Security of Lightweight Cryptographic Primitives

Amy D. Vennos

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Mathematics

Alan Michaels, Co-chair
Gretchen Matthews, Co-chair
Daniel Orr

May 11, 2021
Blacksburg, Virginia

Keywords: Pseudorandom Number Generator, Residue Number System, Reverse Engineering, Shannon Entropy, Mixed-Radix Conversion

Security of Lightweight Cryptographic Primitives

Amy D. Vennos

(ABSTRACT)

Internet-of-Things (IoT) devices are increasing in popularity due to their ability to help automate many aspects of daily life while performing these necessary duties on billions of low-power appliances. However, the perks of these small devices also come with additional constraints to security. Security always has been an issue with the rise of cryptographic backdoors and hackers reverse engineering the security protocols within devices to reveal the original state that was encrypted. Security researchers have done much work to prevent attacks with high power algorithms, such as the international effort to develop the current Advanced Encryption Standard (AES). Unfortunately, IoT devices do not typically have the computational resources to implement high-power algorithms such as AES, and must rely on lightweight primitives such as pseudorandom number generators, or PRNGs. This thesis explores the effectiveness, functionality, and use of PRNGs in different applications. First, this thesis investigates the confidentiality of a single-stage residue number system PRNG, which has previously been shown to provide extremely high quality outputs for simulation and digital communication applications when evaluated through traditional techniques like the battery of statistical tests used in the NIST Random Number Generation and DIEHARD test suites or in using Shannon entropy metrics. In contrast, rather than blindly performing statistical analyses on the outputs of the single-stage RNS PRNG, this thesis provides both white box and black box analyses that facilitate reverse engineering of the underlying RNS number generation algorithm to obtain the residues, or equivalently the key, of the RNS algorithm. This thesis develops and demonstrate a conditional entropy analysis that permits

extraction of the key given a priori knowledge of state transitions as well as reverse engineering of the RNS PRNG algorithm and parameters (but not the key) in problems where the multiplicative RNS characteristic is too large to obtain a priori state transitions. This thesis then discusses multiple defenses and perturbations for the RNS system that defeat the original attack algorithm, including deliberate noise injection and code hopping. We present a modification to the algorithm that accounts for deliberate noise, but rapidly increases the search space and complexity. Lastly, a comparison of memory requirements and time required for the attacker and defender to maintain these defenses is presented.

The next application of PRNGs is in building a translation for binary PRNGs to non-binary uses like card shuffling in a casino. This thesis explores a shuffler algorithm that utilizes RNS in Fisher-Yates shuffles, and that calls for inputs from any PRNG. Entropy is lost through this algorithm by the use of PRNG in lieu of TRNG and by its RNS component: a surjective mapping from a large domain of size 2^J to a substantially smaller set of arbitrary size n. Previous research on the specific RNS mapping process had developed a lower bound on the Shannon entropy loss from such a mapping, but this bound eliminates the mixed-radix component of the original formulation. This thesis calculates a more precise formula which takes into account the radix, n. This formulation is later used to specify the optimal parameters to simulate the shuffler with different test PRNGs. After implementing the shuffler with PRNGs with varying output entropies, the thesis examines the output value frequencies to discuss if utilizing PRNG is a feasible alternative for casinos to the higher-cost TRNG.

Security of Lightweight Cryptographic Primitives

Amy D. Vennos

(GENERAL AUDIENCE ABSTRACT)

Cryptography, or the encrypting of data, has drawn widespread interest for years, initially sparking public concern through headlines and dramatized reenactments of hackers targeting security protocols. Previous cryptographic research commonly focused on developing the quickest, most secure ways to encrypt information on high-power computers. However, as wireless low-power devices such as smart home, security sensors, and learning thermostats gain popularity in ordinary life, interest is rising in protecting information being sent between devices that don't necessarily have the power and capabilities as those in a government facility. Lightweight primitives, the algorithms used to encrypt information between low-power devices, are one solution to this concern, though they are more susceptible to attackers who wish to reverse engineer the encrypting process. The pesudorandom number generator (PRNG) is a type of lightweight primitive that generates numbers that are essentially random even though it is possible to determine the input value, or seed, from the resulting output values. This thesis explores the effectiveness and functionality of PRNGs in different applications. First, this thesis explores a PRNG that has passed many statistical tests to prove its output values are random enough for certain applications. This project analyzes the quality of this PRNG through a new lens: its resistance to reverse engineering attacks. The thesis describes and implements an attack on the PRNG that allows an individual to reverse engineer the initial seed. The thesis then changes perspective from attacker to designer and develop defenses to this attack: by slightly modifying the algorithm, the designer can ensure that the reverse engineering process is so complex, time-consuming, and memory-requiring

that implementing such an attack would be impractical for an attacker. The next application of PRNGs is in the casino industry, in which low-power and cost-effective automatic card shufflers for games like poker are becoming popular. This thesis explores a solution for optimal shuffling of a deck of cards.

Dedication

To my parents, brother, and fiancé.

Acknowledgments

I would like to thank my advisor, Dr. Alan Michaels for his guidance and patience. Throughout the past two years, he has challenged me to becoming a better researcher and mathematician. I would also like to thank Dr. Gretchen Matthews and Dr. Daniel Orr for extending their support in my courses and research.

I would like to thank my parents, Andrew and Jenny Vennos, and my brother, Alex Vennos. I am grateful for NiCole Nardella, Garrett Fowler, and Brittany Boss for their friendship and encouragement. Finally, thank you to my wonderful fiancé, Ryan Shifler, and future mother-in-law, Beth Shifler, for helping me feel at home while writing and conducting research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Cryptography encompasses the techniques used to obscure information, and encryption is required in many aspects of the modern-day world. For example, digital currency is a growing discussion due to the advantages of simple, virtual, and fast monetary exchanges. However, hacking remains a large risk to cryptocurrency, and researchers are analyzing digital money to investigate its safety for widespread use [1]. The automotive industry also relies on cryptographic algorithms to ensure that intra-vehicular data exchanges, which are usually sent via radio signal, is done through secure channels. Security is becoming an even larger issue in this industry due to the incorporation of artificial intelligence into vehicular systems [2]. The reliance of email in daily life, consumer exchanges, and professional scenarios additionally inspire high power techniques to ensure that electronic mail is sent securely [3].

Conventional cryptography standards have been extensively developed and studied in the past twenty years to address the demand for data security in modern applications. Well known algorithms, such as Rijndael [4] and Rivest-Shamir-Adleman (RSA) [5], are used worldwide and are well known due to their accepted high levels of security. In 1997, Rijndael in particular was chosen by the National Institute of Standards and Technology (NIST) as the Advanced Encryption Standard (AES) for its design and strength, and in 2003, it was announced as secure enough to protect classified information for the U.S. government

[4, 6]. Other recent developments in cryptography include Elliptic Curve Cryptography [7] and the Digital Signature Algorithm [8]. Protocols are also being designed with the future in mind. The development of Shor's algorithm [9] awakened the potential of quantum computing, which could be a threat to ECC and DSA. Researchers are already working on solutions to protecting the security of systems in the post-quantum scenario. Examples include strengthening security methods in the automotive [10] and electronic currency [11] industries.

Unfortunately, standard cryptographic protocols can be quite costly in computation time, memory storage, and energy efficiency, as they were designed with security as the main priority. Cost is becoming an important parameter in data security due to the emergence of the Internet of Things (IoT) [12]. IoT continues to expand as demand increases for the production of technologies that provide connected networks of devices used in the day-to-day lives of consumers. The communication between IoT devices has become paramount in critical applications such as healthcare, smart appliances, and transportation [13, 14]. This correspondence mainly exists in the form of data transfer through insecure channels, and thus much work is done to protect the integrity and confidentiality of such transmissions. Furthermore, consumer demand for the newest technological developments in protecting personal data like credit card information and personal identification stored in smart devices has pushed advancements in data privacy and security [15]. IoT devices still require a reasonable level of security that balances computational complexity and confidentiality, and thus security no longer remains the only metric for consideration in designing a cryptographic algorithm. Lightweight cryptography has thus emerged as a new subfield of cryptography and refers to securing the data transfer in IoT applications that are limited in power consumption, size, and/or speed [16].

There are many recent and ongoing contributions in developing and testing lightweight

cryptography algorithms. In 2018, NIST announced a call for lightweight cryptography algorithms for a standardization process [17]. The submissions were judged on multiple criteria such as cost, performance, and resistance to side channel attacks. This process is ongoing - NIST hosted lightweight cryptography workshop in 2020 for candidates whose algorithms passed a first round that tested for vulnerabilities against several reverse engineering attacks [18]. Methods to protect IoT-related security are also being studied for specific devices, such as pacemakers [19], vehicle electronic control systems [20], and wireless security networks (WSNs) [21]. Whether it be for general standardization or for specialized applications, work in lightweight cryptography is necessary and in demand.

Concepts used to develop lightweight methods have other applications than data security. The residue number system (RNS) is a noteworthy example. RNS is advantageous in developing lightweight cryptographic protocols due to its capability to break down complex arithmetic so that complex calculations can be done quickly on low power devices. This characteristic has inspired incorporating RNS implementations in cryptographic processors [22]. Utilizng RNS in other applications are driven by the same incentive: to reduce the complexity of performing computationally complex tasks. Popular implementations of RNS for this purpose are ditigal filtering and image processing [23]. After discussing the security of an RNS-based cryptographic primitive, this thesis will discuss utilizing RNS and lightweight cryptographic primitives in a casino game application, outlined in the following section.

1.2 Scope

This thesis analyzes the effectiveness of lightweight cryptographic primitives and RNS with two objectives: security against reverse-engineering attacks and functionality in a card-shuffling algorithm. Chapter 2 discusses the background of these topics and motivates how

this thesis contributes to the current advancements in these fields. This chapter describes in more detail the specific contributions others have made in both conventional and lightweight cryptography and also provides specifics about the incorporation of RNS in cryptosystem implementations.

Chapter 3 describes a single-stage RNS pseudorandom number generator (PRNG), a cryptographic primitive that generates a deterministic sequence of pseudorandom numbers. This PRNG has possible applications in TRANSEC, cryptography, and other types of secure communications [24]. This project begins by implementing the PRNG on a toy example to facilitate the reader in understanding the algorithm. Then, we execute an attack that exploits the conditional probabilities of output values in an IoT-caliber example. Though this attack may imply this PRNG is useless in cryptographic implementations, we propose three defenses against this reverse-engineering method. These defenses increase the attacker's computational load, and we show that this increase is so immense that the attack is infeasible.

The content in chapter 3 is joint work conducted with a computer engineering graduate student, Kiernan George. While I worked on the mathematical side of describing the algorithm and process of the reverse engineering attack, Kiernan implemented the attack on the realistically-sized IoT example and analyzed two of the defenses against the reverse engineering process. Table 1.1 lists each section in Chapter 3 with the corresponding author(s).

In Chapter 4, this thesis focuses on RNS in a translation for PRNGs to applications such as card shuffling. Section 4.2 introduces a shuffler algorithm that relies on a sequence of uniformly distributed random inputs from a mixed-radix domain to eventually implement a Fisher-Yates [25] shuffle that utilizes inputs from a base-2 PRNG. Section 4.3 explores the entropy loss through the surjective mapping from a large domain of size 2^J to a set of a substantially smaller, yet arbitrary size n. The section then calculates a precise formula for

Section	K. George	A. Vennos
3.1	Contributor	Primary Author
3.2	N/A	Sole Author
3.3	N/A	Sole Author
3.4	Primary Author	Contributor
3.4.1	N/A	Sole Author
3.4.2	Sole Author	N/A
3.4.3	Sole Author	N/A
3.5	Joint Author	Joint Author
3.7	Contributor	Primary Author
3.8	Joint Author	Joint Author

Table 1.1: Division of work for the research presented in Chapter 3

the Shannon entropy loss of the mapping process that takes into account radix. This formula is utilized to prove the monotonicity of the entropy loss, which is helpful in specifying the optimal parameters to simulate the shuffling algorithm using test PRNGs. This simulation shows that this algorithm is consistent with low-power casino implementation and can be extended to explore solutions to PRNG usage in other non-binary and combinatorial problems.

1.3 Contributions

Conference Papers

- A. Vennos, A. Michaels, "Shannon Entropy Loss in Mixed-Radix Conversions." (*in review, Entropy, 2021*).

This paper summarizes the contributions described in Chapter 4. Included is a literature review of research in mixed-radix (MR) techniques and the incorporation of PRNGs in lightweight card shuffling implementations. The paper describes the calculation of the Shan-

non entorpy loss formula and provide the proof of its monotonicity. With this formula, this project explains the parameters that are important in choosing the optimal J value to simulate the algorithm with test PRNGs. Finally, the paper analyzes the effectiveness of the PRNGs to show that even low-power PRNGs have the ability to produce a suitable amount of randomness for a casino shuffling application.

Journal Papers

- K. George, A. J. Michaels, and A. Vennos, "Attacks and Defenses for Single-Stage Residue Number System PRNGs." (*in review, IoT 2021*).

This paper describes the contributions in Chapter 3. The paper includes a literature search on previous methods to quantify the quality of a PRNG, a description and implementations of a reverse-engineering attack on a single-stage RNS PRNG, and an analysis of three modifications to the PRNG to increase the complexity of the attack to the point where it is deemed ineffective.

Chapter 2

Background of Cryptographic Primitives

2.1 Cryptography and the AES

Securing the transfer of information from one source to another has remained a challenge for years: the earliest known forms of cryptography were in the tomb of Ancient Egyptian nobleman Khnumhotep II [26]. Military efforts also inspired encryption in Ancient Rome. One of the more extensively studied ciphers is the Caesar Cipher, a method that replaced the original, or *plaintext*, letter with another letter shifted down in the alphabet in the *ciphertext* [27]. Cryptanalysis began to bloom in the twentieth century for military purposes as well: a notable example is the breaking of German codes in Admiralty's Room 40 during the First World War [28]. Devices created to encrypt and decrypt messages began to develop starting in 1918 with Vernam's teleprinter [29]. Other notable examples of relying on mechanical devices to encrypt messages are the unbreakable one time pad [30] and the Enigma device [31] utilized to encode German messages during World War II. In the 1970's, IBM's cryptography group developed the Lucifer cipher [32] in response to customer demand for encryption. For the next few years, cryptographic algorithms were developed mainly for government applications due to the high cost of computers at the time. The Data Encryption Standard (DES) and Advanced Encrytpion Standard (AES), which will be described in more detail

later in this section, were developed for high-cost computing devices as well. However, with the rapid development of the Internet and other networks such as vehicular ad-hoc networks (VANET), modern-day researchers have been developing more modern methods to secure information on devices with limited processing capacity, which are described in the next section [33].

The purpose of encryption is to secure information from reverse-engineering attacks, which attempt to recreate the plaintext by analyzing the ciphertext. Every cipher has the potential to be reversed through brute-force, though most applications are so large that such an attack would require too much time or resources to be worthwhile. Thus, many attacks aim to reduce the search space of a brute-force attack. For example, the Caesar cipher can be broken through a frequency analysis of the letters, which provide insight to the more probable identities of the plaintext letters, considerably reducing the brute-force search space [34].

Attacks with malicious intentions are a threat to modern-day cryptosystems, and much work is done to ensure that a cipher is resistant to such attacks. In fact, the U.S. Department of Commerce initiated a program in 1972 to create the Data Encryption Standard in response to the Brooks Act, which required a new set of standards for Federal government computers [35]. Many attacks on DES have been published, the most notable being in 1999 by the Electronic Frontier Foundation and distributed.net. Their DES cracker was able to attain a DES key in 22 hours, a much quicker proposition given today's computing resources [36].

The AES is an algorithm established by the National Institute of Standards and Technology (NIST) to replace the Data Encryption Standard (DES) algorithm for encrypting information. The method is also known as Rijndael, as it was created by the Belgian cryptographers Vincent Rijmen and Joan Daemen to win the challenge NIST established in order to find the best algorithm for the AES [4]. To find a new and improved algorithm, NIST released a challenge to submit new algorithms that were more efficient and more secure than the

DES. Submissions were to allow key sizes of 128, 192, and 256 bits, which increased security compared to the 52-bit key of the DES. Also, entries were to operate on blocks of 128 input bits, and work on a variety of different hardware [37]. Fifteen candidates were considered in the contest, and they were judged in two rounds for their ability to resist any potential attacks, their efficiency in both computation and memory cost, and their simplicity. NIST also welcomed and encouraged the general public to comment on the algorithms, and the judging considered these suggestions in the first round of the competition [38]. Considering this factor alone, the five fastest algorithms were determined to be Crypton, RC6, Rigndael, Twofish, and MARS. However, after including other factors such as security, simplicity, and comments made by the general public, the algorithms that made it to round two were MARS, RC6, Rijndael, Serpent, and Twofish [39]. NIST also published summaries about the five finalists after the first round, which touches on both qualitative and quantitative aspects of the algorithms.

Rijndael was described to be a great overall candidate. The summary did not reveal any negatives associated with the Rijndael algorithm, unlike the rest of the summaries. However, there was nothing mentioned that made this algorithm stand out from the others. The algorithm Serpent was described as very secure but lacking in speed, as it proved considerably slow in the round one analyses. Twofish, like Serpent, was noted as both very secure and fast. However, its complex design caused some concern to some reviewers. Round one analyses described MARS as fast and secure, but this algorithm did not support many platforms. Finally, the algorithm RC6 was reported to be very simple in that it utilized rotating digits based on previous data in lieu of substitution tables. The algorithm also did not include security gaps and was very fast. However, RC6 showed lower performance on platforms that did not favor 32-bit variable rotations [40].

In round two, NIST was still interested in finding the algorithm that proved to be successful

consistently across multiple platforms. To do this, the judging team created the cycle count test, which measures the number of clock cycles that are necessary to encrypt a block of data or set up a key. Twelve values were calculated for each algorithm on multiple platforms. The judging team calculated the number of cycles required to create an encryption key, encrypt a block of data, create a decyption key, and to decrypt a block of data. From this analysis, Rijndael was the fastest algorithm in creating the encryption key and decryption key, with RC6 and MARS close behind. The slowest algorithm was predictably Serpent. Another interesting test implemented was on memory requirements, where MARS, Rijndael, and Serpent proved to have low ROM requirements, and Twofish displayed low RAM requirements. Additional analysis observed the algorithms' simplicity, security, word size, and software implementaitons. Statistical tests were also run based on the NIST Statistical Test Suite [41].

After this analysis was completed, there existed some debate on how the winners should be selected. Most stood for the case that mostly quantitative data should be considered in selecting a winner, but determining how much data from the tests should be considered in declaring a winner was difficult [42]. Another decision was if NIST should declare more than one winner. However, this idea was dismissed due to the possibility of technical issues that could result from using multiple algorithms [41]. After reviewing data from the two rounds, conferences, and public opinion, NIST declared Rijndael as the best overall algorithm as it consistently performed well in a wide variety of environments. The algorithm had low time and memory requirements, and it was one of the easiest algorithms to protect against attacks at the time [41]. Though attacks now exist for AES (such as side-channel attacks and biclique attacks[43, 44]), it is still deemed secure enough to encrypt top-secret information in the U.S. federal government [6, 45].

2.2 Lightweight Cryptography

Though secure in many implementations, AES is not a viable solution to every application. Hardware constraints may deem normal cryptographic algorithms such as AES too slow or too energy-consuming, and thus other algorithms must be considered. The Internet of Things (IoT), which leverages communication between physical objects without human interaction, is becoming increasingly popular with modern devices used in everyday applications such as smart devices and healthcare that require a minimal level of security [46].

For example, wireless sensor networks (WSNs) are small sensors used to record physical conditions like pressure, temperature, and audio. WSNs are used in a variety of applications that span from medical implants to devices used to prevent natural disasters [46, 47]. Another application of security solutions to low-cost devices are in smart devices, in which technology advancements are being made in securing stored personal information, such as credit card data and personal information. Due to threats like forgery and hacking, privacy thus is becoming a concern for the general public [48]. In these two scenarios of smart devices and WSNs, devices are limited in memory storage and spell out CPU power and are often implemented in remote areas, meaning that traditional security mechanisms are not suitable for these devices [49]. In response to the increasing need for information security in hardware-constrained devices such as the ones previously listed, the term *lightweight cryptography* describes algorithms and methods that optimize security in low-cost devices. In particular, *lightweight primitives* describe the building blocks in the methods to protect information on resource-constrained devices.

Securing resource-constrained devices is challenging due to the increased vulnerability of lightweight primitives to cryptographic backdoors, or attacks [50]. A popular example of such a backdoor was an attack in 2013 on the Dual Elliptic Curve Deterministic Random

Bit Generator, or, DUAL_EC_DRBG, which was deemed secure by NIST as CSPRNG [51, 52, 53, 54]. Backdoors have also targeted hashing algorithms: for example, Albertini et al. created a backdoor on SHA 1, a variation of the Secure Hashing Algorithm (SHA), which was designated as RFC 3174 standard by NIST [55]. Bannier and Filial also published a backdoor on the BEA-1 algorithm, a simpler version of AES [56].

There is an increased initiative for security researchers to study lightweight backdoors, and in 2018, NIST announced a call for algorithms for a lightweight cryptography standardization process [57]. The target devices that the candidate algorithms will be applied to are embedded systems and sensor networks, especially RFID tags such as the EPCGlobal Gen2 [58] and the ISO/IEC 18000-63 [59]. NIST focused on the following performance metrics to judge the candidates: power and energy consumption, throughput, and latency. Power and energy consumption is important due to nature of the target devices. Latency, or the time required to produce the output from the initial operation request, is significant for applications that require quick response times, such as automotive braking or steering. Throughput, which is the production rate of new outputs, is moderately necessary in many lightweight applications such as real-time RFID tag implementations [57, 60].

In the first round, 57 candidate algorithms were analyzed with their security, performance, and cost as the most important parameters. They were also tested for their resistance against side-channel attacks as well as forgery, length-extention, and distinguishing attacks. Undesirable properties such as reliance on methods with certain sliding properties were also recorded for each algorithm. 33 candidates were selected to proceed to a second round of testing [61]. Information about the second round and finalist selection is being processed and has not been published by NIST, but as of March 2021, the candidates for the final round are ASCON, Elephant, GIFT-COFB, Grain 128-AEAD, ISAP, Photon-Beetle, Romulus, Sparkle, Tiny Jambu, and Xoodyak [17, 18].

Lightweight cryptography is a new and important objective in security, and much work is being done to prevent attacks on lightweight primitives. This thesis will focus on the security and practical use of the pseudorandom number generator (PRNG), a specific type of lightweight primitive used as a source of randomness in cryptographic algorithms. There is a variety of PRNG designs, such as LFSR [62] and combining different types of cellular automata [63]. Some PRNGs are better suited for certain applications than others, with applications ranging from cryptography [64], steganography [65], and non-security areas such as Monte Carlo simulations [66] and lattice-based field theory [67]. Qualitative methods to determine PRNG quality have included the NIST statistical test suite utilized in the AES selection process [68] and the FSU DIEHARD suite [69]. PRNG quality can also be measured by their Kolmogorov [70] and Shannon [71] entropy losses. For PRNGs with security related applications, resistance to reverse engineering attacks are another method to determine their suitability for application. Section 3.1 provides a detailed background on PRNGs and how their quality is measured. Chapter 3 then focuses on the security of a specific PRNG that is based on residue number system arithmetic. Chapter 4 focuses on other applications of PRNG that have non-binary random processes. Many systems model these processes with binary computers, and it is important to understand losses in entropy with different number bases. Section 4.1 provides detailed background of non-binary random processes utilized for applications such as combinatorial problems and casino games as well as the methods used to determine the quality of these processes. The chapter then analyzes the suitability of several PRNGs, varying in statistical metrics of quality, to be utilized in a card-shuffling algorithm.

2.3 Provable Security vs. Efficiency

We elaborate on the term, "security," mentioned many times in this thesis. S. Goldwasser and S. Micali developed the concept of provable security in 1982, which greately impacted research in cryptography [72]. Provable security is defined as a process that involves a security definition detailing the goals and abilities of an attacker, a statement of assumptions about the system, and a proof that the system meets the security statement [73]. This method allows researchers to reduce the question of security to a simpler mathematical question. The computational complexity of the mathematical problem then implies the security of the system in question [74]. The term provable security may be misleading in the sense that an algorithm is provably secure relative only to the assumptions made. For example, RSA has provable security up to the factoring of primes, but if a computationally easy method of factoring primes is found, RSA becomes an easy problem to solve [5]. ElGamal encryption is provably secure relative to the discrete logarithm problem [75].

Depending on the motivations of those intending to use the algorithm, provable security may or may not be as important a concern as for other applications. Commercial uses of lightweight primitives such as PRNGs may not be interested in the provable security of a light bulb in a smart home. However, some of these applications may still require a higher level of security. Though casinos do not necessarily need to invest in the research and resources to implement provable security, their devices still need to be resistant to most attacks. A 2018 attack on a casino demonstrated this need when individuals were able to gain access to the wireless internet network through a fish-tank thermometer [76]. Thus, studying methods that prevent some attacks on systems or algorithms that provide some (but not true) randomness to structures is still useful and important as they raise the expense to an attacker to gain access. Chapter 3 focuses on the security of a PRNG that fits this description. Though the

focus is on improving its security, we are not claiming that this system is cryptographically secure. Though other algorithms can be secure up to a point, this PRNG is not made to be secure against every attack. However, it is a very good efficient tool to generate randomness, and this efficient is often more important than its provable security in contexts like IoT.

2.4 Non-Binary and Mixed-Radix Systems

The base-2, or binary, number system, is a notation which only utilizes the values "0" and "1." Binary systems are thus described to have a *radix* of 2. Reducing traditional base-10 numbers to a series of symbols, or bits, that can take one of two values is advantageous in many applications, most of which lie in computing since Boolean logic can easily be translated to binary. Much research has shown how binary is advantageous in computing applications due to the circuity utilized in computer processing [77, 78, 79]. Binary is also used in computing for its assistance in memory storage. Representing information as zeroes and ones allows digital data to be stored more easily. As more information, such as film, geographical models, music, and healthcare imaging data, is required to be stored digitally, the binary number system allows important information to be encoded in a compact manner [80, 81, 82].

Binary is not utilized in every application, and other number systems have been established for use in other implementations. For example, researchers have developed slight modifications of the binary system. Gray code, or reflected binary, is applied in puzzles [83], positioning technology [84], and genetic algorithms [85]. Another important modification of binary is the Complex Binary Number System, which allows encoding of complex numbers as a solitary unit, granting faster processing for problems that handle complex numbers [86, 87]. There are also many physical processes that are not necessarily optimized with the radix 2.

Research has been done to examine the advantages of adopting the octal number system to represent SI units, money, time, and calendar days for computer accessibility [88]. Other systems include the decimal number system, which is often utilized when high precision is necessary [89], and the Alphanumeric number system, which is commonly utilized in storing colors using the RGB color model [90].

Some processes will require a non-binary source of randomness. PRNGs are being designed to utilize non-binary operations, such as the non-binary Galois linear feedback shift register (LFSR), in which the exclusive-or performs addition modulo-q instead of modulo-2 [91]. In fact, some processes will 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 either is the drawback of an entropy loss that is not truly uniform from relying on a small PRNG, or the disadvantage of extra processing while utilizing a larger PRNG. Employing mixed-radix and non-binary number systems are 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, dimes, and so forth [92]. Generating a random number of seconds within a minute would not provide uniformity if starting with a binary PRNG, this is no longer the issue if another number generator that was the size of 64 bits was used.

Other applications of mixed-radix and non-binary 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 the 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 advan-

tageous to winning [93]. Another example occurs in video games, in which mixed radix indexing processes are used to denote the position of different players [94]. 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 [95]. It is important to convert formulae to input different radix values in these scenarios so that others can understand the entropy loss of these games and problems. This thesis discusses the entropy loss in a card shuffling system that can be applied to casino games. The index representing deck position changes durig every iteration of the system, so it is important to be able to calculate the entropy loss of the system with different radices. Chapter 4 describes this system in more detail and motivates the importance of mixed-radix conversions in card shuffling.

2.5 The Residue Number System

The residue number system (RNS) is a system in which integers are represented by their values modulo coprime integers. Sun Tsu first proposed the concept of RNS in first century A.D., which later became known as the Chinese Remainder Theorem [96]. Euler later proved this theorem in 1734 [97]. Following this proof, RNS research did not blossom until Lehmer et. al. discovered the advantages of utilizing RNS in hardware that implemented computationally complex arithmetic [98]. Utilizing RNS is advantageous in these implementations due to its ability to reduce computational burden by breaking down calculations so that they can be done with high speed on low power devices, inspiring research in RNS applications. M. Soderstrand and K. Nelson applied RNS for digital heterodyne filters [99]. G. Jullien studied moduli selection for various RNS applications and developed the modified quadratic RNS [100, 101]. Jullien also collaborated with W. Miller to apply RNS to digital

signal processing (DSP) applications. In particular, RNS has been shownw to improve the running time of the Cooley-Tukey Fast Fourier Transform (FFT), which is the one of the most common examples in DSP [102]. Additionally, Jullien and Miller later worked with Bayoumi to develop the concept of lookup tables for RNS implementations [103].

RNS made its way into cryptography in the 1990s, when Posch et. al. designed an embedding of RNS in the modular multiplication commonly used in cryptosystems [22]. RNS implementations in cryptographic processors later flourished with the Cox-Rower architecture to parallelize Montgomery multiplication in 2000 [104], giving rise to a substantial increase in research in designing efficient cryptosystems as well as in Montgomery multiplication. Examples of RNS implementations include Canonic Signed Digit (CSD) encoding, which is used to reduce processing requirements and to accelerate computations [105]. Additionally, RNS multipliers are utilized in Modified Booth Encoding to improve the efficiency of computations in real-time calculation applications [106]. Other examples of RNS applications include implementing RNS in Elliptic Curve Cryptography [107] and in RSA cryptography [108, 109, 110, 111]. Additionally, RNS is applied in simulations such as image encryption techniques proposed using MATLAB [112], which only needs to have good statistical properties. On the other hand, applications in transmission security (TRANSEC), such as Digital Chaotic Communications [113] and the intentional addition o fnoise in spread spectrum signals [114], benefit from the presence of noise. In Chapter 3, we analyze on a RNS-based PRNG that incorporates a modified version of the CRT that can be implemented in lightweight cryptography applications. We then shift our focus from lightweight cryptography to RNS utilized in other applications.

The advantage of utilizing RNS in parallel computing and fast arithmetic can be applied in more areas than just encryption. Researchers are considering relying on RNS in developing future technologies such as nanoelectronics, in which RNS may be incorporated into the

structure of the quantum-dot cellular automata [115]. RNS has also played a core part in designing FGPA devices for digital image processing applications [116, 117]. Other applications extend to reversible watermarking [118], adaptive equalizers [119], and embedded computing [120]. Chapter 4 focuses on implementing RNS in a card shuffling algorithm. This algorithm can be tailored to rely on inputs from the RNS PRNG described in Chapter 3 and includes modulo arithmetic in its final shuffling process.

2.6 RNS PRNG in Efficient Security and Mixed-Radix Conversions

This chapter discussed previous work in lightweight cryptography, provable security, mixed-radix and non-binary number systems, and RNS. This thesis focuses on the pseudorandom number generator (PRNG), a type of lightweight primitive that provides a source of randomness in applications that involve devices that lack the processing capacity to rely on a true random number generator (TRNG). Some PRNGs are better suited for certain applications than others. Chapter 3 discusses methods to measure the quality of a PRNG, such as statistical tests like the NIST RNG test suite [68] and the FSU DIEHARD suite [69]. However, for PRNGs with lightweight security applications, it is of high importance to consider a different metric of PRNG quality: resistance to attacks.

This thesis will analyze the security on a RNS-based PRNG [121] that incorporates a modified version of the Chinese Remainder Theorem as well as surjective modulo maps to defend the outputs from reverse engineering attacks. This PRNG has previously been shown to pass many statistical tests of randomness, but it was not created specifically for communications security (COMSEC), which involves protecting telecommunications systems from unsanc-

tioned entities. Instead, this PRNG was built for transmission security (TRANSEC), whose goal is to decrease the probability of interception [122, 123]. Thus the goal of analyzing this PRNG is not to prove security but instead is to decrease the probability of the system being reverse engineered. We present an attack on the system that invalidates the system's true security. Then, defenses are made to prevent these attacks, showing that the PRNG can be modified to resist this specific attack. Therefore, this PRNG, including the suggested modifications, is an efficient method to produce randomness and is resistant to this attack.

Chapter 4 ties together PRNGs and the concept of mixed-radix number systems by introducing a card shuffling algorithm that loses true randomness through a mixed-radix shuffling technique and through PRNG inputs. The pseudorandom input values can be from any choice of PRNG, and this thesis evaluates the effectiveness of utilizing a PRNG that contributes more entropy loss but provides better computational efficiency than higher-power algorithms supply. It is important to decide on the optimal J-value to remain constant while test implementations with different PRNGs take place. To do this, this thesis quantifies the entropy loss of the final shuffling process, which is valuable in that the entropy loss formula takes radix value into account. Thus, entropy loss can be calculated precisely for any index value.

Utilizing PRNG and RNS for card shuffling games is significant. Casinos benefit from relying on lightweight primitives and RNS implementations due to the lower cost and less processing requirements compared to employing high power RNGs such as TRNGs. However, security still remains an issue for casinos, and it is necessary to weigh between cost-reducing methods and better quality randomness. This thesis discusses this balance and utilizes mixed-radix conversions to show that low-power implementations are indeed feasible to use in casino applications, and, with future work, more game and combinatorial applications.

Chapter 3

Attacks and Defenses of Single Stage Residue Number System PRNGs

3.1 Introduction

A variety of systems and algorithms rely on random number generation to provide an element of uncertainty or security within their chosen applications. Random number generators (RNG) generally bifurcate into categories for True RNGs, which rely on naturally occurring entropy sources (e.g., temperature variations [124], oscillator time variations [125], quantum parameter observations [126], or ambient RF noise [127]) for their *random* generation, and Pseudo-Random Number Generators (PRNGs), which employ deterministic algorithms to calculate sequences of *pseudorandom* numbers.

TRNGs offer truly non-repeatable random bit streams, yet can be severely limited in their rate of production due to the entropy of the underlying physical process [124],[128]. While a few higher throughput approaches [129] have been identified, TRNG outputs cannot be replicated at a secondary location, requiring key transfer mechanisms [130] or alternate methods when applying to a coherent application like cryptography or digital communications. As a result, PRNGs are more commonly used in practice: these algorithms can be exceedingly simple, such as an irreducible polynomial calculation with a random seed [131], or exceedingly complex, such as an output of the Advanced Encryption Standard (AES) algorithm

in cipher block chaining feedback mode [132]. Specific designs for PRNGs range from combining cellular automata of different dimensions [63], linear feedback shift registers (LFSR) [62] [133], calculations produced from the chaotic Hénon map [134] [63], residue number system (RNS) arithmetic combined with the Chinese Remainder Theorem (CRT) [135], and large matrix operations [136]. Composite PRNGs with variational processes like precession have also been constructed for increased security as used in the Global Positioning System (GPS) [135]; these combining process employ techniques like bit-wise XORs, Galois extension fields [137], the CRT, or hashing functions [138]. We will focus primarily on the class of PRNGs that compose small polynomial computations over mutually prime rings, which are subsequently combined into a single multi-bit PRNG output word, via the CRT.

Given the vast quantity of RNGs [139] [140], some PRNGs are better suited than others for specific applications. RNGs in transmission security (TRANSEC) [141] [122], cryptography [64] [142], spread spectrum [143], steganography [144], and other types of secure communications require high quality RNGs that can be coherently duplicated and synchronized at a distant end, or coupled with computationally expensive side channel methods for key exchange [130]. The vulnerability in these security-oriented applications is a risk to the underlying data being processed, stored, or transmitted, which may be obtain if the RNG can be duplicated [145] or reverse engineered [146]. The rate of entropy required for these secure communications applications can be many Gigabits per second, eliminating most TRNGs. Applications of RNGs in non-security arenas, such as Monte Carlo simulation parameters [66] and lattice-based field theory [67], need only adhere to good statistical behaviors, consistent with the tests provided by the NIST RNG test suite [68] and FSU DIEHARD suite [69].

Beyond statistical measures, the quality of a PRNG is also determined by the susceptibility to attacks used to reverse engineer its operation. A PRNG involved in a cryptographic algorithm does not add to its system's security if the generator can be reverse engineered in

reasonable time. Further, the attacks and methods to reverse engineer the [usually known] deterministic PRNG process often arise after the PRNG is in place, impacting the security of all previous data transported via the system [147]; even our best algorithms [148] possess identified vulnerabilities as future computing methods become available [9].

A variety of reverse engineering attacks have been developed to target the underlying PRNGs used in various systems. Attempts to reverse engineer LFSR structures proved successful as early as the 1980s [149] [150]. Attacks on the GPS Civilian / Acquisition (C/A) code are routinely taught using the Berlekamp–Massey algorithm [147]. Other attacks and protections have been devised for the use of PRNGs in semi-coherent processing, merging the efforts of deterministic cryptographic processing with noisy signal processing [114]. Additional statistical tests can also be expanded by maps involving the Kolmogorov-Sinai entropy of a system [151] to translate time sequences of outputs into conditional probabilities between output states. Reverse engineering techniques are also used to attack other systems of varying designs and applications. For example, insertion attacks applied to ANSI X9 standards that employ DES and AES algorithms have been proven effective under certain conditions [152]. Another attack developed for a Vernam cipher, a steganalytic approach to hide messages in images, combines a brute force search with parallel programming to extract and later reconstruct hidden information. [65].

After a reverse engineering technique is established, one may try to strengthen a system against this attack through harder-to-implement encryption techniques, such as RSA or Elliptic Curve Cryptography (ECC). A potentially simpler solution involves deliberately introducing noise in the output values that the system creates. Noise derived from natural occurrences can affect output values to the point of over-complicating reverse engineering algorithms. Deliberately introducing noise in ciphertexts and classic cryptosystems such as DES has already been discussed in both the cases in which noise is added before (extrinsi-

cally) or after (intrinsically) the encryption process [153] [154]. Intentionally injecting noise into PRNG systems specifically has been explored and implemented for a signal [114] [155] [156]. In this paper, we apply this technique to strengthen a PRNG system from a certain reverse engineering attack.

In this chapter, we develop a specifically tailored reverse engineering attack on the single-stage RNS-based PRNG [121] that incorporates a modified version of the CRT (mCRT) as well as a surjective modulo map to obfuscate outputs from reverse engineering. A definition of the system parameters and PRNG construction are provided in Section II. In Section III, we extend the basic concept of a Kolmogorov entropy metric to create a multi-dimensional frequency analysis and state mapping under which we are able to reverse engineer the initial state (i.e., key) of the PRNG efficiently provided knowledge of the overall system's state transitions. This discrete "corner-turning" approach uses joint frequencies that sufficiently reduce the search space required in the reverse-engineering process. We demonstrate these techniques through white box attacks on a toy example and Matlab simulations for an Internet of Things (IoT)-caliber RNS generator to gauge the improvement over brute force attacks in Section IV, prior to presenting a reverse engineering approach that permits identification of RNS parameters (but not key) efficiently in black box testing for arbitrarily large RNS constructions. It should be noted that, while this reverse engineering attack succeeds at the RNS generator described in [121], the associated vulnerability has already been addressed via dynamic behavior of the RNS structure as shown in [157]. In section IV, we explore defenses for the RNS-based PRNG. First, we introduce how noise, in the form of slight changes in output values, affects the efficiency of the attack technique. We then modify the reverse engineering algorithm to adapt to the presence of noise by proposing fallback options when errors are introduced. Following this, we explore time and code hopping: common signal protection techniques that can be applied to the PRNG for further security. Section V sets

up simple and large examples of the PRNG and demonstrates the deterministic reverse engineering algorithm. Section VI discusses results with the same examples, but in the stochastic case. Overall conclusions and extensions to future work are then provided in Section VII.

3.2 Problem Framework

The PRNG we investigate involves a series of steps that utilize several mathematical tools, mainly a modified CRT step and modulo arithmetic. These mathematical tools are described in Section 3.2.3. To support these discussions, we define a mathematical model of the RNS construction and underlying assumptions as presented in [121] to better offer a context for the reverse engineering process involving the RNS-based PRNG.

3.2.1 System Description

The described algorithm is based on a single-stage RNS-based digital chaotic circuit previously defined using Galois field arithmetic [121]. This system does not rely on any higher-level algebraic techniques and will be described through simple mappings and permutations.

A set of pre-determined primes

$$\mathcal{P} = \{p_0, \ldots, p_{n-1}\},$$

with multiplicative characteristic

$$M = \Pi_{l=0}^{n-1} p_l$$

are utilized in a series of steps throughout the algorithm.

For each prime, $p_i \in \mathcal{P}$, the vector $X = \sigma(p_i)$ permutes the elements of $\{0, \ldots, p_i - 1\}$. The resulting sequences $\sigma(p_i)$ are used to define a state at time index i, denoted as a vector $\vec{X}_i$ of length n, where

$$X_i[l] = X_i[l \bmod p_i], \tag{3.1}$$

and $0 \leq l < M$.

Next, we apply a step similar to the Chinese Remainder Theorem (see section 2.3 for a discussion of the mathematical tools) that combines the values within each state vector. This step, called the modified Chinese Remainder Theorem step (mCRT), creates the vector $\vec{Y}$ of length M, where

$$Y[l] = \sum_{i=0}^{n-1} X_l[i] \frac{M}{p_i} \pmod{M}. \tag{3.2}$$

The modified version of the CRT eliminates the additional calculations of rotational constant $\left(\frac{M}{p_i}\right)^{-1} \pmod{p_i}$ that is utilized in the original Chinese Remainder Theorem, while maintaining the security of the overall process. Combined, the vector values of $\vec{X}$ form an exhaustive permutation of $\mathbb{Z}_M$.

The final step is a surjective mapping process that implements a modular reduction of the values in $\vec{Y}$ to a field of much smaller size 2^k. We assume that $\mathcal{P}$ consists only of odd primes, ensuring that M is not divisible by 2. The final output is a vector $\vec{Z}$ of size M, where

$$Z[l] = \phi(Y[l]) = Y[l] \pmod{2^k}. \tag{3.3}$$

A depiction of this RNS process, where each prime field is represented by a distinct gear in a composite set that rotates in unison over time, and subsequently aggregated via the mCRT process into an observable output sequence is shown in Fig. 3.1.

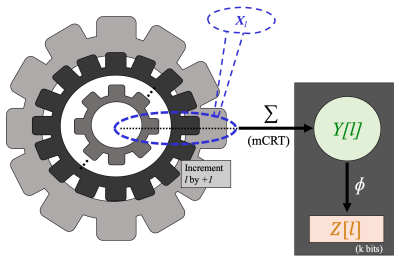

Figure 3.1: A visualization of the RNS PRNG computations.

Label	Description
X_l	$X_i[l] = X_i[l \bmod p_i] = \sigma_l(p_i)$
mCRT	$X_l[i] = Y[l] \cdot \left(\frac{M}{p_i}\right)^{-1} \bmod p_i$
$Y[l]$	$Y[l] = \sum_{i=0}^{n-1} X_l[i]\frac{M}{p_i} \bmod M$
$Z[l]$	$Z[l] = \phi(Y[l]) = Y[l] \bmod 2^k$
ϕ	Surjective mapping $\mathbb{Z}_M \to \mathbb{Z}_{2^k}$

Figure 3.2: Descriptions of the mathematical computations in Figure 3.1.

3.2.2 Assumptions and Goals

The PRNG's resilience to potential attacks that are designed to reverse engineer the system is an important measure of its quality. An algorithm that successfully reverse engineers the PRNG state could significantly influence the value of this system in any application. In this analysis, we aim to construct such a process. A reverse engineering attack on this PRNG relies on some core assumptions. First, we assume that we have complete knowledge of the system's process and parameters as outlined in the previous section for a white box system attack. Understanding the system description, especially the map ϕ, allows us to utilize expected frequency data in the proposed technique. We also assume knowledge of the set of primes $\mathcal{P}$, the permutations σ_i's, and the output vector $\vec{Z}$. In Section 3.4.3, we relax this assumption significantly and demonstrate how to extract a rotated version of such information from a black box attack. Additionally, we assume a priori knowledge of expected L-dimensional frequency data, i.e. the amount of transitions of the form $Z_l[i], Z_l[i+1], \ldots, Z_l[i+L]$ for an integer value L. While this list of assumptions may sound extreme, the underlying assumption for IOT-caliber devices is that an attacker has, or will gain, physical access to a unit that can only afford to protect the key. While practical to obtain such data using brute force techniques for $M \lessapprox 2^{48}$, which covers many IoT applications, and subsequently use to efficiently reverse any newly chosen key value, larger RNS constructions

(e.g., $M \approx 2^{256}$) can be reduced to significantly less than brute force searches using these techniques.

The goal of the reverse engineering process is to determine the initial state $\vec{X}[0]$. Since the states are determined by the known permutations σ_i, identifying any state at a specific index l is sufficient to successfully reverse engineer the entire system. It is also important to note that the CRT step is easily reversible since it is a bijection on $\mathbb{Z}_M$. Thus, the main goal of the present analysis is in reversing ϕ, i.e. identifying one pair (Y_l, Z_l) in which $\phi(Y_l) = Z_l$.

There are exactly M potential values that the function ϕ maps to any given vector value of $\vec{Z}$, allowing for brute force attacks utilizing $\mathbb{Z}_M$ as the search space. Most applications of this system, however, are resilient to these attacks due to the time required to test every value of $\mathbb{Z}_M$ for large M. Expected frequencies of output range $\mathbb{Z}_{2^k}$, calculated from the surjective map ϕ and permutations σ_i, potentially give enough insight to reduce this search space. Our proposed attack outlined in Section III explains how examining multi-dimensional frequencies can reduce the brute force search space to reverse engineer the PRNG efficiently.

3.2.3 Mathematical Tools

In this section, we focus on the constraint on state index l. With each passing time step, the l^{th} component of $X[l]$ accesses the value of the $l \pmod{p_i}^{th}$ component of $\sigma(p_i)$. One can view this changing of values as a rotation of a gear whose p_i teeth are labelled in the order of its corresponding p_i-characteristic permutation. Considering every component of $X[l]$ as l varies, we envision the components of $X[l]$ shifting in tandem. In our analogy, the gears representing each prime also turn in lock step with every passing time value, however their different sizes guarantee that the aggregate gear construction does not repeat until the least common multiple of $\mathcal{P}$, which is equal to M since $p_i \in \mathcal{P}$ have no common divisors. Fig.

3.3 provides a visual of this alignment. If the gears originally align at $X[0]$, the gears will realign after M cycles, the least common multiple of the primes, rotations. We thus attain periodicity of $X[l]$ values, as $X[0] = X[M]$, or generally $X[l] = X[l \pmod M]$.

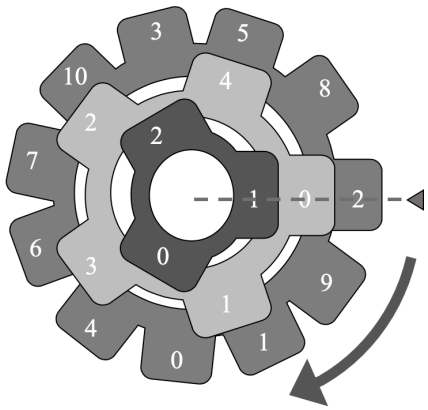

Figure 3.3: A visualization of permutations $\sigma(3), \sigma(5)$, and $\sigma(11)$ as gears. At each time increment, the gears turn clockwise by one tooth. After $M = 165$ iterations, the initial value (2,0,1), realigns.

We also consider the reversibility of the calculations involved in the system description. The first of these steps, generating $X[l]$ for $0 \leq l < M$, is simply reversed by computing each permutation's inverse. The reversibility of this initial step allows some leniency in an attack, as isolating any state $X[l]$, regardless of l, is enough to reverse engineer the PRNG. Equation (3.2) is a modification of the original Chinese Remainder Theorem, defined by

$$Y[l] = \sum_{i=0}^{n-1} x_i[l] \frac{M}{p_i} \left(\frac{M}{p_i} \right)^{-1}_{(\text{mod } p_i)} \pmod M. \tag{3.4}$$

The above equation is easily reversible by computing its output modulo each prime. Equation (3.2) modifies the CRT by removing the operation of multiplying inverses $\left(\frac{M}{p_i}\right)^{-1}$. Removing this operation also transfers this step to the calculation used to reverse equation (3.2): given

output $Y[l]$, we determine the corresponding $X[l]$ by

$$X_i[l] = Y[l] \cdot \left(\frac{M}{p_i}\right)^{-1} \pmod{p_i}. \tag{3.5}$$

The original CRT applied to every combination of n components, each chosen from $\sigma_i(p_i)$, produces every element of $\mathbb{Z}_M$ uniquely once. Thus, applying this formula to calculate $\vec{Y}$ produces a permutation of $\mathbb{Z}_M$. Replacing this formula with equation (3.2) creates a different permutation of $\mathbb{Z}_M$, as the inverses $\left(\frac{M}{p_i}\right)^{-1}$ are bijective rotational factors that scramble the permutation on M elements generated by equation (3.4). This scrambling is trivial to reverse. In section III, we shift our focus to reversing the final surjective mapping ϕ – we create a mathematical model of a potential attack utilizing expected frequencies of $Z[l]$ component values.

3.3 RNS-Based PRNG Attacks

The attack on ϕ relies on expected frequencies of $Z[l]$ component values. The frequencies of $\mathbb{Z}_{2^k}$ elements in $Z[l]$ can be generated without knowledge of the initial state $X[0]$ by successively "stacking" the values of $\mathbb{Z}_M$ into 2^k bins according to their value mod 2^k. Each bin represents the preimage of an element of ϕ's codomain $\mathbb{Z}_{2^k}$, i.e. the brute force search space of each component value $Z[l]$ in $\vec{Z}$. Since 2^k cannot divide M, this one-dimensional frequency analysis produces a "near uniform" distribution in which some elements of $\mathbb{Z}_{2^k}$ appear at a frequency of one higher than others [**?**].

When conducting a brute force attack on small systems, one would benefit from this analysis: the brute force search space for some components of $\vec{Z}$ is slightly smaller than that of

others. For large values of M, the difference between these search spaces is insignificant, and a one-dimensional frequency analysis is unhelpful in an attack. In a two-dimensional frequency analysis, we calculate frequencies of conditioned transitions from $Z[l]$ to $Z[l+1]$. The resulting histogram is not uniform, and we hope to extract more information from this mapping exploiting this non-uniformity. In particular, any transition with a frequency of one uniquely identifies the resulting state since there is only one state capable of following that pattern. Unfortunately, for larger M values, it is unlikely that any transition appears exactly once in a two-dimensional analysis. If uniformly distributed, we anticipate a per-bin two-dimensional density of $\approx \frac{M}{(2^k)^2}$. Extending this approach, we then examine L-dimensional frequencies, which calculate frequencies of transitions of $Z[l-L+1]$, to $Z[l-L+2]$, etc., up to $Z[l]$. We continue performing multi-dimensional analyses until we arrive at a bin with a small enough transitional frequency for a brute force search in reasonable time. The expected density (if uniform) of the L-dimensional bin is thus $\approx \frac{M}{(2^k)^L} = \frac{M}{2^{kL}}$, corresponding to a string of L successive output observations. Once we have isolated a state, and thus successfully reversed ϕ, we can easily translate to a chosen number of states in the past or future to obtain the initial state.

To visualize this analysis for larger dimensions, we envision this process as a corner turning algorithm, in which every corner turn represents a higher dimensional frequency analysis, cumulatively representing a data structure of size $(\mathbb{Z}_{2^k})^L$. The contents of each hallway behind each door value make up the brute force search space represented by the corresponding transition. Figure 3.4 illustrates the one-dimensional (L=1) frequency analysis, which is essentially a histogram on $\mathbb{Z}_{2^k}$ with base axis represented by the doors in the main, darker hallway. At this point of the analysis, we know the contents behind each door, but we do not have access to their placements within the lighter hallways.

Suppose the number of values behind each door, i.e. the size of the brute force search

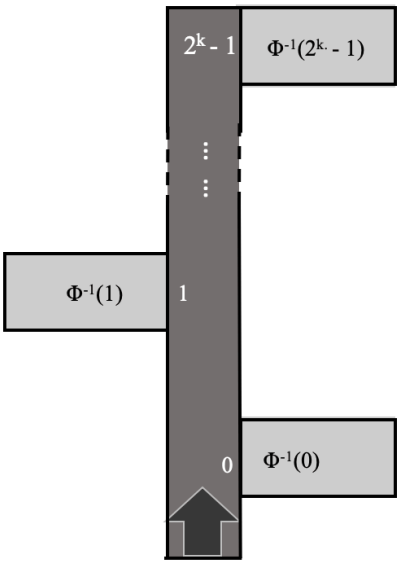

Figure 3.4: A one-dimensional ($L = 1$) frequency analysis is visualized as one "hallway" in the corner-turning process.

space, was too large. Figure 3.5 shows an example of a two-dimensional analysis in which we focus on transitions $0 \to Z[l+1]$, therefore turning the corner represented by Door "0" in the previous dimension. Thus, we are now aware of the search space corresponding to transitions $0 \to j$, where $0 \leq j \leq 2^k - 1$.

Conducting a three-dimensional analysis is analogous to turning a second corner. Figure 3.6 presents an example in which we enter the hallway labeled $2^k - 1$. This corner turn illuminates the search space corresponding to transitions $0 \to 2^k - 1 \to j$, where $0 \leq j < 2^k$. Utilizing the corner-turning illustrations allows us to visualize the frequency analyses and known search spaces for higher dimensions. Under the assumption of near-uniformity over $\mathbb{Z}_{2^k}$, each dimension crossed (or corner turned) reduces the brute force search space by a factor of 2^k. Even for large examples, each corner turn is expected to greatly reduce the amount of time required to reverse engineer the system. We test this hypothesis in the next section, in which we apply this method onto two tangible examples.

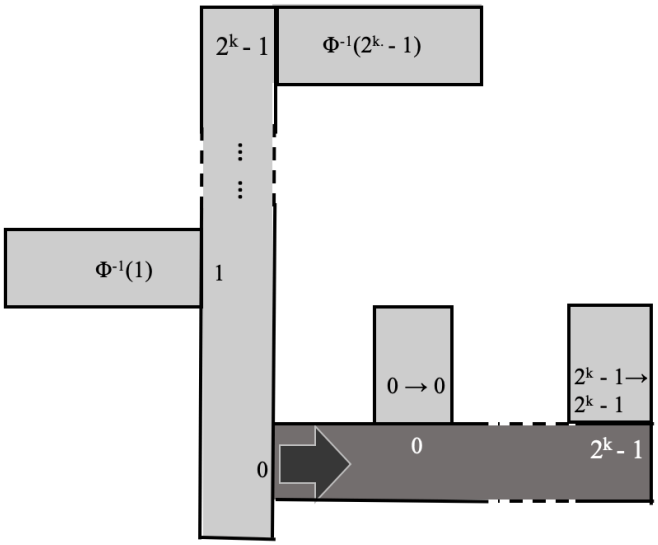

Figure 3.5: The two-dimensional ($L = 2$) analysis represents turning a corner. We are lead into a hallway that distributes the search space $\Phi^{-1}(0)$ into smaller batches.

3.4 RNS Reverse Engineering Attacks

In this section, we detail the experimental results of applying the reverse engineering process in concrete examples. We first implement the proposed attack on a toy example ($M \approx 2^{7.4}$) to provide a better understanding of the system and reverse engineering process. We then conduct the reverse engineering algorithm with a larger simulation $M \approx 2^{34}$ to illuminate the effectiveness of the proposed attack on scale of many IoT devices (i.e., 32-bit keys); that approach is reasonable to extend to sizes on the order of $M \approx 2^{48}$, at which point memory and processing constraints of a brute force search exceed bounds of present day hardware. Using the approach on larger examples can be used to reduce the space of a brute force search by the similar 2^{32} to 2^{64}.

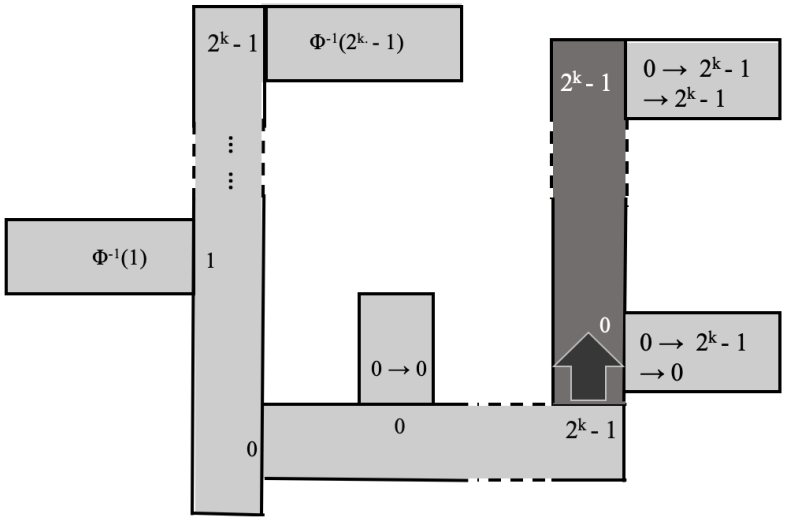

Figure 3.6: Three-dimensional (L=3) visual of the corner-turning process, assuming the turns $0, 2^k - 1$.

3.4.1 Toy Example

First consider an example with a small set of primes, $\mathcal{P} = \{3, 5, 11\}$, $n = 3$, and $M = 165$, where the overall output is reduced to a 3-bit value, $k = 3$. The arbitrarily selected permutations, $\sigma_i(p_i)$, are

$$\begin{aligned} \sigma_0(3) &= (1, 0, 2) \\ \sigma_1(5) &= (0, 1, 3, 2, 4) \\ \sigma_2(11) &= (2, 9, 1, 0, 4, 6, 7, 10, 3, 5, 8), \end{aligned}$$

The initial state of this system is $< 1, 0, 2 >$, and the states $X[l]$, $0 \leq l < 165$, are tabulated in Table 3.2. The states of the system are also represented visually by the gears in Figure 3.3. Table 3.2 also tabulates intermediate calculations (applying Equations 3.4 and 3.2) in the CRT output and $Y[l]$ columns. The final mapping process utilizes the function $\phi : \mathbb{Z}_{165} \rightarrow$

Table 3.1: Data from toy example.

l	$\vec{X}_l$	CRT output	$Y[l]$	$Z[l]$
0	$< 1, 0, 2 >$	145	85	5
1	$< 0, 1, 9 >$	141	3	3
2	$< 2, 3, 1 >$	23	59	3
⋮	⋮	⋮	⋮	⋮
163	$< 0, 2, 5 >$	27	141	5
164	$< 2, 4, 8 >$	74	32	0

$\mathbb{Z}_{2^3}$, where

$$\phi(Y[l]) = Y[l] \pmod 8. \tag{3.6}$$

The output values of this mapping are tabulated in the last column of Table 3.2.

We have successfully set up the PRNG with the given initial state and permutations. Next, we reconstruct the initial state given the assumptions described in Section III.

Toy Example: Implementing the Attack

Reverse engineering this system assumes knowledge of $\mathcal{P}$, all permutations $\sigma_i(p_i)$, the mapping process ϕ, and output vector $\vec{Z}$. As explained in Section 3.2.3, we focus on determining a pair $(Y[l], Z[l])$ in which $\phi(Y[l]) = Z[l]$.

The modular setup of the surjective mapping process allows us to construct a histogram of one-dimensional frequency data of coordinate values from output vector $\vec{Z}$. Creating this histogram is analogous to placing each element of $\mathbb{Z}_{165}$ into 8 bins (columns) corresponding to its value modulo 8, shown in Figure 3.7. Each bin therefore represents the brute force search space for each element of $\mathbb{Z}_{2^3}$.

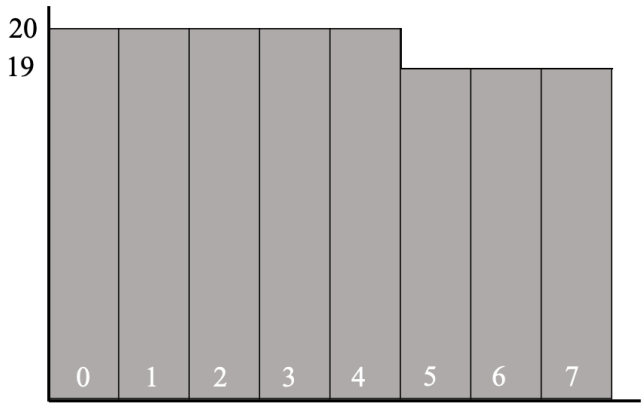

Figure 3.7: Labelled histogram

This histogram is not uniform, as 2^3 does not divide $M = 165$; three of the search spaces are smaller by one value. From this one-dimensional analysis, we find that three of the brute force search spaces are smaller than the others. We can also visualize the one-dimensional analysis in terms of the corner-turning algorithm. In this case, the bins are represented by hallways branching off the main, one-dimensional hallway shown in Figure 3.8.

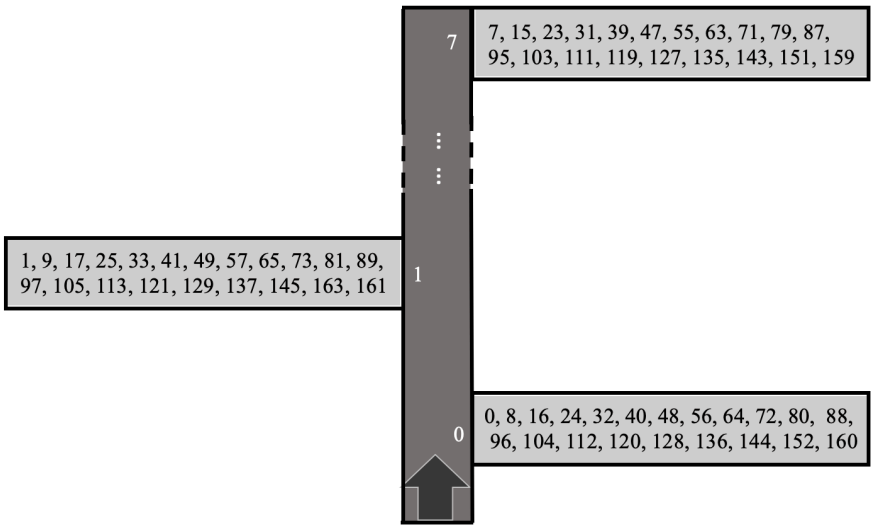

Figure 3.8: A one-dimensional visualization of reverse engineering toy example.

Though any of the bins are small enough to administer a brute force attack on any vector

value of $\vec{Z}$, values labelled 5, 6, or 7 have somewhat smaller search spaces. However, we have not yet isolated a single state, so we implement a two-dimensional frequency analysis.

A two-dimensional analysis evaluates the frequencies of transitions of vector values $Z[l-2+1]$ to $Z[l-1+1]$. We display these frequencies via the joint histogram in Figure 3.9, where, for example, the bin in position 3 along axis n and position 2 along axis m represents the amount of transitions represents the amount of transitions $3 \rightarrow 2$.

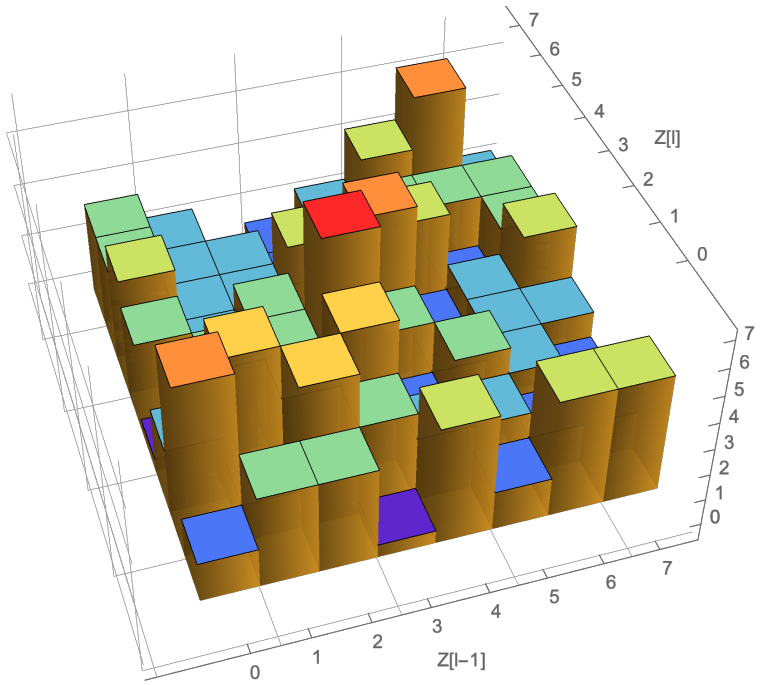

Figure 3.9: Expected two-dimensional transitional frequency data of toy example outputs, shown through a surface plot histogram.

Observing the histogram, there are several transitions of frequency one, highlighted in dark blue. We focus on the bin representing the transition $0 \rightarrow 0$. Since this transition only happens once, the transition corresponds with the $\vec{Y}$ vector value 30, and we have successfully reversed ϕ. Conducting the two-dimensional analysis represents a corner turn. Since we focus on the transition $0 \rightarrow 0$, we turn the corner labelled "0." Turning this corner reveals a hallway with doors with occupancy one - specifically, the door labelled "0," representing the isolated

transition corresponding to the $\vec{Y}$ vector value 30. Figure 3.10 provides a visual of the second corner turn.

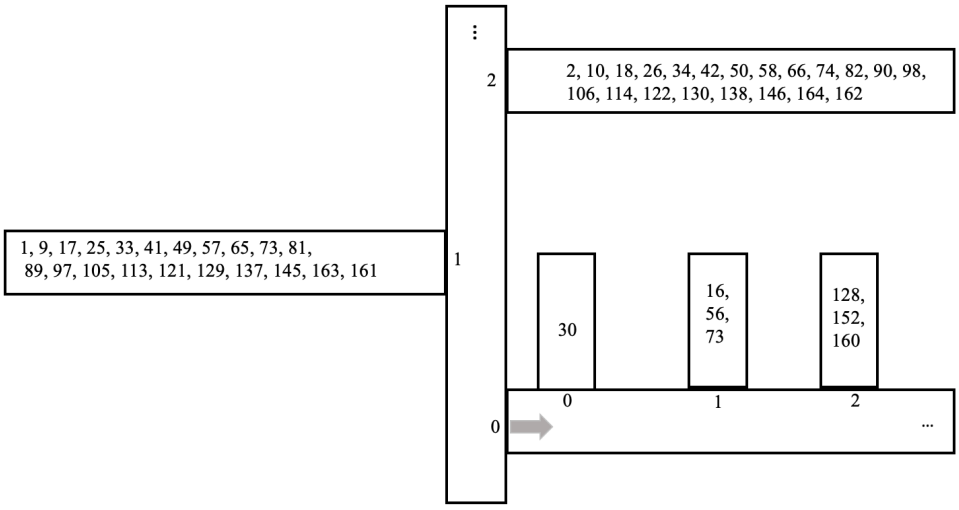

Figure 3.10: A two-dimensional frequency analysis of toy example output $\vec{Z}$ component values.

Since we have isolated a single state, we have successfully reversed ϕ and therefore reversed the PRNG example once that output sequence $(Z[l-1], Z[l])$ is observed. Even in the scenario where the number of corner turns chosen does not result in a single unique element, the remaining set of elements represent a significantly pruned search space for subsequent trials; we will revisit this reverse engineering approach, which assumes a priori knowledge of the state transitions, after discussing a larger example.

Analysis of Toy Example Attack

Reverse engineering the toy example system is feasible via an exhaustive search over $M = 165$ elements due to its simplicity. Our reverse-engineering algorithm, however, allows us to reverse engineer any initial condition within 2-3 observations of the PRNG output. In the

next section, we show how our attack even further reduces the time required for an exhaustive search on a larger, more realistic system.

3.4.2 Implementing PRNG on IoT-Caliber Example

To expand upon that analysis, we explore the approach on a larger example, using a four-prime RNS sequence generator that has a repetition period of approximately $2^{33.7}$:

$$\mathcal{P} = \{251, 257, 467, 479\},$$

and

$$M = \Pi_{i=1}^{4} p_i = 14,429,764,351.$$

In this example, rather than arbitrary permutations, we replace the bijective maps with irreducible digital chaotic polynomials with generating functions f_{p_i}. To further simplify the discussion and show applicability, we choose to use the same irreducible chaotic polynomials used in [121]. The functions

$$f_{p_1=251}(x) = 3x^3 + 3x^2 + x + 39 \qquad f_{p_2=257}(x) = 3x^3 + 3x^2 + x + 110$$

$$f_{p_3=467}(x) = 3x^3 + 3x^2 + x + 15 \qquad f_{p_4=479}(x) = 3x^3 + 3x^2 + x + 233$$

generate the bijective maps for this example. We set the initial state as $< 0, 0, 0, 0 >$. State generation and mCRT steps are then calculated to produce $\vec{Y}$, and the final mapping $\phi : \mathbb{Z}_M \rightarrow \mathbb{Z}_{2^4}$.

We then used a script to calculate all M iterations of the sequence generator. Prior to this,

look-up tables were generated for each prime, based on the polynomials above, to reduce computation time by exploiting the known repetitions of the p_i-adic polynomial calculations, similar to the gears depiction of Fig. 3.3. An $(L = 4)$-dimensional data structure, with each dimension sized 2^k, was created before the sequence is generated. Overall, this data structure is of size 2^{Lk}. This structure has no effect on computation time, but represents how much storage space is used to retain L-dimensional histograms, each consisting of 2^k bins. As the script iterates through the sequence generator, it maintains a sliding window of size L and increments the corresponding coordinate in the L-dimensional structure. Depending on the choice of L for the script, the structure can keep track of L turns at a time. This can also be viewed in context of the corner-turning example from the toy example. Figure 3.11 provides a visual of the sliding window and corner turning method used in the script. The last 4 outputs of the sequence generator are reported as $[1, 5, 10, 3]$. The script then traverses through the 4-dimensional structure as depicted with the corner turns, and increments this state to represent a known state that achieves that output state transition pattern.

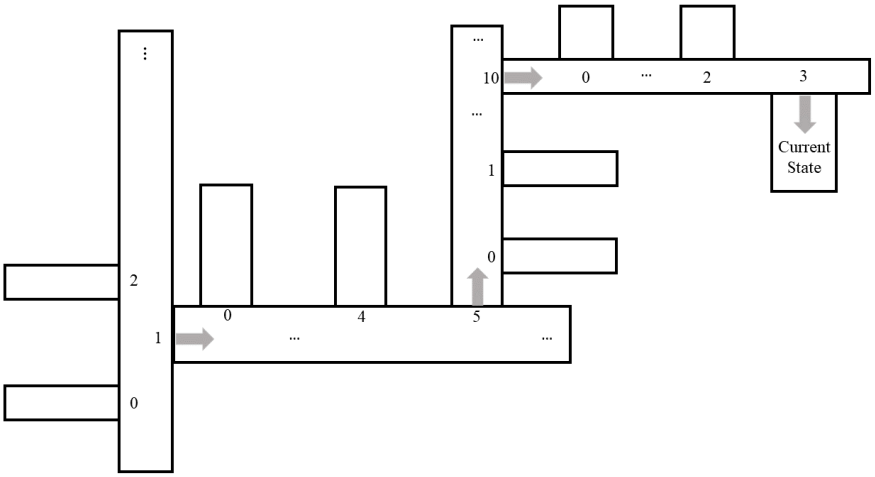

Figure 3.11: A visualization of the sliding window and respective current state used in the large example.

Analysis of IOT-Caliber PRNG Attack

The larger example was run with $k = 4$ and $L = 4$. While the initial run-time of the algorithm requires ≈ 30 hours to brute-force compute all states with $M \approx 33.7$ bits. Once the conditional states are catalogued (a one-time investment) using one core of an Intel processor, the structure generated by the script, which was 57 GB, reduces the search time for any individually observed 4-state sequence by $\frac{M}{2^{Lk}}$, requiring around 18 seconds. The only trade-off is the storage space required to maintain this data structure. Furthermore, by examining a single-state, one can maintain a record of each iteration for which that state was incremented (i.e., retaining a linked list of the elements on $\mathbb{Z}_M$ rather than just tallying a count), and be able to calculate subsequent turns without increasing the variable L in the script. By knowing the iteration number and the prime polynomials, one can then compute subsequent numbers generated to further prune the search space. For example, the state $< 14, 10, 14, 11 >$ is incremented first at iteration 85375 of the sequence generator. From this, the current location in each prime's LUT can be found as $< 35, 51, 381, 113 >$, respectively. The next value of the sequence generator is then found to be 14, giving us insight into the next state to be incremented as well as the next corner turned for the current state. Given a specific sequence such as the one used above, we can begin to isolate the initial condition by calculating further values and pruning the states from the structure that do not take the same turns. For each additional turn, m, calculated, the search space is reduced by a factor of 2^k. As shown in Figure 3.12, the sequence chosen to isolate is $< 14, 10, 14, 11, 1, 12 >$. Initially, the conditional entropy measured is ≈ 7.99. After the first 2 turns, displayed in the first graph, the bins are uniformly distributed, revealing no information about the initial state, but the entropy has decreased substantially to 0.0625. The next 2 turns, shown in the second graph, reduces the search space by 2^{16}, and begins to display slight variation in

the distribution amongst the bins, but the conditional entropy has decreased to a fraction of what it was at the start. An additional 2 steps has reduced the overall search space to 887 items, which can be brute force searched quickly, but there is a trade-off with processing time and memory storage. The last two steps have reduced our search space to find a single state that will produce the given sequence, shown in the last graph. We have isolated the one and only sequence starting index capable of achieving the observed output pattern, and thus have found the unique pair that maps $\phi(Y[l])$ to $Y[l]$, and therefore have the initial state/key of the PRNG. Equation 3.7 sums up the above analysis for reducing the overall search space. Given a structure of size $L = 4$ and $k = 4$, we only need an additional m to reduce our search space to a unique solution. With a change in k and L, we would only need to change our the number of m steps taken to find a unique solution.

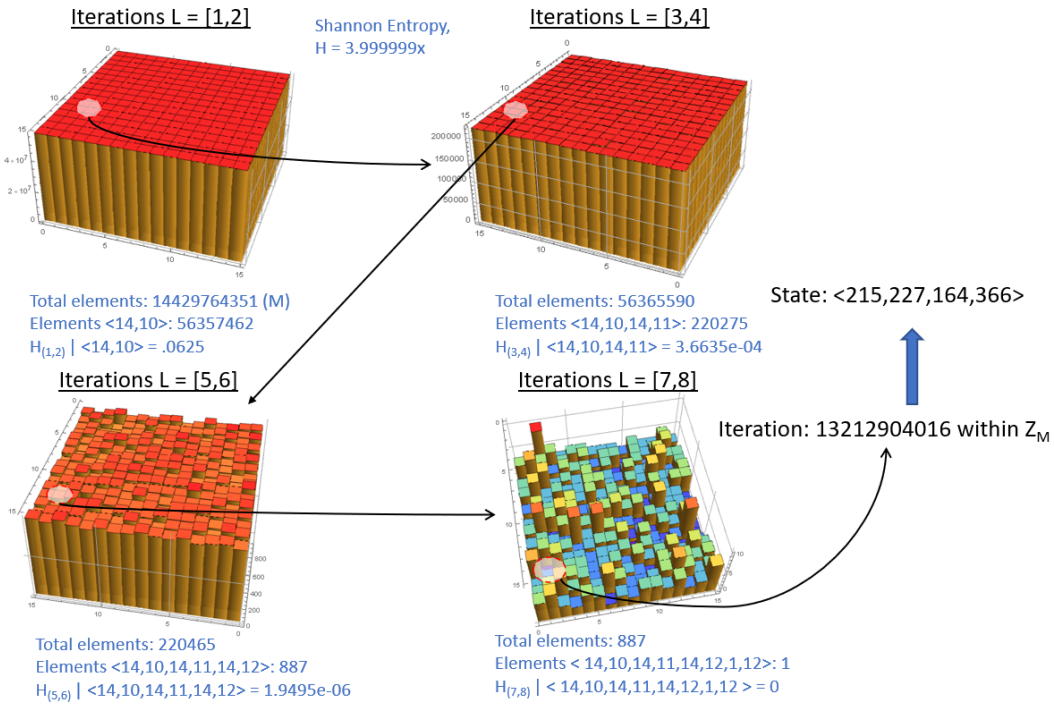

Figure 3.12: A visualization of the pruned search as it converges on a single state.

$$\text{Remaining Search Space} \approx \sum_{m=0}^{3} \frac{M}{2^{k(L+m)}} \tag{3.7}$$

3.4.3 Extending to Even Larger Examples

The previous examples make a significant, though often unrealistic, simplification in that we assumed prior knowledge and tabulation of the state transitions. Such an assumption is likely feasible for an RNS system where $M < 2^{64}$, yet the efficiently scalable design of the RNS generator makes it far easier to expand to $M \approx 2^{256}$, at which point exhaustive analysis is infeasible. Our bounding at 2^{64} is based on the lengths an attacker is to go to if the one-time brute force attack is extensible to all other devices using that PRNG, effectively amortizing its cost. Further, observation of the output $\pmod{2^k}$ limits our visibility into the component outputs $Z[l]$; we turn to the reverse engineering of those component values first and subsequently address the larger reverse engineering problem.

As a variation on a chosen plaintext attack, consider a scenario where we have physical access to the RNS generator, can choose the input state, and can observe the aggregate output state $Z(l) = \sum_{i=0}^{n-1} Z_i[l]$, but have no insight to the individual component values $Z_i[l]$. We may set the input state value to all zeros ($X = \vec{0}$), at which point, we will receive an output observation $Z[\vec{0}]$. By incrementing a single input state component, which is equivalent to making a leap forward in code space of $\frac{M}{p_i}$ steps if incrementing the i^{th} input, i.e.,

$$\vec{0} \rightarrow X[\frac{M}{p_i}] \rightarrow X[2\frac{M}{p_i}] \ldots \rightarrow X[(p_i - 1)\frac{M}{p_i}],$$

we may obtain the additive offsets of each component value $\{Z[0], Z[\frac{M}{p_i}], Z[2\frac{M}{p_i}] \ldots Z[(p_i - 1)\frac{M}{p_i}]\}$ by subtracting the baseline $Z[\vec{0}]$ output $\pmod{2^k}$.

$$\vec{Z}_i = \{Z[0], Z[\frac{M}{p_i}], Z[2\frac{M}{p_i}] \ldots, Z[(p_i - 1)\frac{M}{p_i}]\} - Z[\vec{0}] \tag{3.8}$$

Repeating this operation for all primes, $\{p_i\}_{i=0}^{n-1}$, results in a total of $\sum_{i=0}^{n-1} p_i$ chosen plaintext inputs to map out the entirety of the RNS lookup tables. Even without prior knowledge of the primes $\mathcal{P}$, observation of the repeating pattern will enable derivation of the sub-cycle lengths and subsequent calculation of all RNS system parameters $(\mathcal{P}, M)$. To simplify the discussion, consider a re-formatted series of (3.8) with dual indices indicating which prime i is in use followed by element within that series, where

$$\vec{Z}_i = \{Z_{i,0}, Z_{i,1}, Z_{i,2}, \ldots, Z_{i,p_i-1}\} - Z[\vec{0}].$$

The collection of these vectors then are direct outputs of the entries in the lookup tables and/or the component contributions to the overall composite output $Z[l]$, and thus we have reverse engineered $\sigma_i(p_i) \forall i$. Further, the unknown initial state of the system under test (returning to the operational RNS system where only the aggregate output is observable), $\vec{X}_0$, may be represented in each of these p_i-length component vectors as a rotation in the states.

$$\begin{aligned} \vec{Z}_i^*(s_i) \quad = \quad & \{Z_{i,s_i}, Z_{i,s_i+1}, \ldots, Z_{i,p_i-1}, z_{i,0}, z_{i,1}, \ldots, \\ & Z_{i,s_i-1}\} - Z[\vec{0}] \end{aligned}$$

Recognizing that these sequences repeat in time, this permits transforming the previously discussed corner turning algorithm into a series of L equations (all calculated modulo 2^k)

that may have multiple solutions, equating to all valid states producing a length L string of observed outputs for $Z[l-L+1]\ldots Z[l]$; when only one unique solution to the series of equations exists (increasing L incrementally), then the unique origin state $X[\vec{0}]$ has been identified.

$$\begin{aligned}
\sum_{i=0}^{n-1} \vec{Z}_i^*[s_i - L + 1] \pmod{2^k} &= Z[l-L+1] \\
\sum_{i=0}^{n-1} \vec{Z}_i^*[s_i - L + 2] \pmod{2^k} &= Z[l-L+2] \\
&\vdots \\
\sum_{i=0}^{n-1} \vec{Z}_i^*[s_i] \pmod{2^k} &= Z[l]
\end{aligned}$$

3.5 RNS-Based PRNG Defenses and Perturbations

In this section, we discuss two techniques to defend against the reverse engineering method in the previous section: (1) purposefully injecting noise and (2) redesigning the system to include code hopping. These low cost defense strategies further complicate the attack method and increase the computation time required to implement an attack. We also briefly consider the phenomena of time hopping, which is ultimately shown to present little additional security against an attack.

3.5.1 Noise

This subsection discusses a practical defense for this system: the intentional application of noise. Two types of deliberate noise in modern cryptographic systems have been explored in [154]: intrinsic and extrinsic. The former is noise applied prior to encryption and the latter is noise applied post-encryption. In this case we analyze the effects of intrinsic noise. Though it doesn't completely defeat the attack described above, it can increase complexity and computation time when reverse engineering to the point where it's no longer worth attacking. While there are naturally occurring cases of noise in encryption systems; such as bit corruption or signal noise during transmission, this paper discusses deliberate modification of the sequence's outputs to interfere with reverse engineering attempts. In this context, noise is defined as a ± 1 adjustment to the output, $Z[l]$. From the intended receiver's perspective, there exists a few cases from which they would have knowledge of the noise and be able to recover the correct outputs. The first is knowledge of the initial state used in the sequence generator by the transmission device. The receiver could then calculate outputs as well and compare those with the ones received, fixing any noise as needed. While this would require additional computation from the receiving device, the sequence generating algorithm is efficient and would not consume many resources. An addendum to this is strict knowledge of the PRNG used to apply the noise. The receiver would synchronize their PRNG to the transmitting device with a session key. After receiving data they would perform the inverse operation to the received sequence to remove any noise as shown in Figure 3.13.

Consider the case when the receiver does not know the algorithm, but knows the probability of noise being applied to an output. In this case the receiver must implement the attack algorithm to find the initial state of the sequence generator. In this case, they would need to maintain a sliding window of outputs that would be used to reverse engineer the original state. However, in this scenario they would not have to attempt to remove the noise and increase

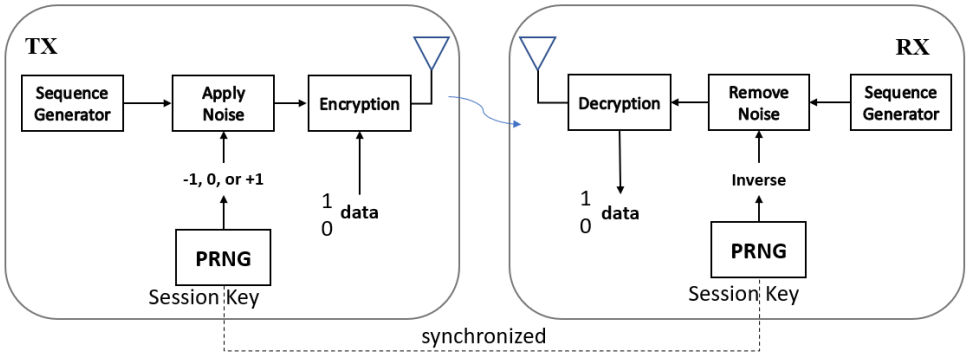

Figure 3.13: A visual of the transmit-receive devices generating an RNS sequence and applying/removing noise to the outputs.

the number of reverse engineering attempts to a maximum of $3L$. Given a small enough probability and large enough window size, they could confidently recapture L outputs and rerun the algorithm assuming noise had not been applied. For example, given a probability of 1% and a window size of 10 outputs, if the receiver does not get the correct state, they can just record 10 more outputs and recalculate. The chance of noise being applied in the next 10 outputs is slim enough that it is more efficient to recapture a set of values instead of attempting to search and rectify the noise themselves. In the event of a single error, as shown in 3.14, with a low probability, the attacker could implement the same strategy mentioned above. However, with a high enough probability, such as 20% given a window size, L, of 10, the attacker would need to modify the deterministic attack algorithm (Section 6). The receiver would need to have knowledge of the sequence generator's initial state and noise PRNG to efficiently receive and decrypt messages.

To further increase complexity, consider the case when the sender introduces multiple errors in close proximity, the attack algorithm would then need to be adjusted to account for these. One solution is to brute force every possible value, deterministic and stochastic, in each dimension L. Depending on the value of k, this substantially decreases the search space. Given a value of $k = 4$ and the noise offset at ± 1, with the potential of multiple errors, the

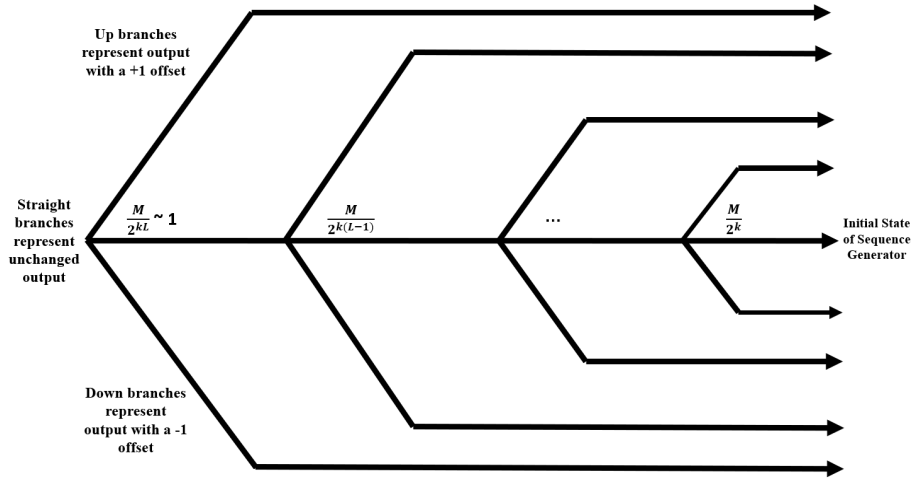

Figure 3.14: The fishbone diagram displays a visual of the possible output branching when a single ± 1 error occurs.

adjusted attack algorithm requires the attacker to examine 3 values at each dimension L instead of 16. If the range of noise increases while the value of k stays the same, the search space dramatically increases. The intended receiver would need to know the initial state and the strength of noise, or the algorithm for applying noise to the outputs.

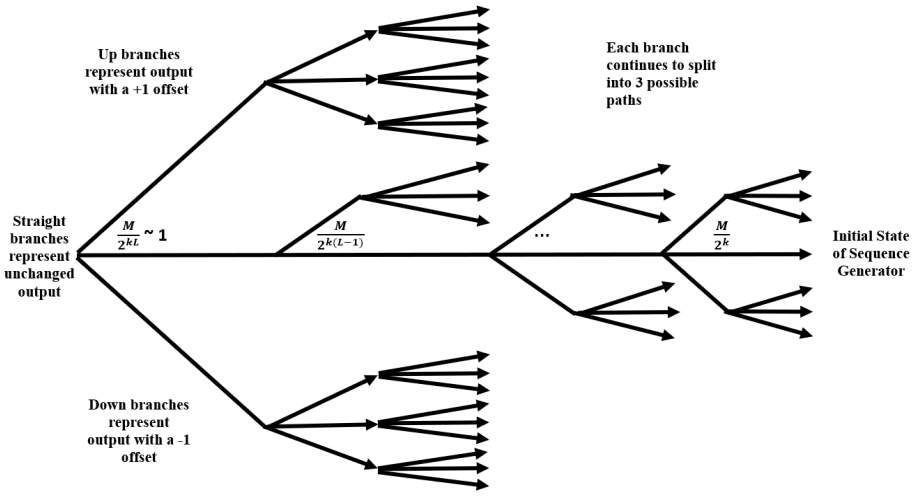

Figure 3.15: Multi-error case.

3.5.2 Code Hopping

A second potential perturbation that can challenge the attacker's efforts is "code hopping," in which the designer periodically changes the permutation parameter of the system during its implementation. In the current PRNG, this technique can be thought of as executing r original systems (as described in section 2.1), $S_0, S_1, \ldots, S_u, \ldots, S_{r-1}$. Each of these systems are defined with the same parameters with the exception of different permutations $\sigma_{u,i}$ for each prime, where u ranges from 0 to r, and i continues to range from 0 to $n-1$. The permutations that distinguish each of the r systems from each other can be determined through the Fischer-Yates shuffle [**?** **?**] on each of the original prime permutations $\sigma_{u,i}(p_i)$ involved. The designer then selects discrete time values $t_0, t_1, \ldots, t_{r-1}$ to partition the integer values 0 through M into time intervals designated for each system, with default value $t_0 = 0$. Between the time steps t_u and t_{u+1}, the results from system S_u will be recorded as output values. Each change in prime permutations during a jump from t_u to t_{u+1} represents a distinct code hop. Figure 3.16 illustrates an example of a code hopping system with $r = 2$. Note that the "right hand side" of the drawing is unchanged from Figure 3.1, illustrating that the change lies only in the permutation parameters.

The reverse engineering assumptions expand those stated in section 2.2. We assume that the attacker has access to all the system's parameters (excluding the initial condition), as well as the permutations for each system S_u and the time values of the code hops $t_0, \ldots, t_{r-1}$. The attack described in Section 3 is not altered when restricted to any of the output values for one particular system S_u. However, when performing the reverse engineering process on the entire M output values, the attacker must repeat the up front brute force algorithm for every one of the r systems implemented.

Adding a code hopping component to the RNG algorithm increases the computational com-

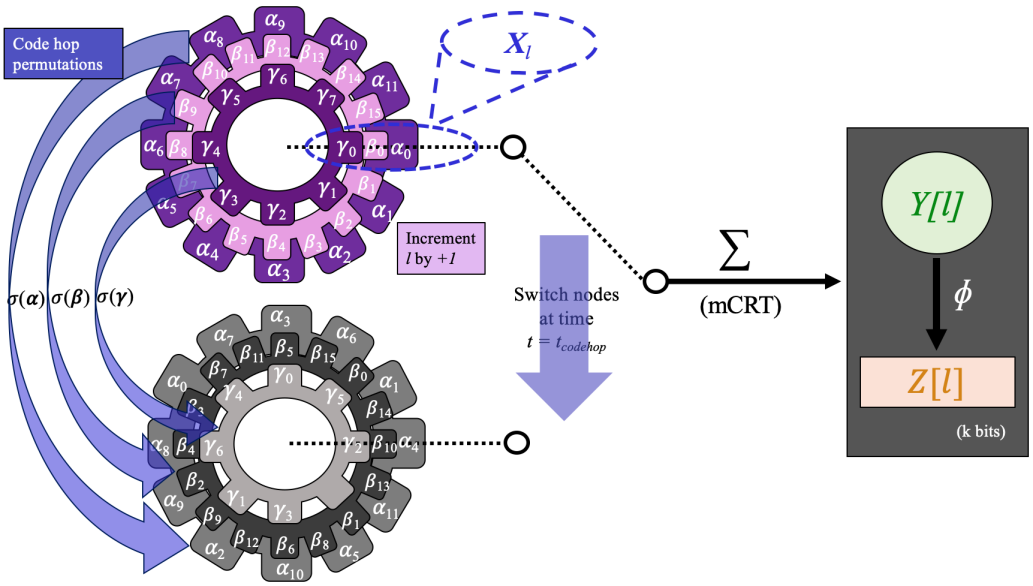

Figure 3.16: An example of code hopping with three different sets of parameters, where α, β, and γ represent the top clockwise orderings $\alpha_1, \alpha_2, \ldots \alpha_9$, etc.

plexity on the attacker's end without costing as much time for the designer. To determine the time required for the RNG designer to implement a system, we profiled the code for the RNG system (using the Fisher-Yates shuffle to generate the required permutations) as a first order estimate of the maximum number of clock cycles required for external memory access.

The code profiling process reveals that inserting code hops into the PRNG introduces an additional Fisher-Yates loop embedded into the code for the RNG system that is re-implemented each code hop. The original system implementation, with a linear time complexity in each of n (number of primes), $p_i \in \mathcal{P}$ (number of elements in each RNS component), is now repeated $(r+1)$ times to account for r code hops. Total processing scales with

$$Time \propto (r+1) \sum_{i=0}^{n-1} p_i \simeq r \cdot n \cdot \text{mean}(\mathcal{P}) \text{ clock cycles} \tag{3.9}$$

is brought up to polynomial time $\mathcal{O}(n^2)$ by introducing a code hopping component. To put

the this into perspective, we calculate the required number of clock cycles used to implement the realistic example presented in Section 4.2.1. Implementing 100 code hops on a system with four primes requires 1.94×10^5 clock cycles, showing that adding the code hopping component does not greatly increase the time required to implement the RNG.

Along with time complexity, we must also quantify how much memory consumption a code hop consumes from both the attacker's and designer's perspectives. When implementing the original PRNG system, $\lceil \log_2(\max \mathcal{P}) \rceil$ represents the amount of bits needed to store the largest of the permutations $\sigma_{u,i}(p_i)$. Unlike the analysis for time complexity, however, the memory component needed to store information does not depend on the number of code hops: in fact, the memory needed for a system with multiple code hops needs only twice the amount of memory: for the current set of permutations and the new set. Since the total number of permutations necessary to store is $\sum_{i=0}^{n-1} p_i$, the number of bits of memory required to implement a system S with r code hops is

$$\zeta_{\text{mem}} = \begin{cases} \lceil \log_2(\max \mathcal{P}) \rceil * \sum_{i=1}^{n} p_i & r = 0 \\ 2\lceil \log_2(\max \mathcal{P}) \rceil * \sum_{i=1}^{n} p_i & r > 0 \end{cases} \tag{3.10}$$

In Section 7, we also quantify the time complexity and memory requirements for an attacker to reverse engineer a code hopping system.

3.5.3 Time Hopping

Another perturbation that could add complexity to the PRNG is time hopping. In this case, the defender would simply reset the PRNG at a different initial state, overwriting $X[\vec{0}]$ to $X[l + y], y \in Z_m$. Unlike code hopping, this does not modify the permutations of the individual primes, but just modifies the current position in the sequence. This perturbation

is visualized in Figure 3.17. While simple to compute for the defender, it would not be worth implementing as the defender would need to hop frequently. Based on the attack algorithm described above, the attacker only needs to capture a window of values to reverse engineer the initial state. Even if the defender hopped at a period of at least 1 less than the window size, the attacker would have reduced the search space sufficiently that completing the reverse engineering efforts would be trivial. Although, coupled with the other perturbations in a complete system, it could add another layer of complexity.

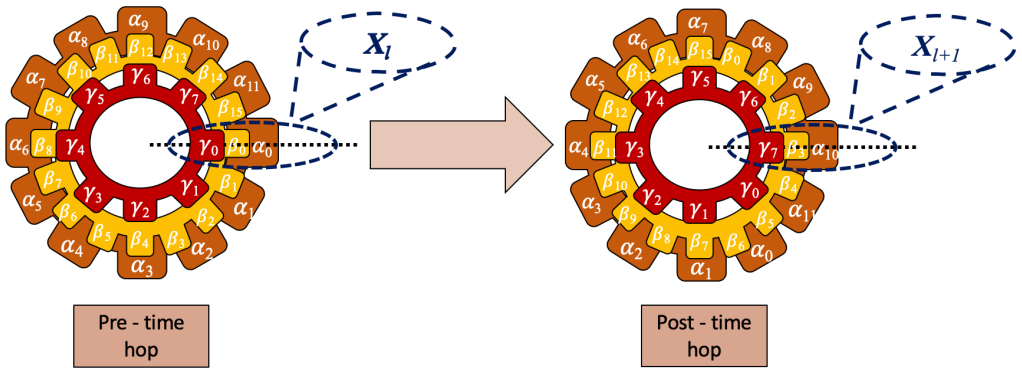

Figure 3.17: An example of time hopping, where α, β, and γ represent the original counter-clockwise orderings $\alpha_1, \alpha_2, \ldots \alpha_9$, etc. The original permutations are not modified during a time hop, but each permutation is essentially rotated, as shown by the gears.

3.6 Modified RNS Attack to Account for Noise

With the addition of noise, the attack algorithm described in Section 3 does not obtain the true initial state of the sequence generator every time. This section discusses the changes required for the algorithm to account for single and multiple errors, as well as the validation process.

This description discusses in greater depth the algorithm shown in Figure 3.18 and Figure

Algorithm 3: Modified RNS Attack for a Single Error

Input: k, L
$L \leftarrow \frac{\lceil \log_2 M \rceil}{k}$;
for L **to** *1* **do**
  if $L == \frac{\lceil \log_2 M \rceil}{k}$ **then**
    Calculate $X[l]$;
    Observe next v outputs;
    if *Initial State is Accurate* **then**
    end
  end
  Apply +1 offset to output at L;
  Calculate $X[l]$;
  Observe next v outputs;
  if *Initial State is Accurate* **then**
  end
  Apply −1 offset to output at L;
  Calculate $X[l]$;
  Observe next n outputs;
  if *Initial State is Accurate* **then**
  end
end

Figure 3.18: Modified RNS Attack for a Single Error

3.19: the modified RNS attack when a single error is present. On the start of the algorithm, L is set to $\frac{\lceil \log_2 M \rceil}{k}$. When L is ≈ 1, this is the lowest dimension or last output value recorded that is required to reverse engineer $X[l]$. On the first iteration of the for-loop, since this is the lowest dimension, the deterministic case is tested. The algorithm calculates $X[l]$ without applying any offset to an output. Following, it executes a validation subroutine where it seeds their sequence generator with the calculated state, then compares n additional outputs with the calculated to a desired assurance. Given a single wrong output, the calculated state is wrong. Following, it attempts to find the error by inverting the offset. It doesn't matter which path is tested first, but for this algorithm, a +1 offset is tested first. The algorithm

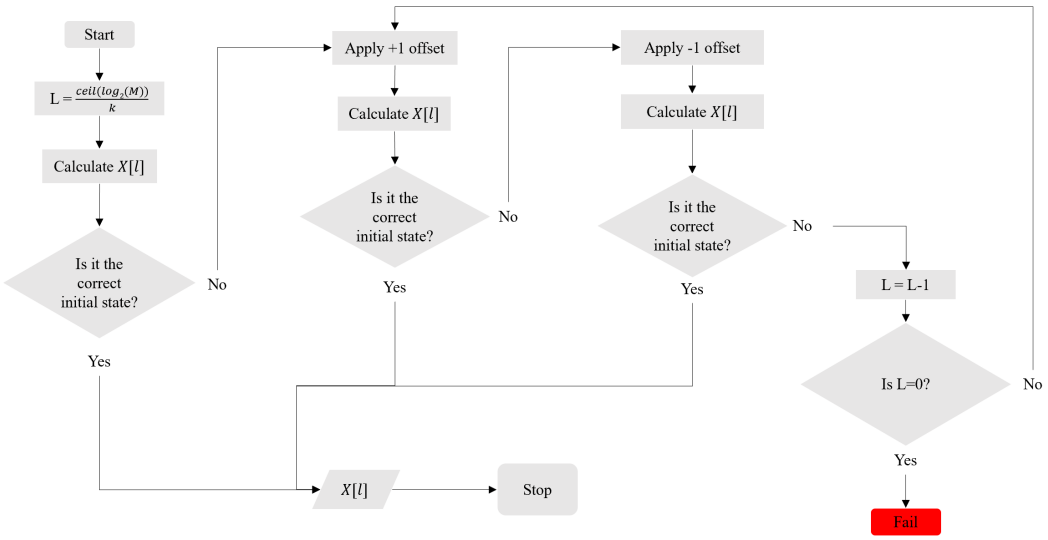

Figure 3.19: A flow diagram showing the modified RNS attack algorithm when a single error is present.

then calculates $X[l]$ given an adjusted output and performs the same validation subroutine. Given a wrong state, the algorithm then tests a -1 offset. If neither of these result in the correct state, the algorithm steps up a dimension to the previous output and repeats the offset tests. Refer to Figure 3.14 for a visual of the single-error output possibilities. If found to be false, the algorithm steps up another dimension L and repeats these steps until it reach the highest dimension. This always return the correct initial state.

In the deterministic case, the search space at any dimension L can be calculated as $\frac{M}{2^{kL}}$. Based on the modifications to the attack algorithm, the addition of noise triples the search space for every dimension at and below the modified output in the sliding window. A maximum of $3L$ outputs must now be considered. In the case of multiple errors per window, the attack algorithm must be modified again. As depicted in Figure 3.15, as each output has the possibility of a ± 1 offset applied, so do each of these branches. In this case, the algorithm described above would not always guarantee the correct answer. In this example, it could be a ternary tree with height L, giving us a maximum of $\frac{3^{L+1}-1}{2}$ cases to check.

The defender can adjust a variety of parameters to increase the complexity of reverse engineering attempts. The strength of noise, m, can also be increased. In a single-error, case the attacker would need to check mL sets, $\frac{m^{L+1}-1}{2}$ in the multi-error case. As k decreases, capturing additional outputs have less of an effect on reducing the overall search space, requiring the attacker to increase L and observe additional values, v, to validate the output.

The result can be validated by calculating additional sequence outputs past the window captured and comparing them with the received. The accuracy of each additional output can be treated as an independent event as the sequence generator has maximal entropy. The attacker can seed their sequence generator with the calculated initial state and generate their own outputs, comparing them with the received. After observing v additional outputs that equal theirs, the accuracy can is $1 - \frac{1}{2^k}^v$.

3.7 Analysis of RNS-Based PRNG Defenses

We aim to measure the effectiveness of the code hopping PRNG defense technique introduced in Section 5.2. The attack presented in Section 3 is altered by the change in permutations during a code hop, and we cannot use the traditional Kolmogorov entropy to determine the quality of a code-hopping RNS system. Therefore, we revert to other methods to quantify the quality of a RNS system with code hops. In this section, we establish formulae that measure the effectiveness of a code hopping RNS system against the reverse engineering attack in Section 3 in terms of time complexity and memory.

We first compare the time complexity for the designer to that of the attacker (ν_{time}). This is done by a ratio of the implementation times for both perspectives. The benefit of implementing a RNG system with r code hops is therefore represented by

$$\mu_{time} \approx \frac{\text{RNG processing}}{\text{Reverse Engineering processing}} \propto \frac{(r+1)\sum_{i=0}^{n-1} p_i}{(r+1)\Pi_{i=0}^{n-1} p_i} = \frac{\sum_{i=0}^{n-1} p_i}{M}. \tag{3.11}$$

A small ratio μ_{time} reveals that the computation complexity required to attack the system far exceeds the complexity required to create such a system, and thus this describes an efficient code hopping implementation in preventing the attack described in Section 6. Simply by increasing the number of primes in $\mathcal{P}$, the computation burden shifts significantly towards the attacker, since they can no longer amortize a one time search over all operations.

Next, we quantify the quality of an RNS system based on memory requirements on the designer vs. attacker. The memory requirement for the designer is given by ζ_{mem} in Section 4.2. On the other hand, maintaining the structure of the reverse engineering process for the attacker requires

$$\nu_{\text{mem}} = 2^{kL} * \frac{M}{2^{kL}} * \lceil \log_2 M \rceil = M * \lceil \log_2 M \rceil \text{ bits.} \tag{3.12}$$

Thus, the efficiency of the memory requirement benefits for the system designer is a ratio of the attack vs. defense memory requirements, given by

$$\mu_{\text{mem}} = \frac{2 * \sum_{i=1}^{n} p_i * \lceil \log_2(\max \mathcal{P}) \rceil}{M * \lceil \log_2 M \rceil}. \tag{3.13}$$

We test this ratio on three examples. The results of this analysis are displayed in Table 2. The first example is based on the toy example in Section 5.2. Since the primes were small and not many primes were used, the memory requirements of the attacker and designer are both relatively low. The example with $n = 4$ is based on the example in Section 5.3 with four larger prime values. Increasing the values of the primes raises the memory requirement

of the attacker. However, when generated, the attacker only needs 18 additional seconds of computation to find the initial state of the seq. generator. We finally test the system on a more realistic implementation with 16 large prime values. The memory requirement is much larger as opposed to the small increase in ζ_{mem}, and μ_{mem} decreases to almost zero. This implies that small increases in n and $\max(\mathcal{P})$ lead to large decreases in μ_{mem}, and the memory requirement of the attacker far exceeds the slight increase of memory required for a designer to implement a system with r code hops.

Table 3.2: Analysis of Time and Memory Requirements of Varying-Sized RNS.

n	$\max \mathcal{P}$	ζ_{mem}	ν_{mem}	μ_{mem}	$\mu_{\text{time},r=100}$
3	11	132	1,216	0.1	≈ 20.2
4	479	25,893	4.9×10^{11}	5.3×10^{-8}	$\approx 1.29 \times 10^{-5}$
16	1021	242,038	5.7×10^{49}	4.2×10^{-45}	$\approx 2.73 \times 10^{-33}$
24	2003	126,229	7.66×10^{61}	1.65×10^{-57}	$\approx 1.26 \times 10^{-53}$

We have shown in this analysis that the time complexity and memory requirement for the attacker exceeds that of the designer. Though the smaller examples such as those demonstrated in Section 5 do not show much difference between the attacker's and designer's time and memory specifications, larger realistic examples show that the attacker requires much more memory and time to reverse engineer a system than a designer requires to implement it. As the designer of an RNS system, the best strategy is to employ a large number of small primes in addition to code hopping. Executing code hops can be utilized as a method of complicating the reverse engineering process introduced in Section 3, and therefore this defense technique has the potential to deter an attacker from attempting the attack.

3.8 Conclusions

Through this chapter, we introduced a PRNG, based on a residue number system construction, with applications in TRANSEC, cryptography, and other types of secure communications. Examples show a non-uniform distribution of conditional output frequency data, highlighting vulnerabilities in the system. We use these frequencies to model an attack to reverse engineer the PRNG, assuming system knowledge and access to output data. Due to the multi-dimensional nature of this attack, we explain this attack as a corner turning algorithm, in which each turn represents a higher dimension reached and a large reduction in the search space. Multiple examples were implemented to clarify this process.

Though the attack makes strong assumptions about the candidate attacker's knowledge of the underlying RNS generation methods, such an approach is consistent with hands-on exploitation of a hardware system implementing the RNS algorithm. The reverse engineering method reduces the brute force search space by a factor of 2^{Lk} with every higher dimension reached within the algorithm. Tradeoffs are presented based on RNS size (one-time brute force processing increasing with M, stored memory allocations on the order of 2^{kL}) and associated processing time for initial and subsequent reverse engineering of newly programmed states. We validated this phenomenon by implementing the attack on a $>$ 32-bit example, which is representative of keys in use with many IoT applications.

Work is continuing to identify better ways to solve this series of equations, with the potential for reduced resolution sequences (small k, large L) expected to offer a rapid method for reducing the search of potentially valid initial states. One such concept is a quantum computer that can *search* these offsets simultaneously, while others are being further explored. In general, we anticipate each increase in L to reduce the overall search space by 2^k, suggesting a residual search space of size $\approx \frac{M}{2^{Lk}}$.

Future work encompasses utilizing these techniques in the presence of noise or with multi-stage RNS generators [157]. Noise, or uncertainty in outputs, obscures expected frequency data and places a limit on the dimensions available in our analysis. Even the natural occurrence of noise can restrict our view of the sequence generator output, consequently impairing reverse engineering attempts. Looking past RNS-based systems, the deliberate addition of noise to LSFRs can increase the security to the point of fooling algorithms like Berlekamp-Massey. Such systems could be implemented as cheap, efficient security in low-power IoT devices.

Chapter 4

Shannon Entropy Loss in Mixed-Radix Conversions

4.1 Introduction

The residue number system (RNS), initially proposed in 1959, was derived from the Chinese Remainder Theorem in the third century [158]. RNS architectures now are applied in many growing fields such as cryptography [100], image processing systems [159], and error-correction codes [160] 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 co-prime number bases, which we refer to as mixed-radix (MR) [161].

Research in MR calculations in RNS implementations has increased due to applications of circuits in which radix choice affects their speed, power, and area [162]. 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 consideration of another radix. For example, this distinction is apparent in Reed-Muller expansions over Galois fields involving cryptographic circuits [163]. In this application, using a lower-order radix usually requires less computation, which decreases the circuit's area, but this technique allows for an increased power consumption due to the large

amount of interconnections. On the other hand, choosing a higher radix will decrease power consumption, but increase circuit area [164]. In the case that an optimal radix is not able to be found, it is common to convert pseudorandomly generated words from one number base to another [165]. It is important to examine efficient ways to use MR techniques in processes like Reed-Muller expansions. Researchers thus have worked on independent MR conversion algorithms and techniques that have short conversion times [162, 166].

MR techniques have also been used for Fast Fourier Transform (FFT) pruning, which is designed to improve computational efficiency [167, 168]. These algorithms have conventional applications, such as recording the flicker of voltage in smart homes [169]. 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, using higher radices reduce latency for the memory-shared type of FFT architecture [170]. This additional use of MR conversions has prompted additional research on MR [168, 171, 172].

Most digital logic and computing systems are base-2, yet algorithms like Cooley-Tukey [173] offer equivalent mixed radix representations to achieve the same overall calculation, yet as a parallel composition of many small RNS operations. In particular, most PRNG methods are designed to produce bi-state binary values, and methods to quantify PRNG output randomness usually require their binary form [174]. 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) [175], the Mersenne Twister [176], and thermal entropy-based true random number generators (TRNG) [177]. As such, the primary source domain for RNG values is typically on $\mathbb{Z}_{2^k}$, while other algorithms consuming the random words may wish for non-binary random words. A specific example, that 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. [178] developed a lower bound on the Shannon entropy loss from such a mapping between an RNS-based source domain onto a $\mathbb{Z}_{2^k}$ target domain, 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. These calculations will be referenced later to choose the correct parameter to reduce computations in our chosen application of card shuffling.

Based on 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 to decrease the time it takes to shuffle cards [179]. Card-shuffling machines that randomize one deck while another is in use removes the time lost and possibility of fraudulent shuffles by a human dealer. These machines, which usually utilize a simple riffle shuffle, are rented for about $500 for each machine every month [180]. 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 [181], though use of physics-based true RNGs can mitigate that risk.

Many prior works have explored testing the quality of a card shuffle. More generally, card randomization is a popular problem in many fields like statistics, combinatorics, and communications [182]. Markov proved results that analyzed card shuffles as early as 1906 using finite Markov chains [183]. More rigorous methods have also been introduced, such as those using Fourier analysis and quantifying the entropy loss of riffle shuffling [184]. More recently, statistical tests for randomness, such as the FSU DIEHARD suite [69], are used to quantify the randomness of a permutation [185]. Additionally, entropy formulations, such as fiber entropy, have been utilized to measure shuffle output randomness [186]. In our application,

we will determine the quality of the shuffle based on the Shannon entropy loss of the mapping process within the shuffle as well as 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 will always provide a result with the highest entropy, utilizing PRNGs in these applications may be feasible, as well as 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 4.2, followed by a complete derivation of Shannon entropy loss in Section 4.3. A simulation model is then presented in Section 4.3.1, followed by overall conclusions in Section 4.5.

4.2 Shuffler Algorithm

To begin, we introduce a card shuffling algorithm characterized by modulo arithmetic on RNG outputs and recurrent Fisher-Yates permutations [25]. The system, illustrated in figure 4.1, shuffles a standard card deck as an ordered array $\mathcal{C} = [1|2|\dots|52]$ by repeated Fisher-Yates-based shuffles that utilize random numbers derived from any base-2 RNG.

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 4.2 demonstrates how X is framed in this step.

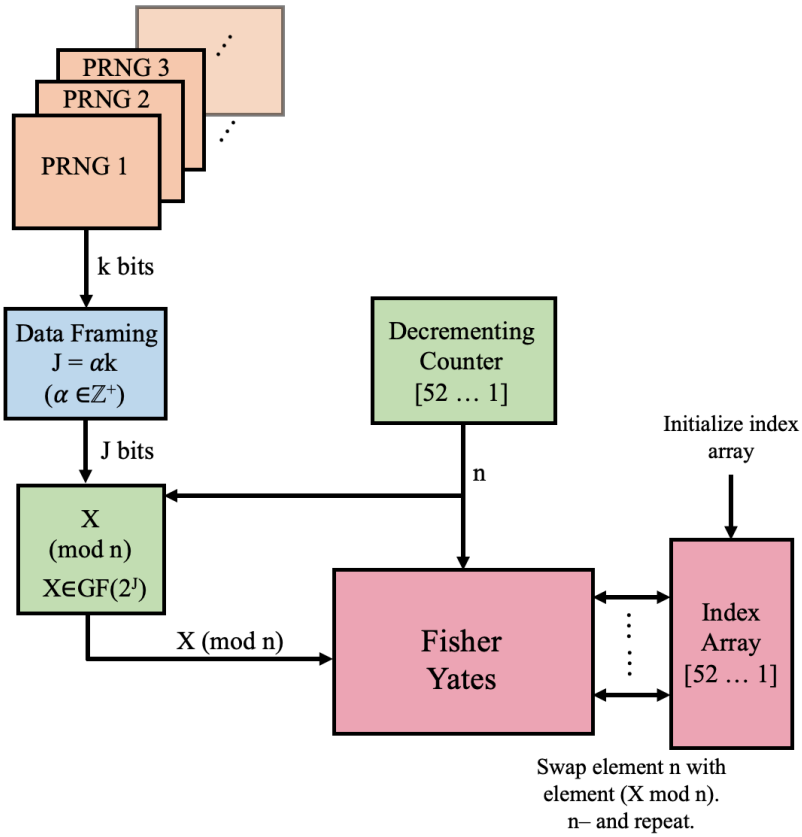

Figure 4.1: Shuffler Diagram

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, the deck $\mathcal{C}$ is fully shuffled.

One RNG may be better suited for this application than others. *Better*, in this case, 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 non-random shuffle. How-

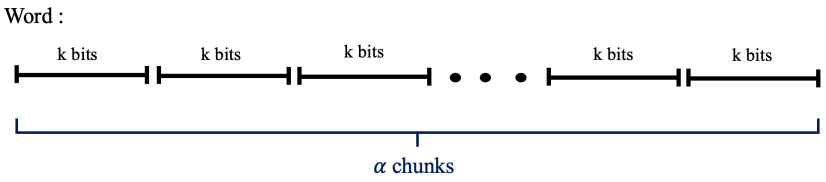

Figure 4.2: The data framing stage concatenates α words, each composed of k bits, to create a J-bit string.

ever, 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.

4.3 Mixed-Radix Conversion Entropy

The shuffler's entropy is not only lost through the PRNG algorithms that produce random input numbers, but it is also lost through the onto mapping process of the mixed-radix conversion. In this section, we provide background on Shannon entropy, the metric we use 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

$$H = -\sum_{i=1}^{L} p_i \log_n p_i. \tag{4.1}$$

We utilize Equation 4.1 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 4.3.

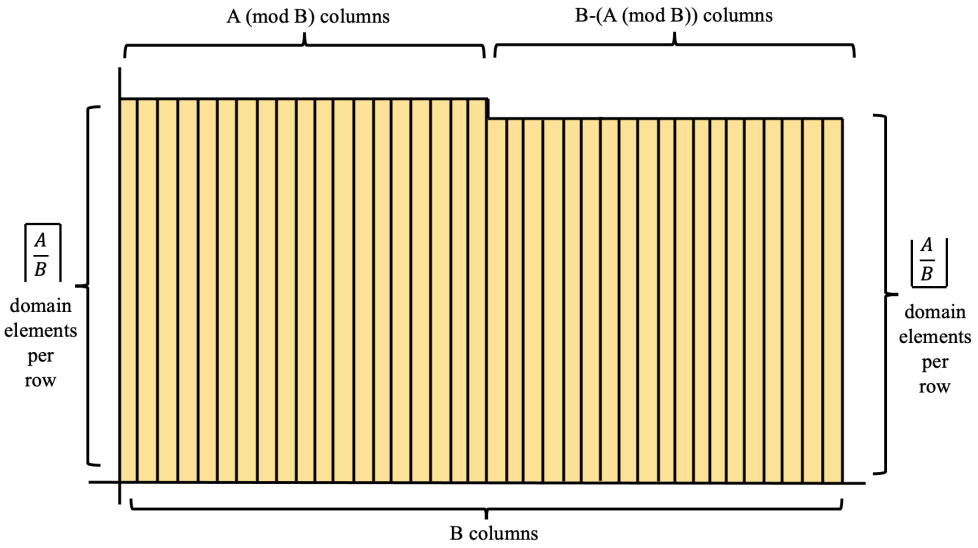

Figure 4.3: A histogram of residuals resulting from the surjective mapping $\mathcal{A} \to \mathcal{B}$, where $|\mathcal{A}| = A$ and $|\mathcal{B}| = B$.

Splitting up the histogram based upon column height, we can calculate the Shannon entropy of the onto map $\mathcal{A} \to \mathcal{B}$. From this splitting, we tailor the Shannon entropy formula in Equation 4.1 to obtain

$$\begin{aligned} H &= -(A \pmod B) \left[\frac{1}{A}\left\lceil\frac{A}{B}\right\rceil \log_B \frac{1}{A}\left\lceil\frac{A}{B}\right\rceil\right] \\ &\quad -(B - A \pmod B) \left[\frac{1}{A}\left\lfloor\frac{A}{B}\right\rfloor \log_B \frac{1}{A}\left\lfloor\frac{A}{B}\right\rfloor\right]. \end{aligned} \tag{4.2}$$

Further, define the entropy loss for the onto map $\mathcal{A} \to \mathcal{B}$ as shown in Equation 4.3.

$$\gamma_{A \to B} = 1 - H(A, B). \tag{4.3}$$

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 $0 \leq \gamma_{A \to B}\ leq1$. We will see the application of Equation 4.3 later in this paper, where entropy will be calculated for mappings in which $A = 2^J$ and B ranges from 2 to 52. Calculating these values is as simple as applying an adaptation of Equation 4.2.

4.3.1 Sources of Entropy

Entropy is not only lost through the shuffler's onto mapping process (discussed in Section 4.3) but randomness is also lost through the chosen input RNG. Even though utilizing a TRNG in lieu of a PRNG or lightweight primitive will provide the highest 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, low power, and cheap. We implement 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 repeatedly is 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 the 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 has justified their use in cryptographic and statistical applications [187]. We use the MATLAB 'mrg32k3a' variant, which has a period of 2^{191} [188].

- *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 based on the Fibonnaci sequence. A Multiplicative LFG (MLFG) will use multiplication as the operation in question [189]. We use MATLAB's 'mlfg6331_64,' whose period is 2^{124} [188].

- *Mersenne Twister*: the default PRNG in simulation engines like Matlab and Python, the Mersenne Twister offers efficient speed and large repetition period [176]. We use the MATLAB 'mt19937ar,' which has a period of $2^{19937} - 1$, a Mersenne prime [190].

- 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 [191]. Output quality has been proven to pass randomness tests, but one must be careful in choosing some parameters [192]. We used 'mcg16807,' which has a period of $2^{31} - 2$ [188].

- Modified subtract with borrow generator (MSBG): MSBG was proposed in 1991 as an improvement to the lagged-Fibonacci PRNG [193], and later applied in the RANLUX generator created for particle physics simulations [194]. We used the MATLAB version 'swb2712," which has a period length of 2^{1492} [188].

After simulating the card shuffle using each of these RNGs, we will analyze the randomness of their outputs. We will 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.

4.3.2 Calculating an Optimal J

In this section, we seek to 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 done 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 based on the shuffles that we see. We employed a hybrid of these two perspectives, where we consider both the numerical Shannon entropy loss from the shuffle and the hardware utilized to implement the shuffle. The following subsections explore both of these approaches.

Monotonicity of $\gamma_{A\to B}$ when $A = 2^J$

We first visualized the entropy loss of the shuffler's onto mapping process to notice any pattern in the behavior of $\gamma_{A\to B}$ with domain sizes that are relevant in this chapter. 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 is a surface plot histogram, shown in Figure 4.4, of entropy loss values of $J = 6 - 64$ and $B = 2 - 52$. These values were calculated on a deciBel scale to account for their size.

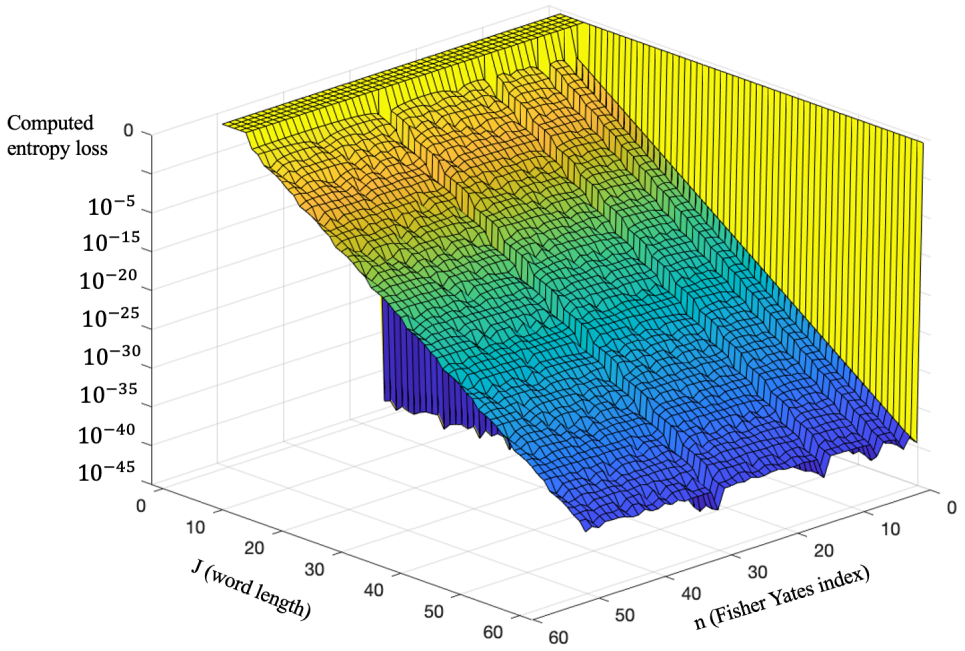

Figure 4.4: Entropy loss values as J ranges from 6-64, and as B ranges from 2-52.

First, note that the n values that are a power of 2 ($n = 4, 8, 16, 32, 64$) have entropy loss values of zero. This is because 2 raised to any power $J \geq 6$ modulo 2^r equals 0, and n automatically wraps around on an even basis, therefore creating zero residual. Visually, this plot appears to be monotonically decreasing with increasing J. We will prove this monotonicity, which is ultimately useful in our choice of J, since this statement will justify that choosing a larger value of J will always result in lower entropy loss.

The theorem and its proof quantifies the entropy loss, H_J and H_{J+1}, of a surjective map $\mathcal{A} \to \mathcal{B}$, where $|\mathcal{A}| = 2^J$ and $|\mathcal{B}| = n$ for an arbitrary value of J and the following value $J+1$. The monotonicity conclusion will be 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.

Theorem 4.1. *Entropy loss as a function of J for our chosen ranges of J, n is a monoton-*

ically decreasing function. Additionally, 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*

$$H_J = 1 + \frac{2x^3 - nx^2 - n^2x}{2n(2^J)^2} + \frac{n^3x - 3n^2x^2 + 4nx^3 - 2x^4}{2n(2^J)^3}. \tag{4.4}$$

Proof. Denote H_J the entropy of the onto map $\mathcal{A} \to \mathcal{B}$, where $A = 2^J$ and $B = n$. We show $H_{J+1} - H_J \geq 0 \forall n$.[1] Let $x = 2^J \mod n$. The ceiling and floor values that occur in the entropy value H are

$$\left\lceil \frac{2^J}{n} \right\rceil = \frac{2^J}{n} + \frac{(n-x)}{n}, \tag{4.5}$$

$$\left\lfloor \frac{2^J}{n} \right\rfloor = \frac{2^J}{n} - \frac{x}{n}. \tag{4.6}$$

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

$$2^{J+1} \pmod n = \begin{cases} 2x & x \leq \frac{n}{2} \\ 2x - n & x > \frac{n}{2} \end{cases} \tag{4.7}$$

$$\left\lceil \frac{2^{J+1}}{n} \right\rceil = \begin{cases} \frac{2^{J+1}}{n} + \frac{n-2x}{n} & x \leq \frac{n}{2} \\ \frac{2^{J+1}}{n} + \frac{n-(2x-n)}{n} & x > \frac{n}{2} \end{cases} \tag{4.8}$$

$$\left\lfloor \frac{2^{J+1}}{n} \right\rfloor = \begin{cases} \frac{2^{J+1}}{n} - \frac{2x}{n} & x \leq \frac{n}{2} \\ \frac{2^{J+1}}{n} + \frac{n-2x}{n} & x > \frac{n}{2} \end{cases} \tag{4.9}$$

[1] Note that this proof considers the general case of $2^J >> n$.

Then,

$$\begin{aligned} H_J &= -\frac{x}{2^J}\left(\frac{2^J+(n-x)}{n}\right)\log_n\left(\frac{2^J+(n-x)}{n2^J}\right) \\ &\quad -\frac{(n-x)}{2^J}\left(\frac{2^J-x}{n}\right)\log_n\left(\frac{2^J-x}{n2^J}\right). \end{aligned} \tag{4.10}$$

Using a degree-2 MacLaurin approximation for the logarithms,

$$\begin{aligned} H_J &= -\frac{x}{2^J}\left(\frac{2^J+(n-x)}{n}\right)\left(\log_n\left(1+\frac{n-x}{2^J}\right)-1\right) \\ &\quad -\frac{n-x}{n}\left(\frac{2^J-x}{2^J}\right)\left(\log_n\left(1-\frac{x}{2^J}\right)-1\right) \end{aligned} \tag{4.11}$$

$$\begin{aligned} &= -\frac{x}{2^J}\left(1+\frac{2^J-x}{n}\right)\left(\frac{n-x}{2^J}-\frac{(n-x)^2}{2(2^J)^2}-1\right) \\ &\quad -\frac{n-x}{n}\left(\frac{2^J-x}{2^J}\right)\left(-\frac{x}{2^J}+\frac{x^2}{2(2^J)^2}-1\right). \end{aligned} \tag{4.12}$$

This value simplifies to

$$H_J = 1+\frac{2x^3-nx^2-n^2x}{2n(2^J)^2}+\frac{n^3x-3n^2x^2+4nx^3-2x^4}{2n(2^J)^3}. \tag{4.13}$$

Equation 4.13 is a key simplification of H_J. This form will allow us to simplify values in the subtraction $H_J - H_{J+1}$. We break into two cases based on the piecewise nature of Equations 4.7-4.9 for the incremented case $J+1$.

Case 1 for H_{J+1}: $x \le \frac{n}{2}$. We use similar methods as we used in deriving equation 4.13 to determine

$$H_{J+1,x\le\frac{n}{2}} = 1+\frac{\frac{1}{2}n^3-\frac{3}{2}n^2x+nx^2}{2n(2^J)^2}+\frac{-\frac{3}{4}n^4+\frac{13}{4}n^3x-\frac{9}{2}n^2x^2+2nx^3}{2n(2^J)^3}. \tag{4.14}$$

With further simplification and subtraction, we have that

$$H_{J+1,x\leq\frac{n}{2}} - H_J = \underbrace{\frac{\frac{1}{2}n^3 - \frac{3}{2}n^2x + nx^2 - 2x^3 + nx^2 + n^2x}{2n(2^J)^2}}_{(*)} + \underbrace{\frac{-\frac{3}{4}n^4 + \frac{13}{4}n^3x - \frac{9}{2}n^2x^2 + 2nx^3 - n^3x + 3n^2x^2 - 4nx^3 + 2x^4}{2n(2^J)^3}}_{(**)} \tag{4.15}$$

(**) is arbitrarily small with practical bound on the order of $\frac{n^3}{(2^J)^3}$, where $\epsilon < \frac{n}{2}$. Consequently, we work with (*). We use $x = \frac{n}{2} + \epsilon$. Since ϵ is the residual between $\frac{n}{2}$ and x, $0 \leq \epsilon \leq \frac{n}{2}$. Then,

$$\text{numerator of (*)} = \frac{1}{2}(n^3 - n^2x + 4nx^2 - 4x^3) \tag{4.16}$$

$$= \frac{1}{2}\left(\frac{n^3}{2} - \epsilon^2 n - 2\epsilon^3\right). \tag{4.17}$$

So,

$$(*) = \frac{1}{4n(2^J)^2}\left(\frac{n^3}{2} - \epsilon^2 n - 2\epsilon^3\right). \tag{4.18}$$

Since $\epsilon \leq \frac{n}{2}$, $(*) \geq 0$. Thus, H_J is an increasing function as J increases. Consequently, the 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 4.14 is

$$H_{J+1,x>\frac{n}{2}} = 1 + \frac{-\frac{1}{2}n^2x - nx^2 + 2x^3}{2n(2^J)^2} + \frac{\frac{1}{4}n^3x - \frac{3}{2}n^2x^2 + 4nx^3 - 2x^4}{2n(2^J)^3} \tag{4.19}$$

We use simplification and subtraction similarly to Case 1 to reveal

$$H_{J+1,x>\frac{n}{2}} - H_J = \underbrace{\frac{-\frac{1}{2}n^2x - nx^2 + 2x^3 - 2x^3 + nx^2 + n^2x}{2n(2^J)^2}}_{(***)} + \underbrace{\frac{\frac{1}{4}n^3x - \frac{3}{2}n^2x^2 + 4nx^3 - 2x^4 - n^3x + 3n^2x^2 - 4nx^3 + 2x^4}{2n(2^J)^3}}_{(****)} \tag{4.20}$$

As in the first case, (****) is arbitrarily small on the order of $\frac{n^3}{(2^J)^3}$, and we work with (***). We apply the substitution $x = \frac{n}{2} - \epsilon$ and further simplifications. Note that ϵ still is the residual between $\frac{n}{2}$ and x, $0 \leq \epsilon \leq \frac{n}{2}$. We arrive at the analogue of equation 4.18,

$$(***) = \frac{1}{2n(2^J)^2}\left(\frac{n^3}{4} - \frac{n^2}{2}\epsilon\right). \tag{4.21}$$

Since $\epsilon \leq \frac{n}{2}$, $(***) \geq 0$, and the result follows. □

We end this section by emphasizing the importance of equation 6.1. We simplified the entropy loss formula in this case into a form that isolates the value 1. This value is relevant in that it represents the maximal entropy of H_J, and thus $0 \leq H_J \leq 1$.

4.3.3 Secondary Impacts

In this section, we discuss the 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 Theorem 4.1, entropy loss is monotonically decreasing in terms of J. We want J as large as possible, and because of the monotonic nature of entropy loss, there is no apparent benefit of mixing J values. Therefore, we will base our

choice of J on the maximum that the processor can handle, regardless of the current index value n.

Next, we consider the 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 will be 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 modulo operations in a linear sieve process using a base-2^m number system [195, 196].

In the sieve process, the J-bit word X is divided into $\frac{J}{2^m}$ smaller m-bit sub-words $X_{\frac{J}{2^m}-1}, \ldots, X_1, X_0$. These sub-words are used to compute $X \pmod n$ by

$$X = \left[\sum_{i=0}^{\frac{J}{2^m}-1} \left(X_i \pmod n \cdot 2^{m\cdot i} \pmod n \right) \right] \pmod n. \tag{4.22}$$

Note that each power of $2^{m\cdot i}$ modulo n can be pre-calculated and stored in memory to reduce extra calculations. Figure 4.5 shows a visual of how an m-bit processor calculates Equation 4.22.

Since the J-bit word X is broken down into m-bit fractional elements, a small processor will

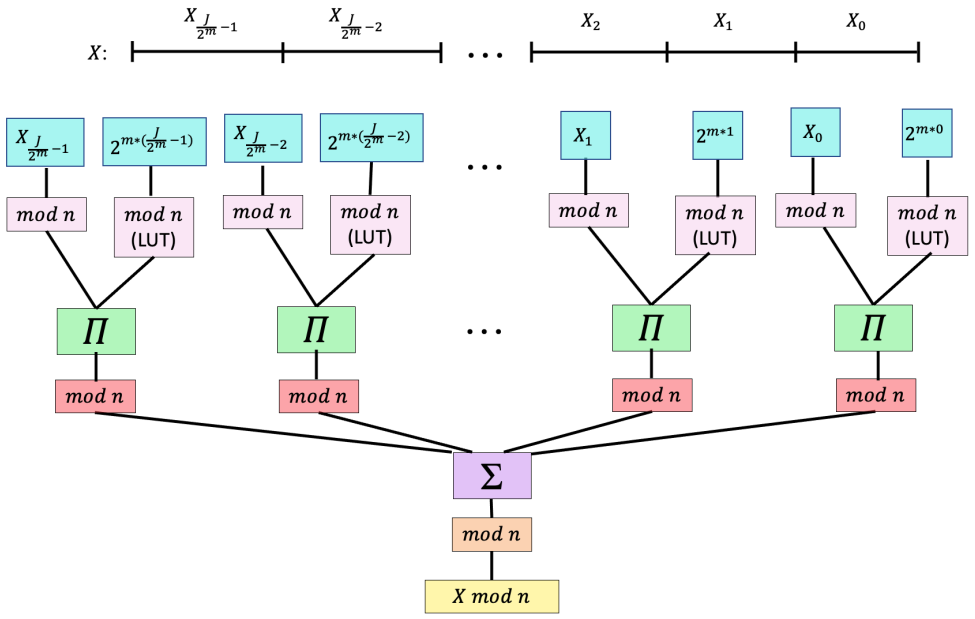

Figure 4.5: A visual representation of the linear sieve process of decomposing the calculation of J-bit binary word modulo n on a m-bit processor.

be able to compute each of the sub-words mod n to create a 6-bit or less output for each sub-word. Thus, even if $J = 128$, which far exceeds the entropy loss expectations, a 32-bit processor suffices since it can decompose the word into four sub-words. Also important in this modular reduction is recognition that sub-words have virtually no pairwise calculations, making them efficient on virtually any reduced precision process.

Combining these secondary impacts, we decide to utilize the values J equal to or larger than the processor size m in our application for $m \geq 32$. If a smaller processor size is utilized, we will use $J = 32$ or higher. Table 4.1 displays the minimum value of J that is recommended for each common processor size.

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

Table 4.1: Choice of J for the card shuffling implementations, based on processor size.

4.4 Prototype Shuffling Algorithm

We used MATLAB to simulate the shuffling algorithm using the PRNGs listed in Section 4.3.1. The code we utilized is outlined in Algorithm 4.6. The inputs to this algorithm include k and α as described in Section 4.2. 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 do assume 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 linear sieve process described in Figure 4.5. [Block 2] then calculates the random numbers by scaling the PRNG values r with its corresponding value from M. Finally, [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.

After simulating the shuffling algorithm with different PRNGs, we used a poker hand ranker to evaluate the output shuffle quality [197]. Although not an explicit measure of security, or *goodness*, this practical application of the shuffling algorithm enables testing of the results beyond esoteric entropy numbers. The number 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 7. The calculated expected frequencies show that even when the algorithm relies on moderate PRNGs, the output values are still randomized; weak

Algorithm 4: Shuffler Algorithm

Input: $k, \alpha, array$

$J \leftarrow k \cdot \alpha$, $X \leftarrow$ array of size 52, $r \leftarrow$ array of 52 zeros, $M \leftarrow 52 \times \alpha$ matrix of zeros, $n \leftarrow 1$.;

[Block 1]

for n *=1* **to** *52* **do**

  for i **to** α **do**

    $M \leftarrow$ 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)

Figure 4.6: Shuffler Algorithm

PRNGs such as the LFSRs do display an observable deviation from the combinatorics-based expected probabilities. A TRNG may be used in lieu of any of these methods, providing greater assurance of maximal entropy.

4.5 Conclusions

Due to the popularity of mechanical card shufflers in casinos, there has been interest in creating real-time shuffling implementations utilizing lightweight primitives like PRNGs.

Poker Hand	Expected	Pseudorandom Number Generator						
		mt19937ar	mrg32k3a	mlfg6331_64	mcg16807	swb2712	LFSR-16	LFSR-24
Straight Flush	153.9	149	146	156	151	155	0	611
Straight	39246.5	39261	39355	39159	39351	39172	40897	46987
Four of a Kind	2401.0	2389	2400	2397	2395	2374	1221	1222
Full House	14405.8	14220	14387	14382	14558	14301	4883	7324
Flush	19654.0	19706	19417	19514	19730	19926	59814	56765
Three of a Kind	211284.5	212045	211264	212396	211633	210891	156260	147093
Two Pair	475390	474767	475219	474461	476083	474606	391231	406509
Pair	4225690	4226512	4223350	4225677	4223706	4225301	4042922	4083281
High Card	5011774	5010951	5014462	5011858	5012393	5013274	5302772	5250208

Figure 4.7: The number of events of nine poker hands out of 10 million runs for each PRNG tested.

We introduced a card shuffling algorithm that, based on 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 [178] for the 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 have 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 7. 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 will build a hardware prototype of this shuffling engine, including incorporation of a TRNG. Applying the 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 will be suitable for immediate incorporation in electronic games (e.g., video poker, slots, etc), which may require different number bases, and adaptation to mechanical shufflers.

Chapter 5

Conclusions and Future Work

This thesis explored two applications of lightweight cryptography primitives. The goal of these applications is to design cheap, but effective, protocols that ideally can be implemented on low-power devices. Chapter 3 focused on a RNS-based PRNG, proposed by Michaels et. al [165], which passes multiple statistical tests for randomness, deeming it "secure" by many metrics. However, examples revealed a non-uniform distribution of conditional output frequency data, suggesting weaknesses in the system given prior knowledge of the RNS characteristics. Utilizing these frequencies, we created an attack to reverse engineer the single-stage PRNG, assuming system knowledge and access to output data. We then were able to implement this system and the attack on a $>$ 32-bit IoT-caliber example in MATLAB, which showed that the attack reduces the brute force search space by a factor of 2^{Lk} with every higher dimension reached within the algorithm, provided prior exhaustive calculation of the sequence and its patterns. This thesis then switched perspectives from attacker to designer and proposed three defenses to protect against the reverse engineering algorithm: deliberate noise injection, time hopping, and code hopping. These defenses slightly modify the original system's design by adding a dynamic nature to the key and/or generation process. The thesis then analyzes the time and memory requirements for both the attacker and designer in each scenario. From this analysis, we concluded that the time and memory required to attack the system in the presence of defenses was considerably greater than that of the designer, and therefore, for large examples, the modified system is resistant

to the proposed reverse engineering attack.

Chapter 4 shifts focus from PRNG security to MR calculations in RNS implementations. Since RNS is commonly utilized to reduce the computational load in applications with complex calculations, the thesis devised a card-shuffling algorithm that incorporates RNS in a Fisher-Yates shuffling process. Previous work provided an upper bound for the Shannon entropy loss of the surjective mapping process from a domain of size 2^J to a set of arbitrary size n. However, this result did not allow for variability in the target radix. The thesis devised a formulation for the Shannon entropy loss for this mapping that was useful in several ways. First, this formula was used to prove the motonicity of the Shannon entropy loss in increasing J. Secondly, this formulation was essential in our selection of the optimal parameters to simulate the shuffling algorithm with different test PRNGs. We later implemented this simulation on MATLAB to demonstrate a sample use case suitable to low-power implementation in a casino. Then, we compared the PRNGs in the shuffling process and quantified the effect of using a low power PRNG on the entropy of the combined operation by tabulating the frequencies of poker hands out of 10 million runs for each PRNG. The resulting frequencies demonstrated that most low-power PRNGs could produce a reasonable amount of randomness for the casino card shuffling applications.

The work done in both projects has an impact in the areas described in Chapter 2. The RNS PRNG analysed in Chapter 3 is an addition to the lightweight cryptography primitives that are currently being developed in response to the rise of IoT. Since no standard lightweight cryptography algorithm has been announced at this time, it is important to develop and analyze many cryptogarphic primitives for different applications. The PRNG in Chapter 3 was built for Transmission Security (TRANSEC), whose goal is to decrease the probability of interception, which is different than proving its security. This results of this thesis strengthened the system for this goal by suggesting modifications that increase the time and

memory requirements and reduce the probability of implementing a specific attack. Chapter 3 also demonstrates one example of a PRNG that, though it has previously been shown to pass multiple statistical tests for randomness, still is subject to an attack. This thesis thus adds to the current work done to prevent statistically well-performing algorithms from reverse engineering attacks.

The thesis adds to previous work on non-security related applications of RNS and PRNG. Non-binary 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 non-binary 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. This thesis adds to these mixed radix and non-binary applications, especially in games and combinatorial problems because it is common for the number base to fluctuate throughout these processes. Equation 4.13, 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 the radix, n. Thus, the Shannon entropy loss can be calculated for multiple radix values in applications that utilize this mapping process.

The thesis additionally explored PRNG use in the card-shuffling applicaiton in casino games. Casinos benefit from relying on lightweight primitives and RNS implementations due to the lower cost and less processing requirements compared to employing high power RNGs such as TRNGs. However, a lower-level of security is still necessary for casinos, and it is necessary to weigh between cost-reducing methods and better quality randomness. Through the poker hand analysis conducted in Chapter 4, the thesis utilizes Equation 4.13 to show that some low-power PRNGs are suitable for for casinos to utilize in card shuffling applications. With future work, these results can also be expanded to incorporate more PRNGs and encompass

more game and combinatorial applications.

5.1 Reccomended Future Directions

There are many potential avenues for future work in both projects. In Chapter 3, the thesis created an attack to reverse engineer the single-stage PRNG, assuming system knowledge and access to output data. A subsequent step is to extend the defenses of intentional noise, time hopping, and code hopping to encompass multi-stage RNS generators [157]. Noise obscures expected frequency data and limits the dimensions available for analysis. By physically implementing the PRNG with its defense modifications, future researchers will reveal the security of these defenses in a IoT-caliber example. Research can expand from here by testing different parameters in each defense mechanism. It would be interesting to evaluate noise injection with different types of noise, such as natural noise created from the hardware itself, or other sources such as burst noise [198] and coupled noise [199]. Additionally, future research can optimize the parameters in both the code hopping and time hopping defenses.

Future work involves extending these defenses in other PRNGs. One PRNG of interest is the linear feedback shift register (LFSR), which is a reasonable choice for lightweight cryptography applications due to its speed and security if created carefully. The polynomial-based code on a binary space repeatedly replaces its output sequence by an exclusive-or (XOR) of its last state. LFSR has the drawback of creating cyclic patterns because it conducts linear operations on a finite set of states, but the algorithm can produce a long cycle, or period, if the polynomial and input sequence is chosen correctly. LFSR is vulnerable to some attacks, such as the Berlekamp–Massey algorithm [200], but the deliberate addition of noise to LFSRs can potentially increase its security to resist against Brelekamp-Massey. The goal of this potential project would be to implement LFSR with a noise injection modification

as a cheap and efficient method of security in IoT-caliber devices.

Additionally, a mathematical analysis can be done to determine a method of computing the expected L-dimensional frequency data. The proposed attack in Chapter 3 relies on the assumption that these expected frequencies, i.e. the amount of transitions of the form $Z_l[i], Z_l[i+1], \ldots, Z_l[i+L]$ for an integer value L, are known. This thesis did not analyze how to obtain these frequencies, and a mathematical exploration of this idea is necessary to implement the attack without this assumption.

Work is still required to ensure the RNS PRNG's resistance to more attacks than just the reverse engineering attack explored in this paper. Since the PRNG was designed for transmission security (TRANSEC), defending this RNG from other potential attacks will strengthen the system for its application by further decreasing the probability of a reverse engineering success.

We now list the future directions related to Chapter 4. Future steps of the work in this chapter include building a hardware prototype of the shuffling algorithm. The hardware implementation will further continue the objective of this chapter to provide a low-cost and efficient device for casino card-shuffling applications. In building the shuffle, it would be interesting to use an optimal recognition system tied to machine learning to relocate each card in the deck. This can be done with tool for computer vision applied to playing cards.

It is of interest to extend the results of Chapter 4 to other mixed-radix applications such as combinatorical problems, representation of different unit systems, and lottery games. In these scenarios, it is important to convert formulas to input different radix values, especially those that help others understand the entropy loss of these games and problems. Equation 4.13 is important in that it describes a relatively universal mapping process, and can be utilized for other mixed-radix applications.

Finally, work is needed to continue the poker hand analysis to include more PRNGs and especially a TRNG. Including analysis with the TRNG will truly manifest the differences in relying on PRNG in lieu of TRNG for this application as there is no Shannon entropy loss involved with utilizing truly random inputs. Utilizing TRNG for the shuffling algorithm therefore reduces the system's entropy loss to only one source: the Fisher-Yates shuffle. Completing the poker hand analysis with more PRNGs will allow future researchers to assess more options for implementing the shuffler in a casino. The goals of this direction are to produce a massive Monte-Carlo simulation with many PRNGs and to discuss which PRNGs best balance the tradeoff between cost-effectiveness and randomness.

Chapter 6

Appendix

6.1 Proof Details

The appendix provides a step-by-step proof of Theorem 4.1. This version of the proof was moved to the appendix due to its length. Theorem 4.1 and its proof quantifies the entropy loss, H_J and H_{J+1}, of a surjective map $\mathcal{A} \to \mathcal{B}$, where $|\mathcal{A}| = 2^J$ and $|\mathcal{B}| = n$ for an arbitrary value of J and the following value $J+1$. It also proves the monotonicity of entropy loss as a function of J. The monotonicity conclusion is utilized in Chapter 4 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. We restate the theorem for the reader.

Theorem 6.1. *Entropy loss as a function of J for our chosen ranges of J, n is a monotonically decreasing function. Additionally, 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*

$$H_J = 1 + \frac{2x^3 - nx^2 - n^2x}{2n(2^J)^2} + \frac{n^3x - 3n^2x^2 + 4nx^3 - 2x^4}{2n(2^J)^3}. \tag{6.1}$$

Proof. Denote H_J the entropy of the onto map $\mathcal{A} \to \mathcal{B}$, where $A = 2^J$ and $B = n$. We show $H_{J+1} - H_J \geq 0 \forall n$. Let $x = 2^J \mod n$. Then, the ceiling and floor values that occur in the entropy value H are

$$\left\lceil \frac{2^J}{n} \right\rceil = \frac{2^J}{n} + \frac{(n-x)}{n}, \tag{6.2}$$

$$\left\lfloor \frac{2^J}{n} \right\rfloor = \frac{2^J}{n} - \frac{x}{n}. \tag{6.3}$$

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

$$2^{J+1} \pmod{n} = \begin{cases} 2x & x \leq \frac{n}{2} \\ 2x - n & x > \frac{n}{2} \end{cases} \tag{6.4}$$

$$\left\lceil \frac{2^{J+1}}{n} \right\rceil = \begin{cases} \frac{2^{J+1}}{n} + \frac{n-2x}{n} & x \leq \frac{n}{2} \\ \frac{2^{J+1}}{n} + \frac{n-(2x-n)}{n} & x > \frac{n}{2} \end{cases} \tag{6.5}$$

$$\left\lfloor \frac{2^{J+1}}{n} \right\rfloor = \begin{cases} \frac{2^{J+1}}{n} - \frac{2x}{n} & x \leq \frac{n}{2} \\ \frac{2^{J+1}}{n} + \frac{n-2x}{n} & x > \frac{n}{2} \end{cases} \tag{6.6}$$

Then,

$$\begin{aligned} H_J = & -\frac{x}{2^J}\left(\frac{2^J + (n-x)}{n}\right)\log_n\left(\frac{2^J + (n-x)}{n2^J}\right) \\ & -\frac{(n-x)}{2^J}\left(\frac{2^J - x}{n}\right)\log_n\left(\frac{2^J - x}{n2^J}.\right) \end{aligned} \tag{6.7}$$

We use the degree 2 MacLaurin approximation to simplify H_J.

$$
\begin{aligned}
H_J &= -\frac{x}{2^J}\left(\frac{2^J+(n-x)}{n}\right)\left(\log_n\left(1+\frac{n-x}{2^J}\right)-1\right) && (6.8)\\
&\quad -\frac{n-x}{n}\left(\frac{2^J-x}{2^J}\right)\left(\log_n\left(1-\frac{x}{2^J}\right)-1\right)\\
&= -\frac{x}{2^J}\left(1+\frac{2^J-x}{n}\right)\left(\frac{n-x}{2^J}-\frac{(n-x)^2}{2(2^J)^2}-1\right) && (6.9)\\
&\quad -\frac{n-x}{n}\left(\frac{2^J-x}{2^J}\right)\left(-\frac{x}{2^J}+\frac{x^2}{2(2^J)^2}-1.\right)
\end{aligned}
$$

We simplify equation (6.9) in the following manner.

$$
\begin{aligned}
H_J &= \left(\frac{x}{2^J}+\frac{x}{n}\frac{(2^J-x)}{2^J}\right)\left(1+\frac{(n-x)^2}{2(2^J)^2}-\frac{n-x}{2^J}\right) && (6.10)\\
&\quad +\left(\frac{n-x}{n}\cdot\frac{2^J-x}{2^J}\right)\left(1+\frac{x}{2^J}-\frac{x^2}{2\cdot(2^J)^2}\right)\\
&= \frac{x}{2^J}+\frac{x}{n}\frac{2^J-x}{2^J}-\frac{x(n-x)}{(2^J)^2}-\frac{x}{n}\frac{(2^J-x)}{2^J}\frac{(n-x)}{2^J} && (6.11)\\
&\quad +\frac{x}{2}\frac{(n-x)^2}{(2^J)^3}+\frac{x}{n}\frac{(2^J-x)}{2^J}\frac{(n-x)^2}{2(2^J)^2}\\
&\quad +\frac{(n-x)}{n}\frac{2^J-x}{2^J}+\frac{(n-x)(2^J-x)x}{n(2^J)^2}-\frac{(n-x)(2^J-x)x^2}{2n(2^J)^3}\\
&= \frac{1}{2n(2^J)^3}[2xn(2^J)^2+2x(2^J-x)(2^J)^2-2xn(n-x)(2^J) && (6.12)\\
&\quad -2x2^J(n-x)(2^J-x)+nx(n-x)^2+x(2^J-x)(n-x)^2\\
&\quad +2(2^J)^2(n-x)(2^J-x)+2(2^J)(n-x)(2^J-x)x\\
&\quad -(n-x)(2^J-x)x^2]
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{2n(2^J)^3}[2x(2^J-x)(2^J)^2 + 2xn(2^J)^2 - 2x(n-x)(2^J)n && (6.13)\\
&\quad +nx(n-x)^2 + x(2^J-x)(n-x)^2 - 2x2^J(2^J-x)(n-x)\\
&\quad +2(2^J)^2(n-x)(2^J-x) + 2(2^J)(n-x)(2^J-x)x\\
&\quad -(n-x)(2^J-x)x^2]\\
&= \frac{1}{2n(2^J)^3}[2x(2^J)^3 - 2x^2(2^J)^2 + 2xn(2^J)^2 - 2xn^2(2^J) && (6.14)\\
&\quad +2x^2n(2^J) + n^3x - 2n^2x^2 + nx^3 + n^2x2^J\\
&\quad -2nx^2 2^J + x^3 2^J - n^2x^2 + 2nx^3 - x^4\\
&\quad -2nx(2^J)^2 + 2x^2(2^J)^2 + 2nx^2 2^J - 2x^3 2^J\\
&\quad +2n(2^J)^3 - 2nx(2^J)^2 - 2x(2^J)^3 + 2x^2(2^J)^2\\
&\quad +2nx(2^J)^2 - 2nx^2(2^J) - 2x^2(2^J)^2 + 2x^3 2^J\\
&\quad -nx^2 2^J + nx^3 + x^3 2^J - x^4]\\
&= \frac{1}{2n(2^J)^3}[(2^J)^3(2x+2n-2x) && (6.15)\\
&\quad +(2^J)^2(-2x^2+2xn-2xn+2x^2-2nx+2x^2+2nx-2x^2)\\
&\quad +(2^J)(-2xn^2+2x^2n+n^2x-2nx^2+x^3+2nx^2-2x^3-2nx^2+2x^3-nx^2+x^3)\\
&\quad +(n^3x-2n^2x^2+nx^3-n^2x^2+2nx^3-x^4+nx^3-x^4)]\\
&= \frac{1}{2n(2^J)^3}[(2^J)^3(2n)+(2^J)^2(0)+(2^J)(2x^3-nx^2-n^2x)\\
&\quad +(n^3x-3n^2x^2+4nx^3-2x^4)]\\
&= 1+\frac{2x^3-nx^2-n^2x}{2n(2^J)^2}+\frac{n^3x-3n^2x^2+4nx^3-2x^4}{2n(2^J)^3}. && (6.16)
\end{aligned}
$$

Thus,

$$
H_J = 1+\frac{2x^3-nx^2-n^2x}{2n(2^J)^2}+\frac{n^3x-3n^2x^2+4nx^3-2x^4}{2n(2^J)^3}. \qquad (6.17)
$$

Equation (6.17) (Equation 4.13 in Chapter 4) is a key simplification of H_J. This form will allow us to simplify values in the subtraction $H_J - H_{J+1}$. We break into two cases based on the piecewise nature of Equations 4.7-4.9 for the incremented case $J+1$.

Case 1 for H_{J+1}: $x \leq \frac{n}{2}$. We use similar methods as we used in deriving equation (6.17) (Equation 4.13 in Chapter 4) in the following derivation.

We utilize equations (6.3), (6.4), and (6.6) to obtain

$$\begin{aligned} H_{J+1,x\leq\frac{n}{2}} &= -2^{J+1} \pmod n \left(2^{-(J+1)} \left\lceil \frac{2^{J+1}}{n} \right\rceil \log_n 2^{-(J+1)} \left\lceil \frac{2^{J+1}}{n} \right\rceil\right) \\ &\quad -(n - 2^{J+1} \pmod n) \left(2^{-(J+1)} \left\lfloor \frac{2^{J+1}}{n} \right\rfloor \log_n 2^{-(J+1)} \left\lfloor \frac{2^{J+1}}{n} \right\rfloor\right). \end{aligned} \tag{6.18}$$

Then, utilizing a degree-2 MacLaurin approximation for the logarithms,

$$\begin{aligned} H_{J+1,x\leq\frac{n}{2}} &= -(2x)\left(\frac{1}{2^{J+1}}\right)\left(\frac{2^{J+1}}{n} + \frac{n-2x}{n}\right)\log_n\left(\frac{1}{2^{J+1}}\left(\frac{2^{J+1}}{n} + \frac{n-2x}{n}\right)\right) \\ &\quad -(n-2x)\left(\frac{1}{2^{J+1}}\right)\left(\frac{2^{J+1}}{n} - \frac{2x}{n}\right)\log_n\left(\frac{1}{2^{J+1}}\left(\frac{2^{J+1}}{n} - \frac{2x}{n}\right)\right) \end{aligned} \tag{6.19}$$

$$\begin{aligned} &= -\frac{2x}{2^{J+1}}\left(\frac{2^{J+1}+n-2x}{n}\right)\log_n\left(\frac{2^{J+1}+n-2x}{n2^{J+1}}\right) \\ &\quad -\frac{n-2x}{2^{J+1}}\left(\frac{2^{J+1}-2x}{n}\right)\log_n\left(\frac{2^{J+1}-2x}{n2^{J+1}}\right). \end{aligned} \tag{6.20}$$

We simplify equation (6.20) in the following manner.

$$
\begin{aligned}
H_{J+1,x\leq\frac{n}{2}} &= -\frac{2x}{2^{J+1}}\left(\frac{2^{J+1}+n-2x}{n}\right)\left(\log_n\left(1+\frac{n-2x}{2^{J+1}}\right)-1\right) && (6.21)\\
&\quad -\frac{n-2x}{2^{J+1}}\left(\frac{2^{J+1}-2x}{n}\right)\left(\log_n\left(1-\frac{2x}{2^{J+1}}\right)-1\right)\\
&= -\frac{2x}{2^{J+1}}\left(1+\frac{2^{J+1}-2x}{n}\right)\left(\log_n\left(1+\frac{n-2x}{2^{J+1}}\right)-1\right) && (6.22)\\
&\quad -\frac{n-2x}{2^{J+1}}\left(\frac{2^{J+1}-2x}{n}\right)\left(\log_n\left(1-\frac{2x}{2^{J+1}}\right)-1\right)\\
&= -\frac{2x}{2^{J+1}}\left(1+\frac{2^{J+1}-2x}{n}\right)\left(\frac{n-2x}{2^{J+1}}-\frac{(n-2x)^2}{2(2^{J+1})^2}-1\right) && (6.23)\\
&\quad -\frac{n-2x}{2^{J+1}}\left(\frac{2^{J+1}-2x}{n}\right)\left(\frac{-2x}{2^{J+1}}+\frac{(2x)^2}{2(2^{J+1})^2}-1\right)\\
&= \left(\frac{2x}{2^{J+1}}+\frac{2x}{n}\frac{2^{J+1}-2x}{2^{J+1}}\right)\left(1+\frac{(n-2x)^2}{2(2^{J+1})^2}-\frac{(n-2x)}{2^{J+1}}\right) && (6.24)\\
&\quad +\left(\frac{n-2x}{n}\right)\left(\frac{2^{J+1}-2x}{2^{J+1}}\right)\left(1+\frac{2x}{2^{J+1}}-\frac{(2x)^2}{2(2^{J+1})^2}\right)\\
&= \frac{2x}{2^{J+1}}+\frac{2x}{n}\frac{2^{J+1}-2x}{2^{J+1}}+\frac{2x}{2^{J+1}}\frac{(n-2x)^2}{2(2^{J+1})^2}+\frac{2x}{n}\frac{2^{J+1}-2x}{2^{J+1}}\frac{(n-2x)^2}{2(2^{J+1})^2} && (6.25)\\
&\quad -\frac{2x}{2^{J+1}}\frac{n-2x}{2^{J+1}}-\frac{2x}{n}\frac{(2^{J+1}-2x)}{2^{J+1}}\frac{n-2x}{2^{J+1}}+\frac{(n-2x)}{n}\frac{(2^{J+1}-2x)}{2^{J+1}}\\
&\quad +\frac{(n-2x)}{n}\frac{(2^{J+1}-2x)}{2^{J+1}}\frac{2x}{2^{J+1}}-\frac{(2^{J+1}-2x)}{2^{J+1}}\frac{(2x)^2}{2(2^{J+1})^2}.\\
&= \frac{2x}{2^{J+1}}+\frac{2x}{n}\frac{2^{J+1}-2x}{2^{J+1}}+\frac{x(n-2x)^2}{(2^{J+1})^3}+\frac{x}{n}\frac{(2^{J+1}-2x)(n-2x)^2}{(2^{J+1})^3} && (6.26)\\
&\quad -\frac{2x(n-2x)}{(2^{J+1})^2}-\frac{2x(2^{J+1}-2x)(n-2x)}{n(2^{J+1})^2}+\frac{(n-2x)(2^{J+1}-2x)}{n2^{J+1}}\\
&\quad +\frac{(n-2x)(2^{J+1}-2x)(2x)}{n(2^{J+1})^2}-\frac{(2x)^2(2^{J+1}-2x)}{2(2^{J+1})^3}\\
&= \frac{1}{2n(2^{J+1})^3}[4xn(2^{J+1})^2+4x(2^{J+1})^2(2^{J+1}-2x)+2xn(n-2x)^2 && (6.27)\\
&\quad +2x(2^{J+1}-2x)(n-2x)^2-4xn(2^{J+1})(n-2x)\\
&\quad -4x(2^{J+1})(n-2x)(2^{J+1}-2x)+2(2^{J+1})^2(n-2x)(2^{J+1}-2x)\\
&\quad +4x(n-2x)(2^{J+1}-2x)(2^{J+1})-n(2x)^2(2^{J+1}-2x)]\\
& && (6.28)
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{2n(2^{J+1})^3}[2n(2^{J+1})^3 + (-2n^2x - 4nx^2 + 8x^3)2^{J+1} + (2n^3x - 12n^2x^2 \qquad (6.29)\\
&\quad +32nx^3 - 16x^4)]\\
&= 1 + \frac{-n^2x - 2nx^2 + 4x^3}{n(2^{J+1})^2} + \frac{n^3x - 6n^2x^2 + 16nx^3 - 8x^4}{n(2^{J+1})^3} \qquad (6.30)\\
&= 1 + \frac{-n^2x - 2nx^2 + 4x^3}{n(2J+1)^2} + \frac{n^3x - 6n^2x^2 + 16nx^3 - 8x^4}{n(2^{J+1})^3} \qquad (6.31)\\
&= 1 + \frac{-n^2x - 2nx^2 + 4x^3}{2(2n(2^J)^2)} + \frac{n^3x - 6n^2x^2 + 16nx^3 - 8x^4}{4(2n(2^J)^3)} \qquad (6.32)\\
&= 1 + \frac{-\frac{1}{2}n^2x - nx^2 + 2x^3}{2n(2^J)^2} + \frac{\frac{1}{4}n^3x - \frac{3}{2}n^2x^2 + 4nx^3 - 2x^4}{2n(2^J)^3}. \qquad (6.33)
\end{aligned}
$$

The fundamental question is if $H_{J+1} - H_J \geq 0$? With further simplification and subtraction, we have that

$$
\begin{aligned}
H_{J+1,x\leq\frac{n}{2}} - H_J &= \underbrace{\frac{\frac{1}{2}n^3 - \frac{3}{2}n^2x + nx^2 - 2x^3 + nx^2 + n^2x}{2n(2^J)^2}}_{(*)} \qquad (6.34)\\
&\quad + \underbrace{\frac{-\frac{3}{4}n^4 + \frac{13}{4}n^3x - \frac{9}{2}n^2x^2 + 2nx^3 - n^3x + 3n^2x^2 - 4nx^3 + 2x^4}{2n(2^J)^3}}_{(**)}
\end{aligned}
$$

(**) is arbitrarily small with practical bound on the order of $\frac{n^3}{(2^J)^3}$, where $\epsilon < \frac{n}{2}$. Consequently, we work with (*). We use $x = \frac{n}{2} + \epsilon$. Since ϵ is the residual between $\frac{n}{2}$ and x, $0 \leq \epsilon \leq \frac{n}{2}$. Then,

$$
\begin{aligned}
\text{numerator of (*)} &= \frac{1}{2}(n^3 - n^2x + 4nx^2 - 4x^3) \qquad (6.35)\\
&= \frac{1}{2}\left(\frac{n^3}{2} - \epsilon^2 n - 2\epsilon^3\right). \qquad (6.36)
\end{aligned}
$$

So,

$$(*) = \frac{1}{4n(2^J)^2}\left(\frac{n^3}{2} - \epsilon^2 n - 2\epsilon^3\right). \tag{6.37}$$

Since $\epsilon \leq \frac{n}{2}$, $(*) \geq 0$. Thus, H_J is an increasing function as J increases. Consequently, the 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}: $x > \frac{n}{2}$. We use similar methods as we used in deriving equation (6.17) (Equation 4.13 in Chapter 4) in the following derivation.

We utilize equations (6.2), (6.4), and (6.5) to obtain

$$\begin{aligned} H_{J+1,x>\frac{n}{2}} &= -2^{J+1} \pmod n \left(2^{-(J+1)}\left\lceil\frac{2^{J+1}}{n}\right\rceil \log_n 2^{-(J+1)}\left\lceil\frac{2^{J+1}}{n}\right\rceil\right) \\ &\quad -(n - 2^{J+1} \pmod n)\left(2^{-(J+1)}\left\lfloor\frac{2^{J+1}}{n}\right\rfloor \log_n 2^{-(J+1)}\left\lfloor\frac{2^{J+1}}{n}\right\rfloor\right). \end{aligned} \tag{6.38}$$

Then, utilizing a degree-2 MacLaurin approximation for the logarithms,

$$\begin{aligned} H_{J+1,x>\frac{n}{2}} &= -\frac{2x-n}{2^{J+1}}\left(\frac{2^{J+1}}{n} + \frac{(n-(2x-n))}{n}\right)\log_n\left(\frac{1}{2^{J+1}}\left(\frac{2^{J+1}}{n} + \frac{n-(2x-n)}{n}\right)\right) \\ &\quad -\frac{n-(2x-n)}{2^{J+1}}\left(\frac{2^{J+1}}{n} + \frac{n-2x}{n}\right)\log_n\left(\frac{1}{2^{J+1}}\left(\frac{2^{J+1}}{n} + \frac{n-2x}{n}\right)\right) \qquad (6.39) \\ &= -\frac{2x-n}{2^{J+1}}\left(\frac{2^{J+1}+2n-2x}{n}\right)\log_n\left(\frac{2^{J+1}+2n-2x}{n2^{J+1}}\right) \qquad (6.40) \\ &\quad -\frac{2n-2x}{2^{J+1}}\left(\frac{2^{J+1}+n-2x}{n}\right)\log_n\left(\frac{2^{J+1}+n-2x}{n2^{J+1}}\right). \end{aligned}$$

We simplify equation (6.40) in the following manner.

$$
\begin{aligned}
H_{J+1,x>\frac{n}{2}} &= -\frac{2x-n}{2^{J+1}}\left(2+\frac{2^{J+1}-2x}{n}\right)\left(\log_n\left(1+\frac{2n-2x}{2^{J+1}}\right)-1\right) && (6.41)\\
&\quad -\frac{2n-2x}{2^{J+1}}\left(1+\frac{2^{J+1}-2x}{n}\right)\left(\log_n\left(1+\frac{n-2x}{2^{J+1}}\right)-1\right)\\
&= -\frac{2x-n}{2^{J+1}}\left(2+\frac{2^{J+1}-2x}{n}\right)\left(\frac{2n-2x}{2^{J+1}}-\frac{(2n-2x)^2}{2(2^{J+1})^2}-1\right) && (6.42)\\
&\quad -\frac{2n-2x}{2^{J+1}}\left(1+\frac{2^{J+1}-2x}{n}\right)\left(\frac{n-2x}{2^{J+1}}-\frac{(n-2x)^2}{2(2^{J+1})^2}-1\right)\\
&= \left(\frac{-2(2x-n)}{2^{J+1}}-\frac{(2x-n)(2^{J+1}-2x)}{n2^{J+1}}\right)\left(\frac{2n-2x}{2^{J+1}}-\frac{(2n-2x)^2}{2(2^{J+1})^2}-1\right) && (6.43)\\
&\quad +\left(\frac{-(2n-2x)}{2^{J+1}}-\frac{(2n-2x)(2^{J+1}-2x)}{n2^{J+1}}\right)\left(\frac{n-2x}{2^{J+1}}-\frac{(n-2x)^2}{2(2^{J+1})^2}-1\right)\\
&= \frac{-2(2x-n)(2n-2x)}{(2^{J+1})^2}+\frac{2(2x-n)(2n-2x)^2}{2(2^{J+1})^3}+\frac{2(2x-n)}{2^{J+1}} && (6.44)\\
&\quad -\frac{(2x-n)(2n-2x)(2^{J+1}-2x)}{n(2^{J+1})^2}+\frac{(2x-n)(2^{J+1}-2x)(2n-2x)^2}{2n(2^{J+1})^3}\\
&\quad +\frac{(2x-n)(2^{J+1}-2x)}{n2^{J+1}}-\frac{(2n-2x)(n-2x)}{(2^{J+1})^2}+\frac{(2n-2x)(n-2x)^2}{2(2^{J+1})^3}\\
&\quad +\frac{(2n-2x)}{2^{J+1}}-\frac{(2n-2x)(n-2x)(2^{J+1}-2x)}{n(2^{J+1})^2}+\frac{(2n-2x)(2^{J+1}-2x)(n-2x)^2}{2n(2^{J+1})^3}\\
&\quad +\frac{(2n-2x)(2^{J+1}-2x)}{n2^{J+1}}\\
&= \frac{1}{2n(2^{J+1})^3}[-4n(2x-n)(2n-2x)2^{J+1}+2n(2x-n)(2n-2x)^2 && (6.45)\\
&\quad +4n(2x-n)(2^{J+1})^2-2(2x-n)(2n-2x)(2^{J+1}-2x)2^{J+1}\\
&\quad +(2x-n)(2^{J+1}-2x)(2n-2x)^2+2(2x-n)(2^{J+1}-2x)(2^{J+1})^2\\
&\quad -2n(2n-2x)(n-2x)2^{J+1}+n(2n-2x)(n-2x)^2+2n(2n-2x)(2^{J+1})^2\\
&\quad -2(2n-2x)(n-2x)(2^{J+1}-2x)2^{J+1}+(2n-2x)(2^{J+1}-2x)(n-2x)^2\\
&\quad +2(2n-2x)(2^{J+1}-2x)(2^{J+1})^2]
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{2n(2^{J+1})^3}[2n(2^{J+1})^3 + (2n^3 - 6n^2x + 4nx^2)(2^{J+1}) + (-6n^4 + 26n^3x \qquad (6.46)\\
&\quad -36n^2x^2 + 16nx^3)]\\
&= 1 + \frac{n^3 - 3n^2x + 2nx^2}{n(2^{J+1})^2} + \frac{(-3n^4 + 13n^3x - 18n^2x^2 + 8nx^3)}{n(2^{J+1})^3} \qquad (6.47)\\
&= 1 + \frac{n^3 - 3n^2x + 2nx^2}{n(2^{J+1})^2} + \frac{-3n^4 + 13n^3x - 18n^2x^2 + 8nx^3}{n(2^{J+1})^3} \qquad (6.48)\\
&= 1 + \frac{n^3 - 3n^2x + 2nx^2}{2(2n(2^J)^2)} + \frac{-3n^4 + 13n^3x - 18n^2x^2 + 8nx^3}{4(2n(2^J)^3)} \qquad (6.49)\\
&= 1 + \frac{\frac{1}{2}n^3 - \frac{3}{2}n^2x + nx^2}{2n(2^J)^2} + \frac{-\frac{3}{4}n^4 + \frac{13}{4}n^3x - \frac{9}{2}n^2x^2 + 2nx^3}{2n(2^J)^3}. \qquad (6.50)
\end{aligned}
$$

The fundamental question is if $H_{J+1} - H_J \geq 0$? We work with the $x > \frac{n}{2}$ case.

$$
\begin{aligned}
H_{J+1,x>\frac{n}{2}} - H_J &= \underbrace{\frac{-\frac{1}{2}n^2x - nx^2 + 2x^3 - 2x^3 + nx^2 + n^2x}{2n(2^J)^2}}_{(***)} \qquad (6.51)\\
&\quad + \underbrace{\frac{\frac{1}{4}n^3x - \frac{3}{2}n^2x^2 + 4nx^3 - 2x^4 - n^3x + 3n^2x^2 - 4nx^3 + 2x^4}{2n(2^J)^3}}_{(****)}
\end{aligned}
$$

As in the first case, (****) is arbitrarily small on the order of $\frac{n^3}{(2^J)^3}$, and we work with (***). We apply the substitution $x = \frac{n}{2} - \epsilon$ and further simplifications. Note that ϵ still is the residual between $\frac{n}{2}$ and x, $0 \leq \epsilon \leq \frac{n}{2}$. We arrive at the analogue of equation 4.18,

$$
(***) = \frac{1}{2n(2^J)^2}\left(\frac{n^3}{4} - \frac{n^2}{2}\epsilon\right). \qquad (6.52)
$$

Since $\epsilon \leq \frac{n}{2}$, $(***) \geq 0$, and the result follows.

□

Bibliography

[1] N. Shrivastva, S. Devi, and J. K. Verma. Digital money: The empowering new currency. In *2020 International Conference on Computational Performance Evaluation (ComPE)*, pages 837–840, 2020. doi: 10.1109/ComPE49325.2020.9200036.

[2] G. Cancellieri and M. Battaglioni. Data transmission in automotive applications and security/safety requirements. In *2020 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*, pages 1–6, 2020. doi: 10.23919/AEITAUTOMOTIVE50086.2020.9307393.

[3] Chen JunLi, Qing Dinghu, Yu Haifeng, Zhang Hao, and MeiJuan Nie. Email encryption system based on hybrid AES and ECC. In *IET International Communication Conference on Wireless Mobile and Computing (CCWMC 2011)*, pages 347–350, 2011. doi: 10.1049/cp.2011.0906.

[4] NIST CNSS Policy No. 15, Fact Sheet No. 1. National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information, June 2003.

[5] J. Bock. RSA-PSS – Provably secure RSA signatures and their implementation. *Humboldt-Universitat zu Berlin*, 2014.

[6] M. J. Dworkin et al. Announcing the Advanced Encryption Standard (AES). *NIST Publication Series Report*, (197), 2001.

[7] B. Qing-Hai, Z. Wen-Bo, J. Peng, and L. Xu. Research on design principles of elliptic curve public key cryptography and its implementation. In *2012 International*

Conference on Computer Science and Service System, pages 1224–1227, 2012. doi: 10.1109/CSSS.2012.310.

[8] T. Yang, Y. Zhang, S. Xiao, and Y. Zhao. Digital signature based on ISRSAC. *China Communications*, 18(1):161–168, 2021. doi: 10.23919/JCC.2021.01.014.

[9] H. K. Saini N. Chouhan and S. C. Jain. A novel technique to modify the Shor's algorithm — scaling the encryption scheme. *Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–4, 2019.

[10] T. Fritzmann, J. Vith, and J. Sepúlveda. Strengthening post-quantum security for automotive systems. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 570–576, 2020. doi: 10.1109/DSD51259.2020.00094.

[11] Y. Liu, X. Liu, J. Wang, L. Zhang, and C. Tang. Security analysis of electronic payment protocols based on quantum cryptography. In *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, pages 1709–1712, 2017. doi: 10.1109/ICISCE.2017.357.

[12] Flavio Mania, Carlos Henrique Da Silva Santos, and Alexandre Alvaro. Outlining low costs and open embedded systems for RFID in Internet of Things applications. In *2014 IEEE Brasil RFID*, pages 16–18, 2014. doi: 10.1109/BrasilRFID.2014.7128954.

[13] E. R. Naru, H. Saini, and M. Sharma. A recent review on lightweight cryptography in IoT. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 887–890, 2017. doi: 10.1109/I-SMAC.2017.8058307.

[14] S. B. Sadkhan and A. O. Salman. A survey on lightweight-cryptography status and future challenges. In *2018 International Conference on Advance of Sus-*

tainable Engineering and its Application (ICASEA), pages 105–108, 2018. doi: 10.1109/ICASEA.2018.8370965.

[15] Shiguo Lian. Multimedia Content Encryption. *CRC Press*, 2009.

[16] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel. A survey of lightweight-cryptography implementations. *IEEE Design Test of Computers*, 24(6): 522–533, 2007. doi: 10.1109/MDT.2007.178.

[17] Meltem Turan, and Cagdas Calik Kerry McKay, Donghoon Chang, and Larry Bassham. Status report on the first round of the NIST lightweight cryptography standardization process. *NIST publication*, 2019. doi: 10.6028/NIST.IR.8268.

[18] NIST. Lightweight cryptography workshop 2020. https://csrc.nist.gov/events/2020/lightweight-cryptography-workshop-2020. Accessed 2021-02-04.

[19] Updata to address cybersecurity vulnerabilities in Abbott's (formerly St. Jude Medical's) implantable cardiac pacemakers: FDA safety communication. https://www.fda.gov/medical-devices/safety-communications/firmware-update-address-cybersecurity-vulnerabilities-identified-abbotts-formerly-
2017. Accessed 2021-02-04.

[20] C. McCarthy, Kevin Harnett, and A. Carter. Characterization of potential security threats in modern automobiles: A composite modeling approach. 2014.

[21] S. T. Patel and N. H. Mistry. A survey: Lightweight cryptography in WSN. pages 11–15, 2015. doi: 10.1109/ICCN.2015.3.

[22] J. Schwemmlein, R. Posch, and K. C. Posch. High performance modular arithmetic using an RNS based chipset. In *Proceedings of the First International Conference on Mas-*

sively Parallel Computing Systems (MPCS) The Challenges of General-Purpose and Special-Purpose Computing, pages 444–451, 1994. doi: 10.1109/MPCS.1994.367046.

[23] H. D. L. Hollmann, R. Rietman, S. de Hoogh, L. Tolhuizen, and P. Gorissen. A multi-layer recursive residue number system. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 1460–1464, 2018. doi: 10.1109/ISIT.2018.8437612.

[24] Jason M. McGinthy and Alan J. Michaels. Semi-coherent transmission security for low power iot devices. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 170–177, 2018. doi: 10.1109/Cybermatics_2018.2018.00059.

[25] R. Fisher and F. Yates. *Statistical tables for biological, agricultural, and medical research*. Edinburgh: Oliver and Boyd, 1948.

[26] N. G. McDonald. Past, present, and future methods of cryptography and data encryption. *SpaceStation*, 2009.

[27] R. F. Churchhouse. *Codes and Ciphers: Julius Caesar, the Enigma, and the Internet*. 2001.

[28] Paul Gannon. The first intelligence war: First world war communications. *Engineering Technology*, 9(6):48–51, 2014. doi: 10.1049/et.2014.0605.

[29] A.E. Sale. Lorenz and Colossus: military cryptography. In *Proceedings 13th IEEE Computer Security Foundations Workshop. CSFW-13*, pages 216–222, 2000. doi: 10.1109/CSFW.2000.856938.

[30] Y. Dodis and J. Spencer. On the (non)universality of the one-time pad. In *The 43rd*

Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings., pages 376–385, 2002. doi: 10.1109/SFCS.2002.1181962.

[31] L. Robertson. Anecdotes. *IEEE Annals of the History of Computing*, 26(2):86–89, 2004. doi: 10.1109/MAHC.2004.1299663.

[32] Amandeep and G. Geetha. On the security of reduced key tiny encryption algorithm. In *2012 International Conference on Computing Sciences*, pages 322–325, 2012. doi: 10.1109/ICCS.2012.51.

[33] Panos Papadimitratos, V. Gligor, and J. Hubaux. Securing vehicular communications - assumptions, requirements, and principles. 2006.

[34] A. Beutelspacher. *Cryptology*. Mathematical Association of America, 1994.

[35] M. E. Smid and D. K. Branstad. Data Encryption Standard: past and future. *Proceedings of the IEEE*, 76(5):550–559, 1988. doi: 10.1109/5.4441.

[36] distributed.net. DES code cracked in record time. http://www.distributed.net/images/7/7e/19990120_-_PR-_nmfusion.pdf. Accessed 2021-02-04.

[37] Wade Trappe and Lawrence C. Washington. *Introduction to Cryptography with Coding Theory (2nd Edition)*. Prentice-Hall, Inc., USA, 2005. ISBN 0131862391.

[38] NIST. Request for Comments on Candidate Algorithms for the Advanced Encryption Standard (AES). https://csrc.nist.gov/news/1998/request-for-comments-on-round-1-aes-candidates, 1998. Accessed: 2021-4-25.

[39] D. Bernstein. Cryptographic Competitions: AES: the Advanced Encryption Standard. https://competitions.cr.yp.to/aes.html, 2017. Accessed: 2021-4-25.

[40] J. Nechvatal et al. Status report on the first round of the development of the Advanced Encryption Standard (AES). *Journal of Research of the National Institute of Standards and Technology*, 104(5):435–459, 1999.

[41] NIST. Report on the Development of the Advanced Encryption Standard (AES. *Journal of Research of the National Institute of Standards and Technology*, 106(3), 2001.

[42] B. Gladman. The Need for Multiple AES Winners. http://cm.1-s.es/USB-1/Disk%202/02-Cryptome-org-13-0702-directories/02/bg/winners.pdf, 1999. Accessed: 2021-4-25.

[43] Andrey Bogdanov. Improved side-channel collision attacks on AES. pages 84–95, 2007.

[44] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 344–371, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-25385-0.

[45] Lynn Hathaway. National policy on the use of the Advanced Encryption Standard (AES) to protect national security systems and national security information. Homeland Security Digital Library, 2003.

[46] A. Kulkarni and S. Sathe. Healthcare applications of the Internet of Things : A review. 2014.

[47] L. Wang et. al. D. Chen, Z. Liu. Natural Disaster Monitoring with Wireless Sensor Networks: A Case Study of Data-intensive Applications upon Low-Cost Scalable Systems. *Mobile Netw Appl*, 18, 103.

[48] Carsten Rolfes, Axel Poschmann, Gregor Leander, and Christof Paar. Ultra-lightweight implementations for smart devices – security for 1000 gate equivalents. In Gilles Gri-

maud and François-Xavier Standaert, editors, *Smart Card Research and Advanced Applications*, pages 89–103, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[49] S. T. Patel and N. H. Mistry. A survey: Lightweight cryptography in WSN. In *2015 International Conference on Communication Networks (ICCN)*, pages 11–15, 2015. doi: 10.1109/ICCN.2015.3.

[50] C. Easttom. A study of cryptographic backdoors in cryptographic primitives. In *Electrical Engineering (ICEE), Iranian Conference on*, pages 1664–1669, 2018. doi: 10.1109/ICEE.2018.8472465.

[51] E. B. Barker and J. M. Kelsey. Recommendation for random number generattion using deterministic random bit generators (revised). *US Department of Commerce, Technology Administration, National Institute of Standards and Technoogy, Computer Security Division, Information Technology Laboratory*, 2012.

[52] S. Landau. Highlights from making sense of Snowden, part II: What's significant in the NSA revelations. *IEEE Security Privacy*, 12(1):62–64, 2014. doi: 10.1109/MSP.2013.161.

[53] Bruce Schneier, Matthew Fredrikson, Tadayoshi Kohno, and Thomas Ristenpart. Surreptitiously weakening cryptographic systems. *IACR Cryptology ePrint Archive*, 2015: 97, 2015. URL https://eprint.iacr.org/2015/097.

[54] M. Amin and M. Afzal. On the vulnerability of EC DRBG. pages 318–322, 2015. doi: 10.1109/IBCAST.2015.7058523.

[55] Malicious hashing: Eve's variant of SHA-1. *Joux A., Youssef A. (eds) Selected Areas in Cryptography – SAC 2014. SAC 2014. Lecture Notes in Computer Science*, 2, 2014.

[56] A. Bannier and E. Filiol. Mathematical backdoors in symmetric encryption systems - proposal for a backdoored AES-like block cipher. 2017.

[57] K. McKay et. al. NISTIR 8114: Report on Lightweight Cryptography. https://nvlpubs.nist.gov/nistpubs/ir/2017/NIST.IR.8114.pdf, 2017. Accessed: 2021-4-25.

[58] Jaemin Park, Junchae Na, and Minjeong Kim. A practical approach for enhancing security of EPC global RFID Gen2 Tag. In *Future Generation Communication and Networking (FGCN 2007)*, volume 1, pages 436–441, 2007. doi: 10.1109/FGCN.2007.35.

[59] ISO, ISO/IEC 18000-63:2015,. Information Technology - Radio frequency identification for item management - Part 63: Parameters for air interface communications at 860 MHz to 960 MHz Type C. https://www.iso.org/standard/63675.html, 2015. Accessed: 2021-4-25.

[60] R. Anusha and V. Veena Devi Shastrimath. LCBC-XTEA: High throughput lightweight cryptographic block cipher model for low-cost RFID systems. In *Cybernetics and Automation Control Theory Methods in Intelligent Algorithms*, pages 185–196, Cham, 2019. Springer International Publishing. ISBN 978-3-030-19813-8.

[61] NIST. Status Report on the First Round of the NIST Lightweight Cryptography Standardization Process. https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8268.pdf, 2019. Accessed: 2021-4-25.

[62] S. Saha, R. K. Karsh, and M. Amrohi. Encryption and decryption of images using secure linear feedback shift registers. pages 295–298, 2018.

[63] Yoo Shin, Sangho & Kee-Young. An improved PRNG based on the hybrid between one- and two- dimensional cellular automata. 2011. 10.5772/15894.

[64] P. Jan B. Paul, G. Trivedi and Z. Němec. Efficient PRNG design and implementation for various high throughput cryptographic and low power security applications. *29th International Conference Radioelektronika*, pages 1–6, 2019.

[65] Dutta S.R. Bhowal, S. and S Mitra. An efficient reduced set brute force attack technique for a particular steganographic tool using Vername algorithm. *2017 Fourth International Conference on Image Information Processing (ICIIP)*, pages 1–4, 2017.

[66] A. Ferrenberg, et al. Monte Carlo simulations: Hidden errors from ''good'' random number generators. *Physical Review Letters*, 69(23):3382–3384, 1992.

[67] Martin Lüscher. A portable high-quality random number generator for lattice field theory simulations. *Computer Physics Communications*, 71(1):100–110, 1994.

[68] A. Rukhin, et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications. 2000.

[69] G. Marsaglia. DIEHARD: A battery of tests of randomness. 1995.

[70] Fengling Zhang, Yanting Ai, and Fei Liu. Typical fault mode determination for rotor test rig based on correlation dimension and kolmogorov entropy. In *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, volume 6, pages 484–488, 2009. doi: 10.1109/FSKD.2009.296.

[71] S. Ho and R. Yeung. On the relation between the Shannon entropy and the von Neumann entropy. In *International Symposium on Information Theory, 2004. ISIT 2004. Proceedings.*, pages 42–, 2004. doi: 10.1109/ISIT.2004.1365079.

[72] S. Goldwasser, S. Micali, and P. Tong. Why and how to establish a private code on a public network. In *23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)*, pages 134–144, 1982. doi: 10.1109/SFCS.1982.100.

[73] S. Kamara. Lectures 2+3: Provable security. *Brown University Lectures*, 2016.

[74] J.Gray, K. Ip, and K. Lui. Provable security for cryptographic protocols-exact analysis and engineering applications. In *Proceedings 10th Computer Security Foundations Workshop*, pages 45–58, 1997. doi: 10.1109/CSFW.1997.596784.

[75] J. Wu and D. Stinson. On the security of the ElGamal Encryption Scheme and Damgard's Variant. *International Association for Cryptologic Research*, 2009.

[76] G. Marks. A casino gets hacked through fish-tank thermometer. *Enterpreneur*, 2021.

[77] Claude E. Shannon. A symbolic analysis of relay and switching circuits. *Electrical Engineering*, 57(12):713–723, 1938. doi: 10.1109/EE.1938.6431064.

[78] Shugang Wei and Kensuke Shimizu. Error detection of arithmetic circuits using a residue checker with signed-digit number system. In *Proceedings 2001 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 72–77, 2001. doi: 10.1109/DFTVS.2001.966754.

[79] D. Bundalo, Z. Bundalo, F. Softić, M. Kostadinović, and D. Pašalić. Logic circuits with high-impedance output state for interconnection of ternary and binary CMOS digital circuits and systems. In *2012 Proceedings of the 35th International Convention MIPRO*, pages 97–102, 2012.

[80] J.F. Berry. A framework for understanding large scale digital storage systems. In *Proceedings of IEEE 14th Symposium on Mass Storage Systems*, pages 293–304, 1995. doi: 10.1109/MASS.1995.528239.

[81] Huan Xie and Xiaohua Tong. An inproved binary encoding algorithm for classification of hyperspectral images. In *2012 4th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, pages 1–4, 2012. doi: 10.1109/WHISPERS.2012.6874331.

[82] Roberto Leyva, Victor Sanchez, and Chang-Tsun Li. Abnormal event detection in videos using binary features. In *2017 40th International Conference on Telecommunications and Signal Processing (TSP)*, pages 621–625, 2017. doi: 10.1109/TSP.2017.8076061.

[83] Zhengsheng Li and Wenyan Ma. 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)*, volume 1, pages 304–307 vol.1, 2000. doi: 10.1109/WCICA.2000.859971.

[84] Junnan Gao, Fang Yang, and Xu Ma. Indoor positioning system based on visible light communication with Gray-coded identification. In *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 899–903, 2017. doi: 10.1109/IWCMC.2017.7986405.

[85] Arjun Tyagi, Ashu Verma, and Abhinav Saxena. Optimal economic dispatch considering wind energy conversion systems using Gray coded genetic algorithm. In *2015 Annual IEEE India Conference (INDICON)*, pages 1–5, 2015. doi: 10.1109/INDICON.2015.7443234.

[86] Tariq Jamil. An introduction to complex binary number system. In *2011 Fourth International Conference on Information and Computing*, pages 229–232, 2011. doi: 10.1109/ICIC.2011.37.

[87] Tariq Jamil. Complex binary associative dataflow processor - a tutorial. In *Southeast-Con 2018*, pages 1–3, 2018. doi: 10.1109/SECON.2018.8478931.

[88] A.R. Plantz and M. Berman. Adoption of the octal number system. *IEEE Transactions on Computers*, C-20(5):593–598, 1971. doi: 10.1109/T-C.1971.223307.

[89] Ronald Vincent and S. L. Anju. Decimal floating point format based on commonly used precision for embedded system applications. In *2013 Annual International Conference on Emerging Research Areas and 2013 International Conference on Microelectronics, Communications and Renewable Energy*, pages 1–4, 2013. doi: 10.1109/AICERA-ICMiCR.2013.6575957.

[90] S. Levachkine, A. Velazquez, and V. Alexandrov. Color image segmentation using false colors and its applications to geo-images treatment: alphanumeric character recognition. In *IGARSS 2001. Scanning the Present and Resolving the Future. Proceedings. IEEE 2001 International Geoscience and Remote Sensing Symposium (Cat. No.01CH37217)*, volume 5, pages 2212–2214 vol.5, 2001. doi: 10.1109/IGARSS.2001.977952.

[91] Non-binary pseudorandom number generators for information security purposes. *Procedia Computer Science*, 123:203–211, 2018. 8th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2017 (Eighth Annual Meeting of the BICA Society), held August 1-6, 2017 in Moscow, Russia.

[92] The art of computer programming: Positional number systems. *informIT*, 2014. Online, Accessed April 23, 2021 at https://www.informit.com/articles/article.aspx?p=2221791.

[93] D. Knuth. *Bitwise tricks and techniques*. Addison-Wesley, . ISBN 0-321-58050-8.

[94] H. Fang J. Glenn and C. Kruskal. A retrograde approximation algorithm for multi-player can't stop. *Computers and Games: 6th International Conference, CG 2008*, 2008.

[95] D. Knuth. Volume 2: Seminumerical algorithms, the art of computer programming. *Addison-Wesley*, page 192, . ISBN 0-201-89684-2.

[96] W. K. Jenkins, M. A. Soderstrand, and C. Radhakrishnan. Historical patterns of emerging residue number system technologies during the evolution of computer engineering and digital signal processing. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2018. doi: 10.1109/ISCAS.2018.8351066.

[97] SHEN KANGSHENG. Historical development of the Chinese Remainder Theorem. *Archive for History of Exact Sciences*, 38(4), 1988.

[98] P. V. Ananda Mohan. *Residue Number Systems: Algorithms and Architecture*. The International Series in Engineering and Computer Science, 2002.

[99] K. E. Nelson and M. A. Soderstrand. Full tunable digital heterodyne IIR filters. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, volume 3, pages 1141–1144 vol.3, 1999. doi: 10.1109/ICASSP.1999.756178.

[100] L. Imbert and G. A. Jullien. Efficient fault-tolerant arithmetic using a symmetrical modulus replication RNS. In *2001 IEEE Workshop on Signal Processing Systems. SiPS 2001. Design and Implementation (Cat. No.01TH8578)*, pages 93–100, 2001. doi: 10.1109/SIPS.2001.957334.

[101] M. A. Soderstrand. Contributions of Graham Jullien and William Miller to modified quadratic number system arithmetic. In *2018 IEEE 61st International Mid-*

west Symposium on Circuits and Systems (MWSCAS), pages 161–164, 2018. doi: 10.1109/MWSCAS.2018.8623961.

[102] Javad Doliskani, Pascal Giorgi, Romain Lebreton, and Eric Schost. Simultaneous conversions with the residue number system using linear algebra. *ACM Trans. Math. Softw.*, 44(3), January 2018. ISSN 0098-3500. doi: 10.1145/3145573. URL https://doi.org/10.1145/3145573.

[103] M. A. Bayoumi, G. A. Jullien, and W. C. Miller. A VLSI implementation of finite impulse response digital filters using residue number systems. *Canadian Electrical Engineering Journal*, 10(2):76–80, 1985. doi: 10.1109/CEEJ.1985.6592022.

[104] Shinichi Kawamura, Masanobu Koike, Fumihiko Sano, and Atsushi Shimbo. Cox-Rower architecture for fast parallel Montgomery multiplication. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, pages 523–538, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[105] G. Dimitrakopoulos and V. Paliouras. Graph-based optimization for a CSD-enhanced RNS multiplier. In *The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002.*, volume 3, 2002. doi: 10.1109/MWSCAS.2002.1187123.

[106] Tso-Bing Juang and Jian-Hao Huang. Multifunction rns modulo (2n±1) multipliers based on modified booth encoding. In *2012 IEEE Asia Pacific Conference on Circuits and Systems*, pages 515–518, 2012. doi: 10.1109/APCCAS.2012.6419085.

[107] D. M. Schinianakis, A. P. Fournaris, H. E. Michail, A. P. Kakarountas, and T. Stouraitis. An RNS implementation of an f_p elliptic curve point multiplier. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 56(6):1202–1213, 2009. doi: 10.1109/TCSI.2008.2008507.

[108] J. Bajard and L. Imbert. A full RNS implementation of RSA. *IEEE Transactions on Computers*, 53(6):769–774, 2004. doi: 10.1109/TC.2004.2.

[109] F. Gandino, F. Lamberti, P. Montuschi, and J. Bajard. A general approach for improving RNS Montgomery exponentiation using pre-processing. In *2011 IEEE 20th Symposium on Computer Arithmetic*, pages 195–204, 2011. doi: 10.1109/ARITH.2011.35.

[110] F. Gandino, F. Lamberti, G. Paravati, J. Bajard, and P. Montuschi. An algorithmic and architectural study on Montgomery Exponentiation in RNS. *IEEE Transactions on Computers*, 61(8):1071–1083, 2012. doi: 10.1109/TC.2012.84.

[111] D. Schinianakis and T. Stouraitis. Multifunction residue architectures for cryptography. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(4):1156–1169, 2014. doi: 10.1109/TCSI.2013.2283674.

[112] Edward Yellakuor Baagyere, Peter Awon-Natemi Agbedemnab, Zhen Qin, Mohammed Ibrahim Daabo, and Zhiguang Qin. A multi-layered data encryption and decryption scheme based on genetic algorithm and residual numbers. *IEEE Access*, 8: 100438–100447, 2020. doi: 10.1109/ACCESS.2020.2997838.

[113] M. Delgado-Restituto and A. Rodriguez-Vazquez. Mixed-signal map-configurable integrated chaos generator for chaotic communications. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 48(12):1462–1474, 2001. doi: 10.1109/TCSI.2001.972853.

[114] J. M. McGinthy and A. J. Michaels. Semi-coherent transmission security for low power IoT devices. pages 170–177, 2018.

[115] A. S. Molahosseini, S. Sorouri, and A. A. E. Zarandi. Research challenges in next-generation residue number system architectures. In *2012 7th International Con-*

ference on Computer Science Education (ICCSE), pages 1658–1661, 2012. doi: 10.1109/ICCSE.2012.6295382.

[116] D. Younes and P. Steffan. Efficient image processing application using residue number system. In *Proceedings of the 20th International Conference Mixed Design of Integrated Circuits and Systems - MIXDES 2013*, pages 468–472, 2013.

[117] A. Michaels and P. Leary. Chaos without clocks. *2014 IEEE Military Communicaitons Conference (MILCOM)*, 2014.

[118] Atta-ur-Rahman, M. T. Naseem, I. M. Qureshi, and M. Z. Muzaffar. Reversible watermarking using residue number system. In *2011 7th International Conference on Information Assurance and Security (IAS)*, pages 162–166, 2011. doi: 10.1109/ISIAS.2011.6122813.

[119] I. Lee and W. K. Jenkins. The design of residue number system arithmetic units for a VLSI adaptive equalizer. In *Proceedings of the 8th Great Lakes Symposium on VLSI (Cat. No.98TB100222)*, pages 179–184, 1998. doi: 10.1109/GLSV.1998.665222.

[120] R. Chokshi. Exploiting residue number system for power-efficient digital signal processing in embedded processors. In *Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems*, 2009.

[121] A. J. Michaels. A maximal entropy digital chaotic circuit. *IEEE International Symposium of Circuits and Systems (ISCAS)*, pages 717–720, 2011.

[122] M. Jessa. Data transmission with adjustable security exploiting chaos-based pseudo-random number generators. 3:III–III, 2002.

[123] Juan Manuel Rodriguez Bejarano, Ana Yun, and Borja De La Cuesta. Security in IP satellite networks: COMSEC and TRANSEC integration aspects. In *2012 6th*

Advanced Satellite Multimedia Systems Conference (ASMS) and 12th Signal Processing for Space Communications Workshop (SPSC), pages 281–288, 2012. doi: 10.1109/ASMS-SPSC.2012.6333089.

[124] R. Luzzi A. Trifiletti M. Bucci, L. Germani and M. Varanonuovo. A high-speed oscillator-based truly random number source for cryptographic applications on a smart card IC. *in IEEE Transactions on Computers*, 52(4):403–409, 2003.

[125] M. A. Şarkışla and S. Ergün. An area efficient true random number generator based on modified ring oscillators. pages 274–278, 2018.

[126] M. Stipčević. Quantum random number generators and their use in cryptography. pages 1474–1479, 2011.

[127] L. Gong, J. Zhang, H. Liu, L. Sang, and Y. Wang. True random number generators using electrical noise. *IEEE Access*, 7:125796–125805, 2019.

[128] W. T. Holman, J. A. Connelly, and A. B. Dowlatabadi. An integrated analog/digital random noise source. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 44(6):521–528, 1997.

[129] Xuecheng Zou Zhenglin Liu Dongfang Li, Zhaojun Lu. PUFKEY: A high-security and high-throughput hardware true random number generator for sensor networks. *Sensors*, 2015.

[130] S. V. Sathyanarayana and R. Lavanya. Group Diffie Hellman key exchange algorithm based secure group communication. *3rd International Conference on Applied and Theoretical Computing and Communication Technology*, pages 281–289, 2017.

[131] N. P. Cagigal and S. Bracho. Algorithmic determination of linear-feedback in a shift

register for pseudorandom binary sequence generation. *in IEE Proceedings G - Electronic Circuits and Systems*, 133(4):191–194, 1986.

[132] S. Dey. SD-C1BBR: SD-count-1-byte-bit randomization: A new advanced cryptographic randomization technique. *World Congress on Information and Communication Technologies*, pages 236,241, 2012.

[133] Noor Jumaa. Digital image encryption using AES and random number generator. *Iraqi Journal for Electrical And Electronic Engineering*, 2018.

[134] R. Hobincu and O. Datcu. A novel chaos based PRNG targeting secret communication. pages 459–462, 2018.

[135] Philip Kwan. Interface specification IS-GPS-200, 2019.

[136] Rafael & Tortosa Leandro & Zamora Antonio. Aguirre, Jose-Vicente & Alvarez. Fast pseudorandom generator based on packed matrices. 2007.

[137] A. J. Michaels, M. Meadows, and J. Ernst. PRNG sequence combination techniques via Galois extension fields. pages 841–845, 2017.

[138] G. Manjula and H. S. Mohan. Improved dynamic S-box generation using hash function for AES and its performance analysis. pages 109–115, 2018.

[139] K. H. Leung K. H. Tsoi and P. H. W. Leong. Compact FPGA-based true and pseudo random number generators. *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 51–61, 2003. Napa, CA, USA.

[140] R. O. Dror J. K. Salmon, M. A. Moraes and D. E. Shaw. Parallel random numbers: As easy as 1, 2, 3. *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2011.

[141] H. K. Verma A. Kaur and R. K. Singh. 3D — Playfair cipher using LFSR based unique random number generator. *Sixth International Conference on Contemporary Computing (IC3)*, pages 18–23, 2013.

[142] G. Setti V. R. Gonzalez-Diaz, F. Pareschi and F. Maloberti. A pseudorandom number generator based on time-variant recursion of accumulators. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 58(9):580–584, 2011.

[143] D. A. Visan, I. Lita, M. Jurian, and M. Gherghe. Pseudorandom sequence generator for spread spectrum communications. pages 1–4, 2018.

[144] J. M. Gutiérrez-Cárdenas. Secret key steganography with message obfuscation by pseudo-random number generators. pages 164–168, 2014.

[145] C. Valli M. Chernyshev and M. Johnstone. Revisiting urban war nibbling: Mobile passive discovery of classic Bluetooth devices using Ubertooth one. *in IEEE Transactions on Information Forensics and Security*, 12(7):1625–1636, 2017.

[146] S. Buchovecká and J. Hlaváč. Frequency injection attack on a random number generator. *IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, pages 128–130, 2013.

[147] T. Nighswander, B. Ledvina, J. Diamond, R. Brumley, and D. Brumley. GPS software attacks. 2012.

[148] D. Adrian, et al. Imperfect forward secrecy: How Diffie-Hellman fails in practice. *In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15).*, pages 5–17, 2015.

[149] K. Imamura and W. Yoshida. A simple derivation of the Berlekamp-Massey algorithm

and some applications. *in IEEE Transactions on Information Theory*, 33(1):146–150, 1987.

[150] G. Feng and K. K. Tzeng. A generalization of the Berlekamp-Massey algorithm for multisequence shift-register synthesis with applications to decoding cyclic codes. *IEEE Transactions on Information Theory*, 37(5):1274–1287, 1991.

[151] M. Jessa and M. Walentynowicz. Statistical properties of number sequences generated by 1d chaotic maps considered as a potential source of pseudorandom number sequences. 1:449–455 vol.1, 2001.

[152] R. Purwoko K. Inayah, B. E. Sukmono and S. Indarjani. Insertion attack effects on standard PRNGs ANSI X9.17 and ANSI X9.31 based on statistical distance tests and entropy difference tests. *International Conference on Computer, Control, Informatics and Its Applications (IC3INA)*, pages 219–224, 2013.

[153] Yahya S. Khiabani, Shuangqing Wei, Jian Yuan, and Jian Wang. Enhancement of secrecy of block ciphered systems by deliberate noise. *IEEE Transactions on Information Forensics and Security*, 7(5):1604–1613, 2012. doi: 10.1109/tifs.2012.2204983.

[154] M. Willett. Deliberate noise in a modern cryptographic system (corresp.). *IEEE Transactions on Information Theory*, 26(1):102–104, 1980. doi: 10.1109/tit.1980.1056136.

[155] João P. Vilela, Matthieu Bloch, João Barros, and Steven W. Mclaughlin. Wireless secrecy regions with friendly jamming. *IEEE Transactions on Information Forensics and Security*, 6(2):256–266, 2011. doi: 10.1109/tifs.2011.2111370.

[156] S. Abhishek Anand and Nitesh Saxena. Vibreaker: Securing vibrational pairing with deliberate acoustic noise. *Proceedings of the 9th ACM Conference on Security Privacy in Wireless and Mobile Networks - WiSec 16*, 2016. doi: 10.1145/2939918.2939934.

[157] A. Michaels. Improved RNS-based PRNGs. *In Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES 2018)*, (220):1–5, 2018. Association for Computing Machinery, New York, NY, USA.

[158] Amos Omundi and Benjamin Premkumar. *Residue Number System: Theory and Implementation*. 01 2007.

[159] A. S. Molahosseini, S. Sorouri, and A. A. E. Zarandi. Research challenges in next-generation residue number system architectures. pages 1658–1661, 2012. doi: 10.1109/ICCSE.2012.6295382.

[160] Hanshen Xiao, Hari Garg, Jianhao Hu, and Guoqiang Xiao. New error control algorithms for residue number system codes. *ETRI Journal*, 38, 11 2015. doi: 10.4218/etrij.16.0115.0575.

[161] H. Xiao, Y. Ye, G. Xiao, and Q. Kang. Algorithms for comparison in residue number systems. pages 1–6, 2016. doi: 10.1109/APSIPA.2016.7820790.

[162] S. Bi and W. J. Gross. The mixed-radix Chinese Remainder Theorem and its applications to residue comparison. *IEEE Transactions on Computers*, 57(12):1624–1632, 2008. doi: 10.1109/TC.2008.126.

[163] A. Rafiev, J. P. Murphy, and A. Yakovlev. Quaternary Reed-Muller expansions of mixed radix arguments in cryptographic circuits. pages 370–376, 2009. doi: 10.1109/ISMVL.2009.21.

[164] A. Rafiev, A. Mokhov, F. Burns, J. Murphy, A. Koelmans, and A. Yakovlev. Mixed radix Reed-Muller expansions. *IEEE Transactions on Computers*, 61(8):1189–1202, 2012. doi: 10.1109/TC.2011.124.

[165] A. Michaels and D. Chester. Mixed radix number generator with chosen statistical artifacts, 2007. URL https://patents.google.com/patent/CA2633923C/en.

[166] C. Huang. A fully parallel mixed-radix conversion algorithm for residue number applications. *IEEE Transactions on Computers*, C-32(4):398–402, 1983. doi: 10.1109/TC.1983.1676242.

[167] L. Wang, X. Zhou, G. E. Sobelman, and R. Liu. Generic mixed-radix FFT pruning. *IEEE Signal Processing Letters*, 19(3):167–170, 2012. doi: 10.1109/LSP.2012.2184283.

[168] R. Kaur and T. Singh. Design of 32-point mixed radix FFT processor using csd multiplier. pages 538–543, 2016. doi: 10.1109/PDGC.2016.7913183.

[169] X. Yin, A. Zhang, H. Zhang, and L. Hao. Research and design of digital power quality analysis system based on mixed radix FFT. pages 7393–7399, 2014. doi: 10.1109/ChiCC.2014.6896228.

[170] Liu Zhizhe and Zhong Shun'an. A reconfigurable and high precision VLSI architecture for Fast Fourier Transform. 4:V4–420–V4–423, 2010. doi: 10.1109/ICCET.2010.5485498.

[171] S. Lin and W. Chung. The split-radix Fast Fourier transforms with radix-4 butterfly units. pages 1–5, 2013. doi: 10.1109/APSIPA.2013.6694148.

[172] Young-jin Moon and Young-il Kim. A mixed-radix 4-2 butterfly with simple bit reversing for ordering the output sequences. 3:4 pp.–1774, 2006. doi: 10.1109/ICACT.2006.206332.

[173] R. Shirbhate, T. Panse, and C. Ralekar. Design of parallel FFT architecture using Cooley Tukey algorithm. pages 0574–0578, 2015. doi: 10.1109/ICCSP.2015.7322551.

[174] Luca Pasqualini and Maurizio Parton. Pseudo random number generation: a reinforcement learning approach. *Procedia Computer Science*, 170:1122–1127, 2020. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2020.03.057. URL https://www.sciencedirect.com/science/article/pii/S1877050920304944. The 11th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 3rd International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops.

[175] Texas Instruments. What's a LFSR? https://www.ti.com/lit/an/scta036a/scta036a.pdf.

[176] X. Tian and K. Benkrid. Mersenne twister random number generation on FPGA, CPU and GPU. pages 460–464, 2009. doi: 10.1109/AHS.2009.11.

[177] K. Wang, Y. Cao, C. Chang, and X. Ji. High-speed true random number generator based on differential current starved ring oscillators with improved thermal stability. pages 1–5, 2019. doi: 10.1109/ISCAS.2019.8702785.

[178] A. J. Michaels and C. C. Lau. Quantization effects in digital chaotic communication systems. pages 1564–1569, 2013. doi: 10.1109/MILCOM.2013.264.

[179] A. Bergvall and D. Khailtash. The card dealer : A card shuffling and dealing robot. *Digitala Vetenskapliga Arkivet*, 2020. URL https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1462013&dswid=-8479.

[180] P. Diaconis, J. Fulman, and S. Holmes. Analysis of casino shelf shuffling machines. *The Annals of Applied Probability*, 2013.

[181] Carson Reynolds and Masatoshi Ishikawa. Robot trickery. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.147.6292.

[182] M. Silverman. Progressive randomization of a deck of playing cards: Experimental tests and statistical analysis of the riffle shuffle. *Open Journal of Statistics*, 9, 2019.

[183] David Aldous and Persi Diaconis. Shuffling cards and stopping times. *The American Mathematical Monthly*, 93(5):333–348, 1986. doi: 10.1080/00029890.1986.11971821. URL https://doi.org/10.1080/00029890.1986.11971821.

[184] D. Stark, A. Ganesh, and N. O'Connell. Information loss in riffle shuffling. *Combinatorics, Probability and Computing*, 11(1):79–95, 2002. doi: 10.1017/S0963548301004990.

[185] iTech Labs. Random number generator. https://www.kamagames.com/files/docs/RNG_Certificate_KG_UK_21Apr20.pdf.

[186] S.P. Lalley. Rifle shuffles and their associated dynamical systems. *Journal of Theoretical Probability*, pages 903–932, 1999. doi: https://doi.org/10.1023/A:1021636902356.

[187] Chung chih Li and Bo Sun. Using linear congruential generators for cryptographic purposes. *20th International Conference on Computers and Their Applications*, 2005.

[188] MathWorks Documentation. Randstream.list. 2021. https://www.mathworks.com/help/matlab/ref/randstream.randstream.list.html.

[189] Ankur, Divyanjali, and T. Bhardwaj. A dissection of pseudorandom number generators. pages 318–323, 2015. doi: 10.1109/SPIN.2015.7095369.

[190] Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8, 1998. URL "http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.315.6296&rep=rep1&type=pdf".

[191] W. E. Thomson. A Modified Congruence Method of Generating Pseudo-random Numbers. *The Computer Journal*, 1(2):83–83, 01 1958. ISSN 0010-4620. doi: 10.1093/comjnl/1.2.83.

[192] Guy Steele and S. Vigna. Computationally easy, spectrally good multipliers for congruential pseudorandom number generators. *ArXiv*, abs/2001.05304, 2020.

[193] George Marsaglia and Arif Zaman. A New Class of Random Number Generators. *The Annals of Applied Probability*, 1(3):462 – 480, 1991. doi: 10.1214/aoap/1177005878. URL https://doi.org/10.1214/aoap/1177005878.

[194] Martin Lüscher. A portable high-quality random number generator for lattice field theory simulations. *Computer Physics Communications*, 79(1):100–110, 1994. ISSN 0010-4655. doi: https://doi.org/10.1016/0010-4655(94)90232-1.

[195] Daniel M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27(1):129–146, 1998. ISSN 0196-6774.

[196] Shay Gueron. Efficient software implementations of modular exponentiation. *IACR Cryptology ePrint Archive*, 2011:239, 01 2011. doi: 10.1007/s13389-012-0031-5.

[197] Rob Slazas. Poker hand ranker. *MathWorks File Exchange*. https://www.mathworks.com/matlabcentral/fileexchange/17579-poker-hand-ranker.

[198] Xiaojuan Chen, Yaru Han, and Jie Wu. Burst noise measuring on the basis of wavelet and fourier transform. In *2010 International Conference on Measuring Technology and Mechatronics Automation*, volume 1, pages 769–771, 2010. doi: 10.1109/ICMTMA.2010.366.

[199] Yongho Oh, Seungyong Lee, Chan Hyeong Park, and Jae-Sung Rieh. Impact of substrate digital noise coupling on the high-frequency noise performance of RF MOS-

FETs. *IEEE Microwave and Wireless Components Letters*, 19(9):557–559, 2009. doi: 10.1109/LMWC.2009.2027064.

[200] Ashirbad Khuntia and Trailokya Nath Sasamal. A novel transient fault injection technique using Berlekamp-Massey Algorithm. In *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, pages 541–545, 2014. doi: 10.1109/ICACCCT.2014.7019144.