}\right\rfloor\right]. \end{aligned} \tag{2}$$

Further, define entropy loss for onto map $\mathcal{A} \to \mathcal{B}$ as shown in Equation (3).

$$\gamma_{A \to B} = 1 - H(A, B). \tag{3}$$

The small entropy loss of a radix conversion indicates that more randomness is retained; likewise, this entropy-loss metric must always be positive and produce a value $0 \leq \gamma_{A \to B} \leq 1$. We see the application of Equation (3) later in this paper, where entropy is calculated for mappings in which $A = 2^J$ and B ranges from 2 to 52. Calculating these values is achieved by applying an adaptation of Equation (2).

2.2. Entropy Sources

Entropy is lost through the shuffler's onto mapping process (discussed in the beginning of Section 2), and randomness is also lost through the chosen input RNG. Even though utilizing a TRNG in lieu of a PRNG or lightweight primitive provides the highest level of entropy, utilizing a PRNG in this application may be a feasible and cost-efficient choice. The concept of this experiment was, therefore, to build a card shuffler that is low-cost and low-power. We implemented the shuffling algorithm with different PRNGs and a TRNG to test their feasibility to create a random shuffle. These algorithms are listed and briefly described below.

- *LFSR*: The linear feedback-shift register (LFSR) is a polynomial-based code on a binary space in which the output sequence is repeatedly replaced by an exclusive-or (XOR) of its last state. Since LFSR produces linear operations on a finite set of states, it creates a cyclic pattern. LFSR can produce a long cycle, or period, if the polynomial and input sequence is chosen correctly. Combined with its speed, this makes LFSR a reasonable choice for applications that require pseudorandom number generation. We used two LFSRs, LFSR-16 and LFSR-24, corresponding to polynomials $z^{16} + z^{15} + z^{13} + z^4 + 1$ and $z^{24} + z^{23} + z^{22} + z^{17} + 1$, respectively.
- *Combined Multiple Recursive PRNG*: The combined multiple recursive PRNG combines multiple linear congruential generators, which are fast and low-cost PRNGs that use recurrence relations to obtain the next state. Linear congruential generator quality depends on choice of parameters, but their speed and low cost justified their use in cryptographic and statistical applications [32]. We used the MATLAB mrg32k3a variant, which has a period of 2^{191} [33].
- *Multiplicative Lagged Fibonacci*: a type of lagged Fibonacci generator (LFG) that was created in the 1950s in hopes of improving the linear congruential generator. These PRNGs utilize a recurrence formula on the basis of the Fibonacci sequence. A multiplicative LFG (MLFG) uses multiplication as the operation in question [34]. We used MATLAB's mlfg6331_64, whose period is 2^{124} [33].
- *Mersenne Twister*: the default PRNG in simulation engines such as MATLAB and Python, the Mersenne Twister offers efficient speed and a large repetition period [20]. We used MATLAB's mt19937ar, which has a period of $2^{19937} - 1$, a Mersenne prime [35].
- Multiplicative congruential generator (MCG): Introduced in 1958, MCG is a specialized version of the linear congruential generator in which a parameter is set to zero [36]. Output quality passed randomness tests, but one must be careful in choosing some parameters [37]. We used mcg16807, which has a period of $2^{31} - 2$ [33].
- Modified subtract with borrow generator (MSBG): MSBG was proposed in 1991 as an improvement to the lagged-Fibonacci PRNG [38], and later applied in the RANLUX generator created for particle-physics simulations [39]. We used MATLAB version swb2712, which has a period length of 2^{1492} [33].

After simulating the card shuffle using each of these RNGs, we analyzed the randomness of their outputs. We then discuss how RNG quality affects the resulting shuffle output. The goal is to uncover that, though a very low-power PRNG such as the LFSR may produce noticeable patterns within the output card shuffle, another lightweight primitive may be just as functional in producing a good shuffle as a high-cost TRNG. In order to explore these topics, we need to find an optimal testing value for the second parameter, J. This process is discussed in the following subsection.

2.3. Calculating an Optimal J

In this section, we find the value of J that should remain constant to test the feasibility and efficiency of the RNGs. The value of J should be large enough so that the algorithm's output is shuffled to acceptable standards, but also small enough to minimize the amount of arithmetic performed in the modulo calculation. To determine a metric of shuffle quality, an engineer may write code and actually prove feasibility on a small device, while a mathematician could show some guarantee about the probabilities on the basis of the

shuffles that we see. We employed a hybrid of these two perspectives, where we considered both the numerical Shannon entropy loss from the shuffle and the utilized hardware to implement the shuffle. The following subsections explore both of these approaches.

3. Results

3.1. Trend of $\gamma_{A\to\ B}$ When $A = 2^J$

We visualized the entropy loss of the shuffler's onto mapping process to notice any pattern in the behavior of $\gamma_{A\to B}$ with source and domain sizes that are relevant in this chapter. Our general frame of reference remains with an assumption that $2^J \gg n$, We created a simulation that tabulates the entropy loss for the onto map $\mathcal{A} \to \mathcal{B}$, where $|\mathcal{A}| = 2^J$ and $|\mathcal{B}| = n$ as n ranges from 2 to 52. The result was a surface plot histogram, shown in Figure 4, of entropy loss values of $J \in \{6, 7, ..., 64\}$ and $B \in \{2, 3, ...52\}$. These entropy loss values were calculated on a deciBel scale to account for their size.

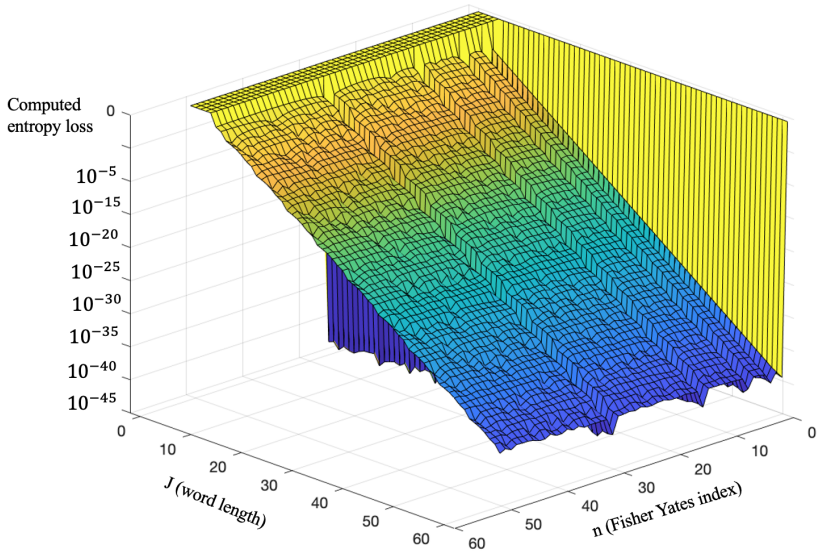

Figure 4. Entropy-loss values as J in the range of 6–64, and as B in the range of 2–52.

First, the n values that are a power of 2 ($n \in \{4, 8, 16, 32, 64\}$) have entropy-loss values of 0. This is because 2 raised to any power $J \geq 6$ modulo n equals 0; n automatically wraps around on an even basis, thereby creating zero residual. Visually, this plot appears to be monotonically decreasing with increasing J, yet the implications of that monotonicity represent a significant simplification in system design (i.e., *use the highest J value possible*) if true. Under the core assumption of $2^J \gg n$, we demonstrate this monotonicity, which is ultimately useful in our choice of J, since this statement justifies that choosing a larger value of J always results in lower entropy loss.

The derived approximation quantifies entropy loss, $\gamma_{2^J\to n}$ and $\gamma_{2^{J+1}\to n}$ of a surjective map $\mathcal{A} \to \mathcal{B}$, in order to estimate the deviation from maximal entropy. The monotonicity conclusion falls apart if 2^J is not much greater than n, but it is utilized later in this chapter to determine a suitable value of J in a casino-game implementation depending on processor size. The simplifications of H_J and H_{J+1} are also helpful in that their algebraic form confirms a maximal entropy of 1 for H_J.

3.2. Approximation of Entropy and Entropy Loss for $\mathbb{Z}_{2^J} \to \mathbb{Z}_n$

Claim: Entropy loss as a function of J for our chosen ranges of (J, n) is a monotonically decreasing function when $2^J \gg n$. Additionally, the entropy, H_J, of a surjective map $\mathbb{Z}_{2^J} \to \mathbb{Z}_n$ under the condition of $2^J \gg n$ is

$$H_J \approx 1 + \frac{2x^3 - nx^2 - n^2x}{2n(2^J)^2} + \frac{n^3x - 3n^2x^2 + 4nx^3 - 2x^4}{2n(2^J)^3}, \tag{4}$$

where $x = 2^J \pmod{n}$. We also show $H_{J+1} - H_J \geq 0$ for all n, when $2^J >> n$.

Proof. Denote H_J the entropy of the onto map $\mathcal{A} \to \mathcal{B}$, where $A = 2^J$ and $B = n$. Let $x = 2^J \mod n$, where $2^J \gg n$. Using residual count x, the ceiling and floor values that occur in entropy value H are

$$\left\lceil \frac{2^J}{n} \right\rceil = \frac{2^J}{n} + \frac{(n-x)}{n}, \tag{5}$$

$$\left\lfloor \frac{2^J}{n} \right\rfloor = \frac{2^J}{n} - \frac{x}{n}. \tag{6}$$

For values including 2^{J+1} in the proof, we tabulate the following values in terms of x.

$$2^{J+1} \pmod{n} = \begin{cases} 2x, & \text{if } x \leq \frac{n}{2} \\ 2x - n, & \text{if } x > \frac{n}{2} \end{cases} \tag{7}$$

$$\left\lceil \frac{2^{J+1}}{n} \right\rceil = \begin{cases} \frac{2^{J+1}}{n} + \frac{n-2x}{n}, & \text{if } x \leq \frac{n}{2} \\ \frac{2^{J+1}}{n} + \frac{n-(2x-n)}{n}, & \text{if } x > \frac{n}{2} \end{cases} \tag{8}$$

$$\left\lfloor \frac{2^{J+1}}{n} \right\rfloor = \begin{cases} \frac{2^{J+1}}{n} - \frac{2x}{n}, & \text{if } x \leq \frac{n}{2} \\ \frac{2^{J+1}}{n} + \frac{n-2x}{n}, & \text{if } x > \frac{n}{2} \end{cases} \tag{9}$$

Then,

$$\begin{aligned} H_J &= -\frac{x}{2^J}\left(\frac{2^J + (n-x)}{n}\right)\log_n\left(\frac{2^J + (n-x)}{n2^J}\right) \\ &\quad -\frac{(n-x)}{2^J}\left(\frac{2^J - x}{n}\right)\log_n\left(\frac{2^J - x}{n2^J}\right). \end{aligned} \tag{10}$$

Using a degree-2 Taylor series approximation for the logarithms,

$$\begin{aligned} H_J &= -\frac{x}{2^J}\left(\frac{2^J + (n-x)}{n}\right)\left(\log_n\left(1 + \frac{n-x}{2^J}\right) - 1\right) \\ &\quad -\frac{n-x}{n}\left(\frac{2^J - x}{2^J}\right)\left(\log_n\left(1 - \frac{x}{2^J}\right) - 1\right) \end{aligned} \tag{11}$$

$$\begin{aligned} &= -\frac{x}{2^J}\left(1 + \frac{2^J - x}{n}\right)\left(\frac{n-x}{2^J} - \frac{(n-x)^2}{2(2^J)^2} + \left(\sum_{i=3}^{\infty}(-1)^{i+1}\frac{(n-x)^i}{i(2^J)^i}\right) - 1\right) \\ &\quad -\frac{n-x}{n}\left(\frac{2^J - x}{2^J}\right)\left(-\frac{x}{2^J} + \frac{x^2}{2(2^J)^2} - \left(\sum_{i=3}^{\infty}(-1)^{i+1}\frac{x^i}{i(2^J)^i}\right) - 1\right). \end{aligned} \tag{12}$$

The higher-order terms in the infinite summations of Equation (12) should be recognized as an alternating series that rapidly decays given our prior assumption of $2^J \gg n$. Via the alternating series remainder theorem [40], both residual summations are bounded by their third positive terms. With the extraction of these terms into a residual $R(J, n)$ that incorporates the leading scalar coefficients, we may restate Equation (12) as

$$\begin{aligned} H_J &= -\frac{x}{n}\left(\frac{2^J+(n-x)}{2^J}\right)\left(\frac{n-x}{2^J}-\frac{(n-x)^2}{2(2^J)^2}-1\right) \\ &\quad -\frac{n-x}{n}\left(\frac{2^J-x}{2^J}\right)\left(-\frac{x}{2^J}+\frac{x^2}{2(2^J)^2}-1\right) \\ &\quad +R(J,n), \end{aligned} \tag{13}$$

where

$$\begin{aligned} |R(J,n)| &\leq \left|-\frac{1}{3}\left(\frac{x}{n}\right)\left(\frac{2^J+(n-x)}{2^J}\right)\frac{(n-x)^3}{(2^J)^3}\right| \\ &\quad +\left|\frac{1}{3}\left(\frac{n-x}{n}\right)\left(\frac{2^J-x}{2^J}\right)\frac{x^3}{(2^J)^3}\right| \\ &= o((\frac{n}{2^J})^3). \end{aligned} \tag{14}$$

In stating the little *o()* notation, we recognize that the scalar coefficients of each term are approximately $\frac{1}{3}$ each; thus, any additive combination between them is less than 1; likewise, $x \leq n$, giving us an overall bound on residual term $R(J,n)$. Further, given the magnitude of n in comparison to the magnitude of 2^J for our application, and the accompanying assumption that $2^J \gg n$, this residual term $R(J,n)$ decays exceedingly quickly. If the assumption of $2^J \gg n$ is invalid, either additional terms must be retained in the Taylor series expansion and/or the chosen approach is invalid for drawing a conclusion. In the extreme case where $2^J \simeq n$, the onto map contains very few domain elements per row ($\lceil \frac{A}{B} \rceil$ described earlier); thus, the probabilities considered in the Shannon entropy estimation diverge from uniform distribution.

This entropy value simplifies to

$$H_J = 1+\frac{2x^3-nx^2-n^2x}{2n(2^J)^2}+\frac{n^3x-3n^2x^2+4nx^3-2x^4}{2n(2^J)^3}+o((\frac{n}{2^J})^3) \tag{15}$$

Equation (15) is a key simplification of H_J. This form allows for us to simplify values in subtraction $H_J - H_{J+1}$. It remains to prove that H_J is a decreasing function of J, for which we temporarily ignore the algebra of the bounded residual term. We broke that evaluation into two independent cases on the basis of the piecewise nature of Equations (7)–(9) for incremented case $J+1$.

Case 1 for H_{J+1}: $x \leq \frac{n}{2}$. We used similar methods as those that we used in deriving Equation (15) to determine

$$H_{J+1,x\leq\frac{n}{2}} \sim 1+\frac{\frac{1}{2}n^3-\frac{3}{2}n^2x+nx^2}{2n(2^J)^2}+\frac{-\frac{3}{4}n^4+\frac{13}{4}n^3x-\frac{9}{2}n^2x^2+2nx^3}{2n(2^J)^3} \tag{16}$$

With further simplification and subtraction, we have that

$$\begin{aligned} H_{J+1,x\leq\frac{n}{2}} - H_J &\sim \underbrace{\frac{\frac{1}{2}n^3-\frac{3}{2}n^2x+nx^2-2x^3+nx^2+n^2x}{2n(2^J)^2}}_{(*)} \\ &\quad + \underbrace{\frac{-\frac{3}{4}n^4+\frac{13}{4}n^3x-\frac{9}{2}n^2x^2+2nx^3-n^3x+3n^2x^2-4nx^3+2x^4}{2n(2^J)^3}}_{(**)}. \end{aligned} \tag{17}$$

Given the prior assumption that $2^J \gg n$, (**) is arbitrarily small with practical bound on the order of $\frac{n^3}{(2^J)^3}$. This terms is also on par with the previously calculated $R(J,n)$ residual. Consequently, we work with (*) to evaluate whether the difference of entropies is greater than 0. If we define $x = \frac{n}{2} + \epsilon$, letting ϵ be the residual between $\frac{n}{2}$ and x, $0 \leq \epsilon \leq \frac{n}{2}$. Then,

$$\begin{aligned} \text{numerator of (*)} &= \frac{1}{2}(n^3 - n^2x + 4nx^2 - 4x^3) \\ &= \frac{1}{2}\left(\frac{n^3}{2} - \epsilon^2 n - 2\epsilon^3\right). \end{aligned} \tag{18}$$

So,

$$(*) = \frac{1}{4n(2^J)^2}\left(\frac{n^3}{2} - \epsilon^2 n - 2\epsilon^3\right). \tag{19}$$

Since $\epsilon \leq \frac{n}{2}$, $(*) \geq 0$. Thus, H_J is an increasing function as J increases. Consequently, entropy loss $1 - H_J$ is a decreasing function in this case. The second case follows likewise, but we show the formulas for further clarification.

Case 2 for H_{J+1}: $x > \frac{n}{2}$. The equivalent for Equation (16) is

$$H_{J+1,x>\frac{n}{2}} \sim 1 + \frac{-\frac{1}{2}n^2x - nx^2 + 2x^3}{2n(2^J)^2} + \frac{\frac{1}{4}n^3x - \frac{3}{2}n^2x^2 + 4nx^3 - 2x^4}{2n(2^J)^3} \tag{20}$$

We use simplification and subtraction similarly to Case 1 to reveal

$$\begin{aligned} H_{J+1,x>\frac{n}{2}} - H_J \simeq & \underbrace{\frac{-\frac{1}{2}n^2x - nx^2 + 2x^3 - 2x^3 + nx^2 + n^2x}{2n(2^J)^2}}_{(***)} \\ & + \underbrace{\frac{\frac{1}{4}n^3x - \frac{3}{2}n^2x^2 + 4nx^3 - 2x^4 - n^3x + 3n^2x^2 - 4nx^3 + 2x^4}{2n(2^J)^3}}_{(****)} \end{aligned} \tag{21}$$

As in the first case, (****) is substantially smaller than (***) on the order of $\frac{n^3}{(2^J)^3}$, which is comparable to the previously discussed $R(J,n)$, so we worked with (***). We applied substitution $x = \frac{n}{2} - \epsilon$ and further simplifications as before. ϵ is still the residual between $\frac{n}{2}$ and x, $0 \leq \epsilon \leq \frac{n}{2}$. We arrive at the analog of Equation (19),

$$(***) = \frac{1}{2n(2^J)^2}\left(\frac{n^3}{4} - \frac{n^2}{2}\epsilon\right). \tag{22}$$

Since $\epsilon \leq \frac{n}{2}$, $(***) \geq 0$, and the discarded residual terms are sufficiently smaller than either difference in Equation (19) or (22), we conclude the monotonicity under the initial assumption of $2^J \gg n$. □

We end this subsection by emphasizing the importance of Equation (4). We simplified the entropy approximation into a closed form that approaches the value of 1. This value is relevant in that it represents the maximal entropy of H_J, and thus $0 \leq H_J \leq 1$.

3.3. Applications and Relevance

Our reduction in Shannon entropy is both applicable for the card-shuffling application in this paper, and to measure entropy loss in other nonbinary applications. Binary modifications, such as gray code and the complex-binary-number system, are applied in puzzles [41], positioning technology [42], genetic algorithms [43], and granting faster processing for problems that handle complex numbers [44,45]. There are also many physical processes that are not necessarily optimized with radix 2. Research was conducted

to examine the advantages of adopting the octal number system to represent SI units, money, time, and calendar days for computer accessibility [46]. Other systems include the decimal-number system, which is often utilized when high precision is necessary [47], and the alphanumeric number system, which is commonly utilized in storing colors using the RGB color model [48].

Some processes require a nonbinary source of randomness. PRNGs are designed to utilize nonbinary operations, such as the nonbinary Galois linear feedback shift register (LFSR), in which the exclusive-or performs addition modulo-q instead of modulo-2 [49]. In fact, some processes require a mix of different radices or mixed-radix as a source of randomness for PRNGs. This is common in instances in which there is interest in a uniform value on a specific domain size. In these cases, there is either the drawback of entropy loss that is not truly uniform from relying on a small PRNG, or the disadvantage of extra processing while utilizing a larger PRNG. One such example is mapping from one RNS space to a second coprime RNS space. Employing mixed-radix and nonbinary-number systems is useful in depicting metrics. For example, representing time in terms of years, days, hours, minutes, and seconds requires a number system that takes into account the cycle length of each unit. Currency is another example, in which we denote money in terms of dollars, quarters, nickels, and dimes [50], yet commensurate analysis would fail in fanciful analysis using prime-based currency (galleons, sickles, knuts). Generating a random number of seconds within a minute would not provide uniformity if it began with a binary PRNG; this may be performed with a conscious understanding of entropy loss given a J-sized processor.

Other applications of mixed-radix and nonbinary-number systems exist in games and combinatorial problems where the number base fluctuates throughout the processes. Though it may be simple to analyze the entropy of these games for one radix value, it is a much harder problem to understand entropy loss when the radix constantly changes, which is common in games. For example, researchers utilize the combinatorial number system to analyze lottery games, in which strictly decreasing combinations of numbers are advantageous to winning [51]. Another example occurs in video games, in which mixed-radix indexing processes are used to denote the position of different players [52]. Mixed radix is also necessary for combinatorial problems and simulations. For example, the factorial number system is a mixed-radix system that is utilized to analyze combinatorial problems in which permutations are represented as numbers [53]. It is important to convert formulae into input different radix values in these scenarios, so that others can understand the entropy loss of these games and problems. This paper highlights entropy loss in a card-shuffling system that can be applied to casino games: the index representing deck-position changes during every iteration of the system, so it is important to be able to calculate the entropy loss of the system with different radices.

Lastly, nonbinary and mixed-radix number systems are utilized in many scenarios that encompass even simple applications, such as representing metrics such as time and length. Utilizing nonbinary PRNG can help provide uniformity in generating random values for these applications. Utilizing RNS itself is also advantageous in parallel computing and fast arithmetic, which has applications in encryption, nanoelectronics, digital-image processing, and embedded computing. Our claim of Section 3.1 may be used to calculate the entropy loss of many of these mixed-radix and nonbinary applications, especially in games and combinatorial problems, because it is common for the number base to fluctuate throughout these processes. Equation (15), which quantifies the entropy loss of a surjective map $\mathcal{A} \to \mathcal{B}$, where $|\mathcal{A}| = 2^J$ and $|\mathcal{B}| = n$ for an arbitrary value of J, is important in that it takes into account radix n. Thus, Shannon entropy loss can be calculated for multiple radix values in applications that utilize this mapping process.

3.4. Secondary Impact

In this section, we discuss parameters that influence our choice of J in testing PRNGs for the casino application. First, we must consider the impact of keeping J constant as

n varies, or *mixing* J values, such that we optimally select J, $J(n)$, within an allowable range to minimize entropy loss for each n. By the approximation of Section 3.1, entropy loss is monotonically decreasing in terms of J. We want J to be as large as possible, and because of the monotonic nature of entropy loss, there is no apparent benefit in mixing J values. Therefore, we base our choice of J on the maximum that the processor can handle, regardless of current index value n.

Next, we consider hardware in choosing which J value to use in the simulations. In other words, we configure the choice of J to use available resources. Due to currently available standard processor sizes, the choices for J are restricted to $J = \{8, 16, 32, 64, 128\}$. This is justifiable due to the monotonicity of entropy of $\gamma_{A \to B}$ when $A = 2^J$. For example, if $J = 30$, increasing this value to $J = 32$ does not add any additional cost to the solution and would still result in lower entropy loss.

The fundamental hardware consideration in choosing the optimal J value depends on the processor size. The amount of arithmetic in the modulo process must not be too excessive for the m-bit choice of processor. The primary computation that would cause a large amount of arithmetic in the shuffle is computing a large J-bit value modulo n, where $n \leq 52$, on a m-bit processor when $m < J$. In the following paragraphs, we show that exceeding the processor constraints is not a realistic possibility for the shuffler, since many processors decompose large operations using bit-slice processes on a base-2^m number system [54,55].

In the bit-slice process, the J-bit word X is divided into $\frac{J}{2^m}$ smaller m-bit subwords $X_{\frac{J}{2^m}-1}, \ldots, X_1, X_0$. In our adaptation of a bit-slice processor [56], these subwords are used to compute $X \pmod{n}$ by Equation (23):

$$X = \left[\sum_{i=0}^{\frac{J}{2^m}-1} \left(X_i \pmod{n} \cdot 2^{m \cdot i} \pmod{n} \right) \right] \pmod{n}. \tag{23}$$

Each power of $2^{m \cdot i}$ modulo n can be precalculated and stored in memory to reduce the number of additional calculations. Figure 5 shows how an m-bit processor calculates Equation (23).

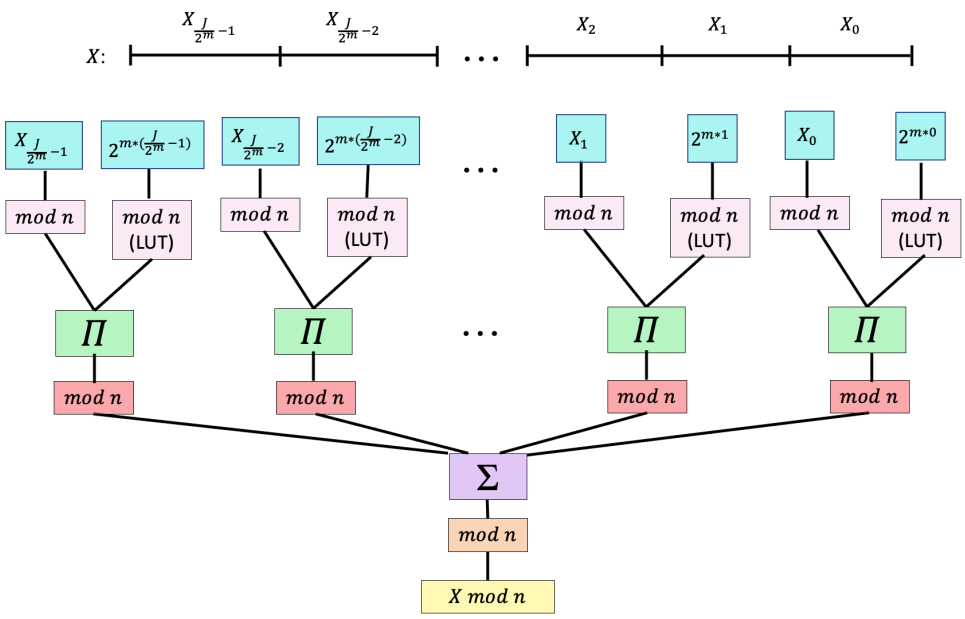

Figure 5. Bit-slice process of decomposing the calculation of J-bit binary word modulo n on a m-bit processor.

Since the J-bit word X is broken down into m-bit fractional elements, a small processor can compute each subword mod n to create a 6-bit or less output for each subword. Thus, even if $J = 128$, which far exceeds entropy-loss expectations, a 32-bit processor suffices since it can decompose the word into four subwords. Important in this modular reduction is the recognition that subwords have virtually no pairwise calculations, making them efficient on virtually any reduced precision process.

Combining these secondary-impact types, we utilized values J equal to or larger than processor size m in our application for $m \geq 32$. If a smaller processor size was utilized, we used $J = 32$ or higher. Table 1 displays the minimal value of J that is recommended for each common processor size.

Table 1. Choice of J for card-shuffling implementations based on processor size.

Processor Size m	Minimal Choice of J
8	32
16	64
32	128
64	128
128	128

3.5. Prototype Shuffling Algorithm

We used MATLAB to simulate the shuffling algorithm using the PRNGs listed in Section 2.2. The code that we utilized is outlined in Algorithm 1. Inputs to this algorithm include k and α as described in Section 2. The final input is *array*, which is an ordered set that can be mapped from a deck of 52 cards. This deck does not have to be in the order of a standard deck, yet we assumed a fresh deck between iterations in our experiment.

Block 1 creates a look-up table of precalculated values $2^i \mod n$ for $i = 1 : \alpha$ and $n = 1 : 52$. These calculations utilize the bit-slice process described in Figure 5. Block 2 then calculates the random numbers by scaling PRNG value r with its corresponding value from M. Lastly, Block 3 implements the Fisher–Yates shuffle on *array* utilizing the random numbers from X. The output is a shuffled *array* that represents the final shuffled deck.

Algorithm 1: Shuffler Algorithm.

```
Input: k, α, array
J ← k · α, X ← array of size 52, r ← array of 52 zeros, M ← 52 × α matrix of zeros,
  n ← 1.;
[Block 1]
for n =1 to 52 do
    for i to α do
        M ← values of 2^i mod n
        n++, i++
    end
end
[Block 2]
for n = 1 to 52 do
    Generate α random numbers into r
    Calculate X(n) using M and r.
    n++, i++
end
[Block 3]
for n = 1 to 52 do
    Perform Fisher–Yates shuffle on array using X
    n++
end
Output: array (shuffled)
```

After simulating the shuffling algorithm with different PRNGs, we used a poker-hand ranker to evaluate the output shuffle quality [57]. Although not an explicit measure of security or goodness, this practical application of the shuffling algorithm enables testing the results beyond esoteric entropy numbers. The numbers of poker hands expected and obtained in a 10 million hand run, each consisting of an independent shuffle and dealing of the first 5 cards, are displayed in Figure 6. The calculated expected frequencies show that, even when the algorithm relies on moderate PRNGs, the output values are still randomized; weak PRNGs such as the LFSRs display an observable deviation from combinatorics-based expected probabilities. A TRNG may be used in lieu of any of these methods, providing greater assurance of maximal entropy.

Poker Hand		Pseudorandom Number Generator						
	Expected	mt19937ar	mrg32k3a	mlfg6331_64	mcg16807	swb2712	LFSR-16	LFSR-24
Straight flush	153.9	149	146	156	151	155	0	611
Straight	39,246.5	39,261	39,355	39,159	39,351	39,172	40,897	46,987
Four of a kind	2401.0	2389	2400	2397	2395	2374	1221	1222
Full house	14,405.8	14,220	14,387	14,382	14,558	14,301	4883	7324
Flush	19,654.0	19,706	19,417	19,514	19,730	19,926	59814	56765
Three of a kind	211,284.5	212,045	211,264	212,396	211,633	210,891	156260	147093
Two pair	475,390	474,767	475,219	474,461	476,083	474,606	391,231	406,509
Pair	4,225,690	4,226,512	4,223,350	4,225,677	4,223,706	4,225,301	4,042,922	4,083,281
High card	5011774	5,010,951	5,014,462	5,011,858	5,012,393	5,013,274	5,302,772	5250208

Figure 6. Number of events of 9 poker hands out of 10 million runs for each tested PRNG.

4. Discussion

Due to the popularity of mechanical card shufflers in casinos, there is interest in creating real-time shuffling implementations utilizing lightweight primitives like PRNGs. We introduced a card-shuffling algorithm that, on the basis of RNG choice, can be low-power and cost-effective. We designed an experiment to test PRNGs of differing quality to determine how to optimize cost effectiveness and output shuffle quality.

Entropy is lost from this algorithm in two ways: through the onto-mapping process in modulo operations and through the selected PRNG. As an extension to the bound created by [22] for entropy loss resulting from the onto-mapping process, we calculated a more precise formula that describes the exact entropy loss. In particular, we quantified the Shannon entropy of surjective mappings $\mathcal{A} \to \mathcal{B}$ where $|\mathcal{A}| = A$ and $|\mathcal{B}| = B$. After computing a generic formula for arbitrary domain sizes, we refined the formula for $A = 2^J$. We utilized these formulas to prove the monotonicity of the onto map for $A = 2^J$ and reasonable bounds on n. We created deterministic formulas and proven properties of the Shannon entropy of the onto map from $2^J \to n$. The motivation behind this proof was for choosing a suitable testing value of J in order to the functionality of different PRNGs in a casino shuffling algorithm.

We set up and proved the optimal parameters for testing to minimize entropy loss from the random-number-generation process in the algorithm. Then, we compared different PRNGs in the RNG process and examined the effect of using a low-power PRNG on the entropy of the combined operation by examining the frequencies of poker hands out of 10 million runs for each PRNG, listed in Figure 6. The resulting frequencies demonstrated that even low-power PRNGs are able to produce a suitable amount of randomness for a casino shuffling application, though we also showed that not all PRNGs make the cut.

Future work is building a hardware prototype of this shuffling engine, including the incorporation of a TRNG. Applying poker-hand analysis with TRNG is expected to provide long-term probability values for comparison that can truly highlight the differences in utilizing PRNG in lieu of TRNG for this application. Moreover, the prototype is suitable for immediate incorporation in electronic games (e.g., video poker and slots), which may require different number bases and adaptation to mechanical shufflers.

Author Contributions: Conceptualization, A.V. and A.M.; methodology, A.V. and A.M.; software, A.V. ; writing—original-draft preparation, A.V. and A.M.; writing—review and editing, A.M.; visualization, A.V. and A.M.; supervision, A.M. Both authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: This study did not report any data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Omondi, A.; Premkumar, B. *Residue Number Systems: Theory and Implementation*; Imperial College Press: London, UK, 2007.
2. Bajard, J.; Imbert, L. A full RNS implementation of RSA. *IEEE Trans. Comput.* **2004**, *53*, 769–774. [CrossRef]
3. Molahosseini, A.S.; Sorouri, S.; Zarandi, A.A.E. Research challenges in next-generation residue number system architectures. In Proceedings of the 2012 7th International Conference on Computer Science Education (ICCSE), Melbourne, Australia, 14–17 July 2012; pp. 1658–1661. [CrossRef]
4. Xiao, H.; Garg, H.; Hu, J.; Xiao, G. New Error Control Algorithms for Residue Number System Codes. *ETRI J.* **2015**, *38*. [CrossRef]
5. Xiao, H.; Ye, Y.; Xiao, G.; Kang, Q. Algorithms for comparison in residue number systems. In Proceedings of the 2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA), Jeju, Korea, 13–15 December 2016; pp. 1–6. [CrossRef]
6. Bi, S.; Gross, W.J. The Mixed-Radix Chinese Remainder Theorem and Its Applications to Residue Comparison. *IEEE Trans. Comput.* **2008**, *57*, 1624–1632. [CrossRef]
7. Rafiev, A.; Murphy, J.P.; Yakovlev, A. Quaternary Reed-Muller Expansions of Mixed Radix Arguments in Cryptographic Circuits. In Proceedings of the 2009 39th International Symposium on Multiple-Valued Logic, Naha, Japan, 21–23 May 2009; pp. 370–376. [CrossRef]
8. Rafiev, A.; Mokhov, A.; Burns, F.; Murphy, J.; Koelmans, A.; Yakovlev, A. Mixed radix Reed-Muller expansions. *IEEE Trans. Comput.* **2012**, *61*, 1189–1202. [CrossRef]
9. Michaels, A.; Chester, D. Mixed Radix Number Generator with Chosen Statistical Artifacts. U.S. Patent No. 7,962,540, 14 June 2011.
10. Huang, C. A Fully Parallel Mixed-Radix Conversion Algorithm for Residue Number Applications. *IEEE Trans. Comput.* **1983**, *C-32*, 398–402. [CrossRef]
11. Wang, L.; Zhou, X.; Sobelman, G.E.; Liu, R. Generic Mixed-Radix FFT Pruning. *IEEE Signal Process. Lett.* **2012**, *19*, 167–170. [CrossRef]
12. Kaur, R.; Singh, T. Design of 32-point mixed radix FFT processor using CSD multiplier. In Proceedings of the 2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC), Solan, India, 22–24 December 2016; pp. 538–543. [CrossRef]
13. Yin, X.; Zhang, A.; Zhang, H.; Hao, L. Research and design of digital power quality analysis system based on mixed radix FFT. In Proceedings of the 33rd Chinese Control Conference, Nanjing, China, 28–30 June 2014; pp. 7393–7399. [CrossRef]
14. Zhizhe, L.; Shun'an, Z. A reconfigurable and high precision VLSI architecture for Fast Fourier Transform. In Proceedings of the 2010 2nd International Conference on Computer Engineering and Technology, Chengdu, China, 16–18 April 2010; Volume 4, pp. V4-420–V4-423. [CrossRef]
15. Lin, S.; Chung, W. The split-radix Fast Fourier transforms with radix-4 butterfly units. In Proceedings of the 2013 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference, Kaohsiung, Taiwan, 29 October–1 November 2013; pp. 1–5. [CrossRef]
16. Moon, Y.-J.; Kim, Y.-I. A mixed-radix 4-2 butterfly with simple bit reversing for ordering the output sequences. In Proceedings of the 2006 8th International Conference Advanced Communication Technology, Phoenix Park, Korea, 20–22 February 2006; Volume 3, p. 1774. [CrossRef]
17. Shirbhate, R.; Panse, T.; Ralekar, C. Design of parallel FFT architecture using Cooley Tukey algorithm. In Proceedings of the 2015 International Conference on Communications and Signal Processing (ICCSP), Melmaruvathur, India, 2–4 April 2015; pp. 0574–0578. [CrossRef]
18. Pasqualini, L.; Parton, M. Pseudo Random Number Generation: A Reinforcement Learning approach. *Procedia Comput. Sci.* **2020**, *170*, 1122–1127. [CrossRef]
19. Instruments, T. What's a LFSR? Available online: https://www.ti.com/lit/an/scta036a/scta036a.pdf (accessed on 20 March 2021).
20. Tian, X.; Benkrid, K. Mersenne Twister Random Number Generation on FPGA, CPU and GPU. In Proceedings of the 2009 NASA/ESA Conference on Adaptive Hardware and Systems, San Francisco, CA, USA, 29 July–1 August 2009; pp. 460–464. [CrossRef]

21. Wang, K.; Cao, Y.; Chang, C.; Ji, X. High-Speed True Random Number Generator Based on Differential Current Starved Ring Oscillators with Improved Thermal Stability. In Proceedings of the 2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 26–29 May 2019; pp. 1–5. [CrossRef]
22. Michaels, A.J.; Lau, C.C. Quantization Effects in Digital Chaotic Communication Systems. In Proceedings of the MILCOM 2013-2013 IEEE Military Communications Conference, San Diego, CA, USA, 18–20 November 2013; pp. 1564–1569. [CrossRef]
23. Diaconis, P.; Fulman, J.; Holmes, S. Analysis of Casino Shelf Shuffling Machines. *Ann. Appl. Probab.* **2013**, *23*, 1692–1720. [CrossRef]
24. Reynolds, C.; Ishikawa, M. Robot Trickery. Available online: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.147.6292 (accessed on 20 March 2021).
25. Silverman, M. Progressive Randomization of a Deck of Playing Cards: Experimental Tests and Statistical Analysis of the Riffle Shuffle. *Open J. Stat.* **2019**, *9*, 268. [CrossRef]
26. Aldous, D.; Diaconis, P. Shuffling Cards and Stopping Times. *Am. Math. Mon.* **1986**, *93*, 333–348. [CrossRef]
27. Stark, D.; Ganesh, A.; O'Connell, N. Information Loss in Riffle Shuffling. *Comb. Probab. Comput.* **2002**, *11*, 79–95. [CrossRef]
28. Marsaglia, G. *DIEHARD: A Battery of Tests of Randomness*; Florida State University: Tallahassee, FL, USA, 1995.
29. iTech Labs. Random Number Generator. Available online: https://www.kamagames.com/files/docs/RNG_Certificate_KG_UK_21Apr20.pdf (accessed on 20 March 2021).
30. Lalley, S. Rifle Shuffles and Their Associated Dynamical Systems. *J. Theor. Probab.* **1999**, 903–932.:1021636902356. [CrossRef]
31. Fisher, R.; Yates, F. *Statistical Tables for Biological, Agricultural, and Medical Research*; Oliver and Boyd: Edinburgh, UK, 1948.
32. Li, C.C.; Sun, B. Using Linear Congruential Generators for Cryptographic Purposes. In Proceedings of the 20th International Conference on Computers and Their Applications, Honolulu, HI, USA, 18–20 March 2005.
33. Documentation, M. RandStream.list. 2021. Available online: https://www.mathworks.com/help/matlab/ref/randstream.randstream.list.html (accessed on 20 March 2021).
34. Ankur; Divyanjali; Bhardwaj, T. A dissection of pseudorandom number generators. In Proceedings of the 2015 2nd International Conference on Signal Processing and Integrated Networks (SPIN), Noida, India, 19–20 February 2015; pp. 318–323. [CrossRef]
35. Matsumoto, M.; Nishimura, T. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Trans. Model. Comput. Simul.* **1998**, *8*, 3–30. [CrossRef]
36. Thomson, W.E. A Modified Congruence Method of Generating Pseudo-random Numbers. *Comput. J.* **1958**, *1*, 83. [CrossRef]
37. Steele, G.; Vigna, S. Computationally easy, spectrally good multipliers for congruential pseudorandom number generators. *arXiv* **2020**, arXiv:abs/2001.05304.
38. Marsaglia, G.; Zaman, A. A New Class of Random Number Generators. *Ann. Appl. Probab.* **1991**, *1*, 462–480. [CrossRef]
39. Lüscher, M. A portable high-quality random number generator for lattice field theory simulations. *Comput. Phys. Commun.* **1994**, *79*, 100–110. [CrossRef]
40. Villarino, M.B. The Error in an Alternating Series. *Am. Math. Mon.* **2018**, *125*, 360–364. [CrossRef]
41. Li, Z.; Ma, W. An AI problem: Knot-unknotting Chinese puzzle ring by the counting law of Gray code. In Proceedings of the 3rd World Congress on Intelligent Control and Automation (Cat. No.00EX393), Hefei, China, 26 June–2 July 2000; Volume 1, pp. 304–307. [CrossRef]
42. Gao, J.; Yang, F.; Ma, X. Indoor positioning system based on visible light communication with Gray-coded identification. In Proceedings of the 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), Valencia, Spain, 26–30 June 2017; pp. 899–903. [CrossRef]
43. Tyagi, A.; Verma, A.; Saxena, A. Optimal economic dispatch considering wind energy conversion systems using Gray coded genetic algorithm. In Proceedings of the 2015 Annual IEEE India Conference (INDICON), New Delhi, India, 17–20 December 2015; pp. 1–5. [CrossRef]
44. Jamil, T. An Introduction to Complex Binary Number System. In Proceedings of the 2011 Fourth International Conference on Information and Computing, Phuket, Thailand, 25–27 April 2011; pp. 229–232. [CrossRef]
45. Jamil, T. Complex Binary Associative Dataflow Processor-A Tutorial. In Proceedings of the SoutheastCon 2018, St. Petersburg, FL, USA, 19–22 April 2018; pp. 1–3. [CrossRef]
46. Plantz, A.; Berman, M. Adoption of the Octal Number System. *IEEE Trans. Comput.* **1971**, *C-20*, 593–598. [CrossRef]
47. Vincent, R.; Anju, S.L. Decimal floating point format based on commonly used precision for embedded system applications. In Proceedings of the 2013 Annual International Conference on Emerging Research Areas and 2013 International Conference on Microelectronics, Communications and Renewable Energy, Kanjirapally, India, 4–6 June 2013; pp. 1–4. [CrossRef]
48. Levachkine, S.; Velazquez, A.; Alexandrov, V. Color image segmentation using false colors and its applications to geo-images treatment: Alphanumeric character recognition. In Proceedings of the IGARSS 2001. Scanning the Present and Resolving the Future. Proceedings. IEEE 2001 International Geoscience and Remote Sensing Symposium (Cat. No.01CH37217), Sydney, NSW, Australia, 9–13 July 2001; Volume 5, pp. 2212–2214. [CrossRef]
49. Ivanov, M.; Roslyj, E.; Starikovskiy, A.; Krasnikova, S.; Shevchenko, N.; Shustova, L. Non-Binary Pseudorandom Number Generators For Information Security Purposes. *Procedia Comput. Sci.* **2018**, *123*, 203–211. [CrossRef]
50. The Art of Computer Programming: Positional Number Systems, 2014. Available online: https://www.informit.com/articles/article.aspx?p=2221791 (accessed on 23 April 2021).
51. Knuth, D. *Bitwise Tricks and Techniques*; Addison-Wesley: Boston, MA, USA, 2020; p. 54.

52. Glenn, J.; Fang, H.; Kruskal, C. A Retrograde Approximation Algorithm for Multi-player can't stop. In Proceedings of the Computers and Games: 6th International Conference, CG 2008, Beijing, China, 29 September–1 October 2008.
53. Knuth, D. *Volume 2: Seminumerical Algorithms, The Art of Computer Programming*; Addison-Wesley: Boston, MA, USA, p. 192. ISBN 0-201-89684-2.
54. Gordon, D.M. A Survey of Fast Exponentiation Methods. *J. Algorithms* **1998**, *27*, 129–146. [CrossRef]
55. Gueron, S. Efficient Software Implementations of Modular Exponentiation. *IACR Cryptol. EPrint Arch.* **2011**, *2011*, 239. [CrossRef]
56. Tang, G.M.; Qu, P.Y.; Ye, X.C.; Fan, D.R. Logic Design of a 16-bit Bit-Slice Arithmetic Logic Unit for 32-/64-bit RSFQ Microprocessors. *IEEE Trans. Appl. Supercond.* **2018**, *28*, 1–5. [CrossRef]
57. Slazas, R. Poker hand ranker. *MathWorks File Exchange*. Available online: https://www.mathworks.com/matlabcentral/fileexchange/17579-poker-hand-ranker (accessed on 20 March 2021).