Analysis and Implementation of RC4 Stream Cipher

A thesis presented to Indian Statistical Institute
in fulfillment of the thesis requirement for the degree of
Doctor of Philosophy in Computer Science

by

Sourav Sen Gupta

under the supervision of

Professor Subhamoy Maitra

Applied Statistics Unit
INDIAN STATISTICAL INSTITUTE
Kolkata, West Bengal, India, 2013

To the virtually endless periods of
sweet procrastination
that kept me sane during the strenuous
one-night stands with my thesis.

Abstract

RC4 has been the most popular stream cipher in the history of symmetric key cryptography. Designed in 1987 by Ron Rivest, RC4 is the most widely deployed commercial stream cipher, having applications in network protocols such as SSL, WEP, WPA and in Microsoft Windows, Apple OCE, Secure SQL, etc. The enigmatic appeal of the cipher has roots in its simple design, which is undoubtedly the simplest for any practical cryptographic algorithm to date. In this thesis, we focus on the analysis and implementation of RC4.

For the first time in RC4 literature, we report significant keystream biases depending on the length of RC4 secret key. In the process, we prove two empirical biases that were experimentally reported and used in recent attacks against WEP and WPA by Sepehrdad, Vaudenay and Vuagnoux in EUROCRYPT 2011. In addition to this, we present a conclusive proof for the extended keylength dependent biases in RC4, a follow-up problem to our keylength dependent results, identified and partially solved by Isobe, Ohigashi, Watanabe and Morii in FSE 2013.

In a recent result by AlFardan, Bernstein, Paterson, Poettering and Schuldt, to appear in USENIX Security Symposium 2013, the authors observed a bias of the first output byte towards 129. While attempting the proof of this bias, we observed that the bias occurs prominently only for certain lengths of the RC4 secret key. In addition, our findings revealed that this bias may be related to the old and unsolved problem of 'anomalies' in the distribution of the state array S_0, right after the Key Scheduling Algorithm. In this connection, we prove the bias in $(S_0[128] = 127)$, a problem left open for more

than a decade since the observation of anomaly pairs by Mantin in 2001.

Subsequently, we present theoretical proofs of some significant initial-round empirical biases involving the state variables of RC4, observed by Sepehrdad, Vaudenay and Vuagnoux in SAC 2010. We also study the non-random behavior of index j and completely characterize the first two instances j_1, j_2 of the index. In the same part of the thesis, we show that there are certain events in the RC4 keystream that leak state information with probability $3/N$, higher than the best existing glimpse correlations identified by Jenkins in 1996.

We further study the short-term and long-term biases in RC4 keystream bytes. In the process, we derive the complete probability distribution of the first byte of RC4 keystream, a problem left open for a decade since the observation by Mironov in CRYPTO 2002. In addition to this, we contradict a claim by Mantin and Shamir from FSE 2001 by proving that there exist positive biases towards zero for all the initial bytes 3 to 255 of RC4.

In the recent work of AlFardan, Bernstein, Paterson, Poettering and Schuldt, extensive computations were conducted to identify significant short-term single-byte keystream biases of RC4, some of which already exist in the current literature, and some of which are new. We prove (almost) all new, unproved or partially proved significant biases observed in the aforesaid work. We also investigate for long-term non-randomness in the keystream, and prove a new long-term bias of RC4. Our proofs and observations in this thesis, along with their connections to the contemporary and old results, provide a comprehensive view on the state-of-the-art literature in RC4 cryptanalysis.

In the second part of this thesis, we systematically study the hardware implementation of RC4, and propose the fastest known design for the cipher. For the first time in the literature, we exploit the idea of loop unrolling in RC4 hardware design, and subsequently combine hardware pipelining to loop unrolling in a hybrid architecture that produces 2 bytes-per-cycle throughput in RC4. This is better than the existing 3 cycles-per-byte design by Kitsos, Kostopoulos, Sklavos and Koufopavlou, and the 3 cycles-per-byte and 1 cycle-per-byte patents of Matthews Jr. We have optimized and implemented our proposed designs using VHDL description, synthesized with 130 nm, 90 nm and 65 nm fabrication technologies, for proper comparison with existing designs in terms of keystream throughput and hardware footprint.

Acknowledgments

You can't connect the dots looking forward; you can only connect them looking backwards. – Steve Jobs

Looking back at the last few years that I have spent at Indian Statistical Institute as a doctoral student, I see several people who have significantly influenced the course of my thesis, as well as my life. I take this opportunity to offer my gratitude to all who stood beside me in the hardest of times.

First and foremost, I extend my heartfelt gratitude to my PhD supervisor, Prof. Subhamoy Maitra, Indian Statistical Institute, Kolkata. The ease with which Subhamoy-da converted our 'student-supervisor' relationship into a lasting friendship, will remain an eternal mystery to me. Every research problem that we have discussed, debated upon and solved, has enriched my academic outlook; and every morsel of experience that he has kindly shared with me will continue to add value to my life. I will always be grateful to Subhamoy-da for being my friend, philosopher and guide during the last few years; primarily because he kept his sanity and sense of humor alive whenever I lost mine!

It was a delight working with my collaborators and co-authors – Prof. Bhabani P Sinha of ISI Kolkata, Prof. Dr.-Ing. Anupam Chattopadhyay of RWTH Aachen, Dr. Goutam Paul of Jadavpur University, Dr. Santanu Sarkar of CMI Chennai and Dr. Koushik Sinha of HP Labs India. Their academic insight and vision have taught me a lot over the last few years, and I look forward to further collaborations with them in the future. I would like to specially thank Goutam-da and Anupam-da for mentoring and supporting me in numerous ways, academic and otherwise, during the course of this thesis.

I would like to express my sincerest gratitude to the esteemed anonymous reviewers of my thesis for their invaluable comments, which helped me improve the technical presentation and editorial quality of the thesis to a great extent.

I am thankful to Applied Statistics Unit, ISI Kolkata for supporting me during the formative years of my doctoral studies; Centre of Excellence in Cryptology, ISI Kolkata, funded by DRDO, Government of India, for supporting parts of my research; and MPSoC Architectures, UMIC Research Centre, RWTH Aachen University, Germany, for hosting me during Summer 2011.

I have always idolized Prof. Bimal Roy, Director, Indian Statistical Institute, without whose active support and unparalleled vision, the field of Cryptology would probably not exist in Indian academia in the first place. I extend my sincerest gratitude to Bimal-da; it is because of him that I could get the perfect academic ambiance at the Institute, and the coveted exposure in this niche area that would otherwise remain only a distant dream for me.

I am indebted to Prof. Rana Barua, Prof. Palash Sarkar, Dr. Kishan Chand Gupta and Dr. Mridul Nandi for several invaluable discussions during the tenure of my PhD. I would also like to thank my past and present colleagues at ISI Kolkata, Sibu-da, Mrinal-da, Somitra-da, Sumanta-da, Pinaki-da, Ashish-da, Sushmita-di, Srimanta-da, Rishi-da, Sumit-da, Tapas-da, Indranil-da, Subhadeep-da, Sanjay, Subhabrata, Somindu, Shashank, Samiran, Nilanjan, Avik, Satrajit and Pratyay, for providing me with an adorable social support. Special thanks to Rishi-da, as it was only through the uncountably many philosophical and technical discussions with him that I could see myself through the most arduous times during the last few years.

I humbly acknowledge the kind support of Cryptology Research Society of India, and that of Mr. Amitabha Sinha (our beloved Amitabha-da), during my stay at ISI Kolkata. I am also thankful to the Director's office, the Dean's office, the ASU, BIRU and CoEC offices, and all administrative sectors of the Institute, for their continuous support to make my life smooth at the Institute.

My family has always been the 'happy place' for me; even in the hardest of times. This thesis could not be completed without the ardent support and inspiration from my parents, *Baapi* and *Maa*, who made me what I am by resting their unflinching faith in me at every step of my life. Undoubtedly the most significant contribution towards this thesis has been made by *Sananda* – my wife, who painstakingly accepted the thesis as her co-wife, night after night, even during the first year of our marriage. Thank you *Love*!

Contents

List of Tables

List of Figures

Chapter 1

Preliminaries and Motivation

"I have grown to love secrecy. It seems to be the one thing that can make modern life mysterious or marvelous to us. The commonest thing is delightful if only one hides it." – Oscar Wilde

Since the dawn of humanity, one of the strongest intrinsic characteristics of the human mind has been its affinity towards secrecy. Be it in terms of secret possessions or secret intentions, the human race has interacted and evolved within the veils of secrecy – often regarded as the best form of security.

In the modern information age we live in, the notion of security often translates to the idea of confidentiality of information; especially of digital information utilized and transacted over various communication networks. It is important to note that the requirement for confidentiality and security is in fact entirely a social construct. If there was nobody in the universe interested in the piece of information you hold so dear, you may never feel the need for secure possession and transmission of information in the first place. In fact, we still judge security qualitatively in terms of the eagerness and competence of an adversary who may be interested in the information that is kept secure.

Cryptology – inherently a social science – refers to the art of bridging this bizarre gap between an entirely social notion called 'security', and the logical foundations of mathematics, computer science and allied domains.

1.1 Scope of this Thesis

Borrowing the basic notions from mathematics and computer science, the subject of cryptology has spread wide enough to be considered an independent discipline on its own. The structure of this subject is pictorially depicted in Figure 1.1 (idea from [111]), with a taxonomy of main cryptographic primitives.

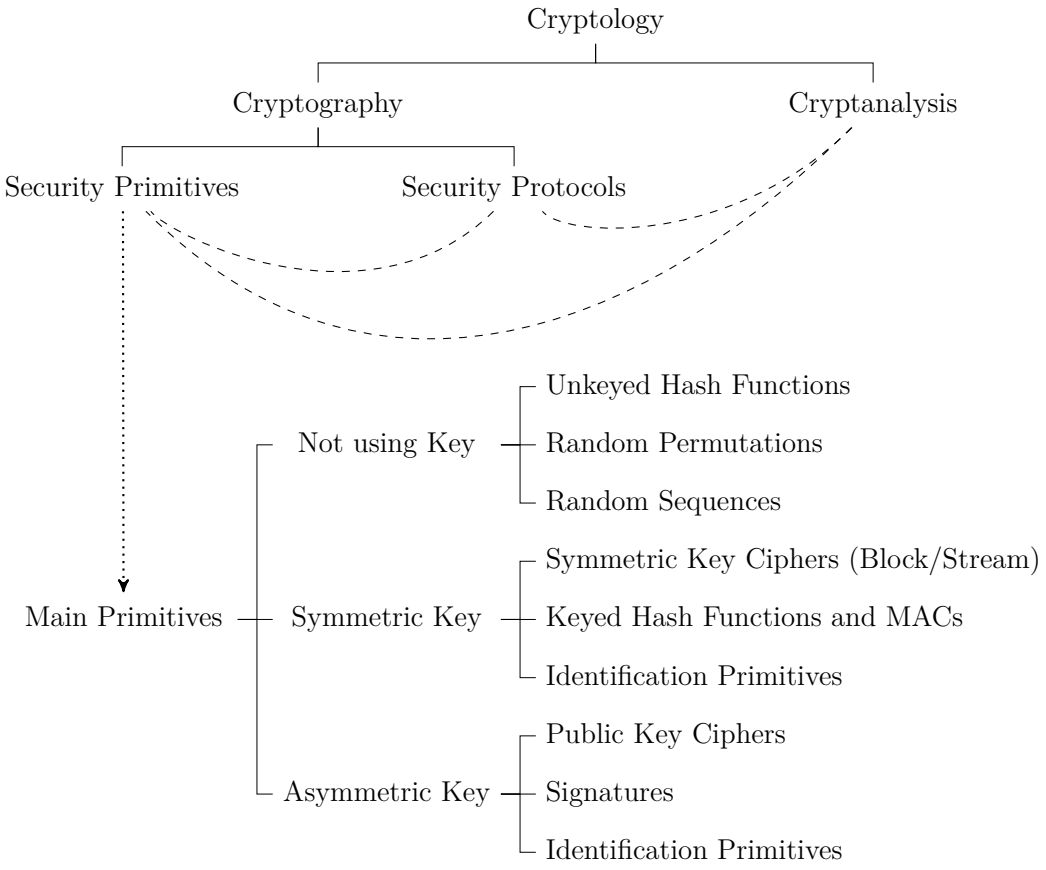

Figure 1.1: The scope, range and structure of Cryptology

Figure 1.1 connects the relevant pieces within the structure of the subject, although the list of primitives is not exhaustive. The reader may refer to [111, 142] for a detailed exposition of each topic. We consider the following path:

Security Primitives → *Symmetric Key Ciphers* → *Stream Ciphers.*

In this thesis, our focus is on the analysis and implementation of the most popular commercial stream cipher to date – RC4 [124].

1.2 Stream Ciphers

The domain of symmetric key cryptology contains two major cryptographic primitives – block ciphers and stream ciphers. Naively speaking, a block cipher encrypts block-by-block by applying a key dependent transformation on a block of message bits at a time. Whereas, a stream cipher produces a pseudorandom sequence of bits, called the *keystream*, and encryption is done by masking the plaintext (considered as a sequence of bits) by the keystream; using a simple XOR operation in general. The ciphertext thus obtained is also a sequence of bits of the same length as that of the plaintext. To increase efficiency in software applications, modern day stream ciphers are adopting a word-oriented approach, where word-wise (size of the word depends on machine architecture) operations are used instead of bitwise operations. However, the fundamental philosophy remains the same.

1.2.1 One-Time Pad and Perfect Secrecy

The motivation behind stream ciphers came from *One-Time Pad* (OTP), also called the *Vernam cipher*. OTP is a truly random bit-string of the same length as that of the plaintext – unique for each plaintext – which is bitwise XOR-ed with the plaintext to produce the ciphertext. Shannon [138] showed that OTP is *perfectly secure*, i.e., even for an adversary with unbounded computational power, it is impossible to derive any *new* information about the plaintext from the ciphertext beyond what is possible via random guess. In formal terms, one may define 'perfect secrecy' as follows (refer to [76] for details).

Definition 1.1. An encryption scheme is perfectly secret if for every probability distribution over the message space $\mathcal{M}$, every message $m \in \mathcal{M}$, and every ciphertext $c \in \mathcal{C}$ for which $\Pr[C = c] > 0$, $\Pr[M = m \mid C = c] = \Pr[M = m]$.

Intuitively, for a perfectly secret encryption, the ciphertext reveals no information about the plaintext over prior knowledge. To achieve ideal information theoretic 'perfect secrecy' in OTP, it is absolutely required that the masking (bitwise XOR) is done using a unique sequence of bits for each plaintext; owing to the name *one*-time-pad. In reality however, the OTP scheme is not practical, as we require to manage unique keys as large as the plaintexts.

1.2.2 Generic model for Stream Ciphers

Stream ciphers provide a practical alternative to OTP by relaxing the requirement of unconditional perfect-secrecy to its computational counterpart. Instead of using a unique truly random bit-string as long as the plaintext, as in the case of OTP, a stream cipher uses a short truly random string (called *key*) to efficiently generate a long string that behaves like a random string to a computationally bounded adversary. The generated string is not actually random but *looks like* a random string to the computationally bounded adversary; hence called a *pseudorandom* keystream. This serves the purpose of a truly random string for all practical requirements.

A stream cipher can be viewed as an efficient, deterministic algorithm, implementing a function $E : \{0,1\}^l \times \{0,1\}^s \longrightarrow \{0,1\}^n$ that maps an l-bit secret key $K = K_1K_2 \ldots K_l$ and an s-bit public initialization vector $IV = IV_1IV_2 \ldots IV_s$ to an n-bit keystream $Z = Z_1Z_2 \ldots Z_n$. This keystream Z is then bitwise XOR-ed with the plaintext $P = P_1P_2 \ldots P_n$ to produce the ciphertext $C = C_1C_2 \ldots C_n$. In order to be meaningful and practical, the length n of the generated keystream is generally greater than the length l of the secret key K. Figure 1.2 depicts this model of a stream cipher.

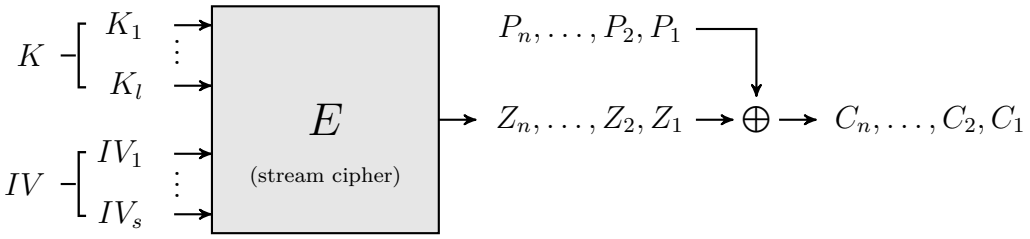

Figure 1.2: Generic Key-IV model of a stream cipher.

The model of a stream cipher presented here is that of a *binary additive synchronous stream cipher* and is not universally applicable. Although almost all modern-day stream cipher designs incorporate initialization vector, there are some classical stream ciphers that do not take IV as a default input. The focus of this thesis, RC4 stream cipher, does not take IV as a default input, but allows provision for it in practical protocols designed based on the cipher. Thus, Figure 1.2 does not depict the basic RC4 cipher, but illustrates the model of RC4-based designs that take IV as input, like WEP, WPA and SSL/TLS.

In theoretical formalization, a classical stream cipher without IV is idealized as a *pseudorandom generator* (PRG), a function $E : \{0,1\}^l \longrightarrow \{0,1\}^n$ mapping (or stretching) a short l-bit truly random secret key K to a long n-bit pseudorandom keystream Z (where $n > l$) that is computationally indistinguishable from a truly random string of the same length. However, a modern stream cipher with IV can be viewed as a *pseudorandom function* (PRF), a keyed function $E_K : \{0,1\}^s \longrightarrow \{0,1\}^n$, mapping s-bit public initialization vector IV to an n-bit keystream Z, where E_K is computationally indistinguishable from a random function over the same domain and range where key K is chosen uniformly at random from the key-space.

Stream Cipher viewed as a Finite State Machine

The functional structure of a generic stream cipher can be modeled as a *finite state machine* (FSM), as shown in Figure 1.3. Private key K and public initialization vector IV are used to generate the initial state S_0 through the Key-IV setup process. Subsequently, the r-th keystream bit (or word) Z_r is derived from the r-th state S_r through the output function f_{out}, and the FSM moves to the next state S_{r+1}, which is derived from the present state S_r through a state-update function f_{update}. The key K or the initialization vector IV are generally not used after the generation of the initial state S_0.

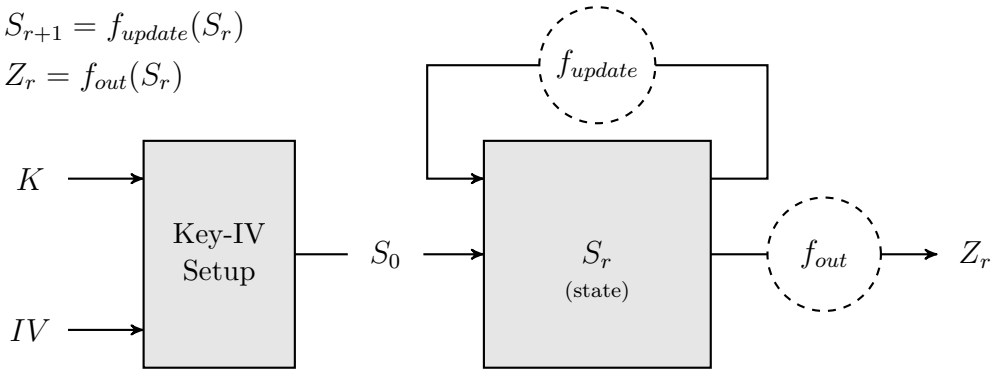

Figure 1.3: Stream cipher viewed as a finite state machine.

Note that the FSM has a finite number of states, and the state-update function being deterministic, the states repeat in a periodic fashion. Since the keystream output bits (words) Z_r are derived from the corresponding states through the deterministic function f_{out}, keystream bits (words) also repeat in a

periodic fashion. In case of stream ciphers considered for practical applications, it is required that this period of keystream repetition is as large as possible, in order to prevent certain attacks. Thus it is ensured during the design of the algorithm that the period of the stream cipher is sufficiently large.

Synchronous and Self-Synchronizing Stream Ciphers

The model we have described so far suits the class of *synchronous stream ciphers*. These stream ciphers, although most prominently studied and used in practice, have a practical drawback. Once the synchronization between the sender and the receiver is compromised, decryption of the ciphertext does not yield the corresponding plaintext at the receiver end. In such a case, the ciphers at the sender and the receiver ends need to re-synchronize for subsequent operation. This requirement for re-synchronization poses practical problems while operating a synchronous stream cipher over a noisy channel, where (burst) noise may frequently destroy parts of the communicated ciphertexts.

There is another class of stream ciphers known as *self-synchronizing stream ciphers*, where the r-th keystream bit (word) depends on the key as well as on some predetermined number (t) of previous ciphertext bits (words). So, the decryption works correctly if the previous t ciphertext bits (words) are received properly, irrespective of earlier deletions. Hence, even if a part of the communicated ciphertext is destroyed due to (burst) noise, only t subsequent deciphered plaintext bits (words) are affected; beyond which the decryption works correctly. Due to this self-synchronizing ability, the problem of re-synchronization of stream ciphers in a noisy channel may be overcome.

Practical example of a modern self-synchronizing stream cipher is Moustique [36], designed by Daemen and Kitsos in 2006. This is a Phase 3 candidate of the eSTREAM [40] project; a tweaked version of the original Phase 1 submission called Mosquito [35]. The tweaks were considered while designing Moustique in order to circumvent certain chosen ciphertext attacks [74] on the original design. Apart from dedicated designs, a block cipher in cipher feedback (CFB) mode can also be considered a self-synchronizing stream cipher. However in general, the class of self-synchronizing stream ciphers is difficult to design, less studied in the literature, and less frequently used in practice.

1.2.3 Cryptanalysis of Stream Ciphers

Instead of 'perfect secrecy', a computational notion of secrecy is ensured by the pseudorandom output sequence (keystream) generated by a stream cipher. By construction, any non-random event in the internal state or the keystream of a stream cipher that can be computationally identified, is not desirable from a cryptographic point of view. Thus it is imperative that a rigorous analysis of a stream cipher is performed to identify the presence of any such non-randomness in its design. There are some cryptanalytic attacks on stream ciphers which are relatively general in nature and any new stream cipher should exhibit sufficient immunity against these attacks. We present some attack models and techniques before proceeding any further.

Power of the Adversary

Cryptanalysis of cryptographic primitives are broadly classified under four basic models of security, each of which is based on a specific assumption on the power of the adversary (refer to [111, 142] for details).

- *Ciphertext only attack:* The adversary knows the ciphertexts of several messages, encrypted using the same key or different keys, but does not have the knowledge of the plaintext messages.
- *Known plaintext attack:* The adversary knows the plaintext-ciphertext pairs for several messages, where the ciphertexts have been generated by encrypting the plaintexts using the same key or different keys.
- *Chosen plaintext attack:* In this model, the adversary is given access to an encryption oracle, which produces ciphertexts for messages chosen by the adversary, under the same key or different keys.
- *Known and Chosen IV attack:* In this attack scenario, the adversary either knows or chooses some initialization vectors and obtains the corresponding keystreams generated by the cipher under a fixed key or different keys. This attack model is also known as 're-synchronization attack', and it assumes an underlying 'known plaintext model' to obtain the keystreams from the ciphertexts.

- *Chosen ciphertext attack:* This is the strongest of the attack models, where the adversary is given access to a decryption oracle that produces plaintexts corresponding to ciphertexts chosen by the adversary.

Given a plaintext-ciphertext pair for a stream cipher, it is trivial to obtain the keystream ($Z = P \oplus C$). Thus for known and chosen plaintext attack models, the output keystream of the stream cipher is always available to the adversary.

Modes of attack on Stream Ciphers

Depending on the attack model and the power of the adversary, there may be several modes of attacking a stream cipher. The common ones are as follows:

1. *Brute force attack:* This attack, also known as the *exhaustive key search*, is the most general attack applicable against any cipher. The idea is to search the key space exhaustively in order to find correct key [111, 142].

2. *Algebraic attack:* This is a 'known plaintext attack' where the goal of the adversary is to recover the key. Broadly speaking, the attack starts by generating algebraic equations involving key bits and keystream bits, and finally tries to solve the generated system of equations to get the values of the unknown key bits from the known keystream bits (obtained from plaintext-ciphertext pairs). Originally proposed for public key cryptanalysis [78], the algebraic attacks have found strong relevance in analyzing practical stream ciphers as well [6, 10, 11, 33, 34, 63].

3. *Time-memory trade-off attack:* This is a 'chosen plaintext attack', and in the basic form, it aims to find a trade-off between brute force and dictionary attacks for searching the key space [38, 66]. For brute force attack, time complexity of the attack is of the order of the key space, while space complexity is negligible. For dictionary attack, space complexity is of the order of the key space, while time complexity is negligible. In time-memory trade-off attack, the goal is to bring down both time and space complexity in between these two extremes. Although the idea was first proposed to attack DES block cipher [66], it later extended quite successfully to cryptanalysis of stream ciphers as well [23, 24, 51, 69, 126].

4. *Differential attack:* This is a 'chosen plaintext attack', although with some sophisticated techniques it can be converted into a 'known plaintext attack', or even to a 'ciphertext only attack'. The idea is to observe the differences in pairs of ciphertexts generated from pairs of plaintexts chosen according to some fixed differences. If these two differences are strongly related, then such an information can be exploited in various ways, depending upon the context. The idea was first proposed for block ciphers [20, 21], and was later extended to stream cipher analysis. To date, the idea of differential cryptanalysis has been exploited against several practical stream ciphers [17, 44, 73, 151, 152].

5. *Linear or Correlation attack:* This attack targets LFSR based stream ciphers, more specifically stream ciphers with a non-linear combiner, where outputs of several LFSRs are combined using a non-linear boolean function to produce the final output. The goal is to recover the states of the individual LFSRs from the given output keystream. A divide-and-conquer approach is taken to reduce the time-complexity of brute force search on the states of the LFSRs. There exist numerous correlation attacks (and variants) on stream ciphers in the literature, and one may refer to [29, 32, 49, 50, 55, 56, 71, 72, 90, 109, 110] for further details.

6. *Statistical Distinguishing attack:* Stream ciphers theoretically claim to be pseudo-random generators (PRGs), capable of producing stream of bits that is indistinguishable from a truly random bit-stream from the point of view of a computationally bounded adversary. Thus, efficient distinguishing attacks based on statistical weaknesses can pose generic threat against any stream cipher construction. A prime example of this class of attacks is the linear statistical distinguisher, introduced by Golic [48] in 1994. The attack linearly models an arbitrary binary keystream generator into a linear feedback shift register of bounded length. This in turn exposes any generic design to subsequent structure-dependent and initial-state-dependent statistical tests, potentially resulting in correlation attacks. This strategy has proved to be quite effective against practical stream ciphers [52, 53].

7. *Side-channel attack:* In contrast to the previous ones, this attack does not target the actual algorithmic description of a cipher; but targets its implementation in hardware and software [141]. Any implementation of a cipher leaks out information in terms of operating parameters such as power consumption, electromagnetic radiation, timing for operations, etc. These additional sources of information are known as 'side-channels'. Some common variants of side-channel attacks are as follows:

   a) *Timing analysis* – The adversary measures the time required to perform certain operations of the cipher; especially in software. This pattern in time requirements is observed and exploited in timing attacks [82] – quite applicable towards certain modern stream ciphers [88, 153].

   b) *Power analysis* – The adversary measures the power consumed during certain cipher operations; especially in hardware implementations, namely RFID tags or smart cards. The idea is to guess a few bits of the key by observing any pattern in power consumption or dissipation [83]. The idea is well exploited against certain stream ciphers [43, 87].

   c) *Fault attack* – The adversary can inject some faults into the data during the operation cycle of a cipher, and can exploit the variations resulting from the fault. Faults may be injected into the hardware (sometimes software) implementation of a cipher by various means, and the effects of the fault may be monitored at the output to extract information about the internal operating state of the cipher [22, 27]. Fault attacks are frequently mounted on stream ciphers in practice [7, 8, 18, 67, 68].

Goals of the Adversary

Whatever be the model or technique of the attack, general goals of an adversary mounting an attack on a stream cipher can be summarized as follows:

Key recovery – This is the dream goal of the adversary, and it is the strongest form of attack against stream ciphers, or any other primitive in general.

State recovery – This can be an intermediate goal of the adversary. Once the internal state of a stream cipher is recovered at any stage of pseudo-random generation, it is possible to generate subsequent keystream bits

(words) using the known output function and the state-update function. In the case where the key-generation and key-scheduling functions are reversible (one-to-one), it is also possible to extend a state recovery attack to a key recovery attack on the cipher. This is the case for the Grain family of stream ciphers [64, 65]. For RC4 however, the key-scheduling is not one-to-one, and thus extending a state recovery attack to a key recovery attack on RC4 becomes a probabilistic process [117].

Distinguishing attack – The goal of a distinguishing attack on a stream cipher is to distinguish the generated keystream of the cipher from a truly random bit-stream with non-negligible probability. Although it may not lead to either state or key recovery, a distinguishing attack challenges the basic claim of a stream cipher, that is, pseudorandom generation.

Key correlation – In this case, the attacker aims at finding 'leaks' of the secret key in the output keystream. If any part of the secret key is correlated to the output keystream, it may be possible to trace back and recover that part of the secret key with probability more than that of random guess. In such a case, the attacker may identify and exploit the key correlations towards a potential key recovery attack on the cipher.

Our focus for analysis of stream ciphers in this thesis is on *distinguishing attacks* and *key correlations*, discussed later in Chapter 2. First we take a look at some practical designs of stream ciphers from the literature.

1.2.4 Practical Designs

The most important and cryptographically significant goal of a stream cipher is to produce a pseudorandom sequence of bits or words using a fixed length secret key, often paired with a fixed length public initialization vector. Over the last three decades of research and development in stream ciphers, a number of designs have been proposed and analyzed by the cryptology community. We briefly discuss a few major stream ciphers, relevant in terms of practical application and cutting-edge design, as follows.

RC4 Stream Cipher

RC4, also known as Alleged RC4 or ARC4, is the most widely deployed commercial stream cipher, having applications in network protocols such as SSL, WEP, WPA and in Microsoft Windows, Apple OCE, Secure SQL, etc. It was designed in 1987 by Ron Rivest for RSA Data Security (now RSA Security). The design was a trade secret since then, and was anonymously posted on the web in 1994. Later, the public description was verified by comparing the outputs of the leaked design with those of the licensed systems using proprietary versions of the original cipher. Although the public design has never been officially approved or claimed by RSA Security to be the original cipher, this note [124] confirms that the leaked code is indeed RC4. The cipher has gained immense popularity for its intriguing simplicity, which has also made it widely accepted for numerous software and web applications.

Bluetooth™ Stream Cipher

Bluetooth™ is one of the major modern technologies for wireless communication, prevalent in an array of practical devices. The technology was developed by the Bluetooth Special Interest Group (SIG), formed in 1998. The technology has been embraced by all companies in the communication business ever since. For confidentiality in Bluetooth transmission, the E0 stream cipher is used as the pseudorandom keystream generator [26]. The cipher follows the standard design model of a combiner generator using linear feedback shift registers, where the keylength is typically 128 bits. Several attacks have been mounted on E0 since 1999, resulting in practical and near-practical breaches [90–93].

GSM Stream Ciphers

A5/1 and A5/2 stream ciphers, designed around late 1980's, are used to provide privacy in the GSM cellular network. A5/2 is a (deliberately) weakened version of A5/1, created for certain export regions. Both the ciphers A5/1 and A5/2, initially kept secret, became public in 1994 through leaks and reverse engineering [28]. After several minor and major attacks [9, 16, 24, 39, 47, 51] on A5/1 and A5/2 published during 1994–2006, the GSM Association mandated

that the GSM phones will not support A5/2 any more, and usage of A5/1 was mandated by the 3GPP association. Later in the 3G cellular systems, the keystream generation algorithm for privacy was modified to A5/3, which uses the block cipher KASUMI.

4G Stream Ciphers

In the race towards 4G mobile technology, 3GPP LTE Advanced [3] is the leading contender. For LTE Advanced technology, the chosen security algorithms for encryption and authentication employ two different stream ciphers – SNOW 3G [2] and ZUC [1]. While SNOW 3G had already been deployed in the earlier 3G platform, along with KASUMI, the other cipher ZUC is a brand new design. Both the ciphers are based on similar design principles using word-oriented linear feedback shift registers, and are used in the LTE Advanced technology within a portfolio, along with the block cipher AES-128.

eSTREAM Portfolio Ciphers

The eSTREAM project, coordinated under ECRYPT framework from 2004 to 2008, was dedicated towards stream cipher research, with an aim to "identify new stream ciphers suitable for widespread adoption". Following the call for primitives, thirty-four stream ciphers were submitted to eSTREAM and an overall evaluation was done in three phases. The project came to an end in 2008 and a portfolio of seven stream ciphers was announced [40]. The current portfolio contains HC-128, Rabbit, Salsa20/12 and SOSEMANUK under the Software (SW) profile, and Grain v1, MICKEY v2 and Trivium under the Hardware (HW) profile. These ciphers follow cutting-edge design principles, and are projected as the stream ciphers of the future.

1.3 Motivation of the Thesis

So far we have discussed the generic description of stream ciphers, and have listed some of the most prominent designs in practice:

RC4, E0, A5/1, A5/2, SNOW 3G, ZUC, HC-128, Rabbit,

Salsa20/12, SOSEMANUK, Grain v1, MICKEY v2, Trivium.

Given this array of major practical stream ciphers in the literature, choice of the specific cipher RC4 for analysis and implementation deserves an explanation. The main motivation of this thesis, focused on RC4 analysis and implementation, will be summarized after a short description of the cipher.

1.3.1 Description of RC4

One of the main ideas for building a stream cipher relies on constructing a pseudorandom permutation and thereafter extracting a pseudorandom sequence of words from this permutation. RC4 follows this design principle to extract pseudorandom bytes from pseudorandom permutations.

The cipher consists of two major components, the Key Scheduling Algorithm (KSA) and the Pseudorandom Generation Algorithm (PRGA). The internal state of RC4 contains a permutation of all 8-bit words, i.e., a permutation of $N = 2^8 = 256$ bytes, and the KSA produces the initial pseudorandom permutation of RC4 by scrambling an identity permutation using the secret key k. The secret key k of RC4 is of length typically between 5 to 32 bytes, which generates the expanded key K of length $N = 256$ bytes by simple repetition. If the length of the secret key k is l bytes (typically $5 \leq l \leq 32$), then the expanded key K is constructed as $K[i] = k[i \bmod l]$ for $0 \leq i \leq N - 1$. The initial permutation produced by the KSA acts as an input to the next procedure PRGA that generates the keystream. The RC4 algorithms KSA and PRGA are depicted in Figure 1.4.

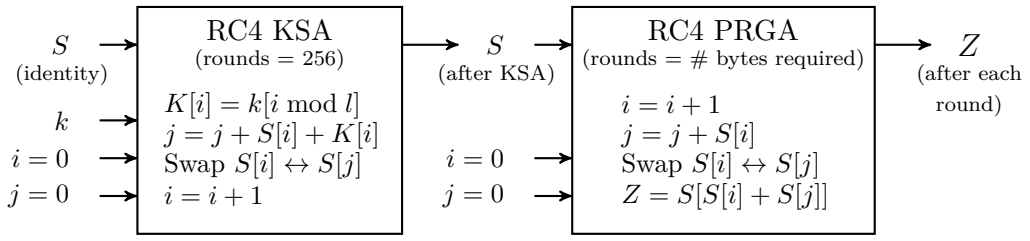

Figure 1.4: Description of RC4 stream cipher.

Notation: For round $r = 1, 2, \ldots$ of RC4 PRGA, we denote the indices by i_r, j_r, the keystream output byte by Z_r, the output byte-extraction index as

$t_r = S_r[i_r] + S_r[j_r]$, and the permutations before and after the swap by S_{r-1} and S_r respectively. After r rounds of KSA, we denote the state variables by adding a superscript K to each variable. By S_0^K and S_0, we denote the initial permutations before KSA and PRGA respectively. Note that S_0^K is the identity permutation and $S_0 = S_N^K$ is the permutation obtained right after the completion of KSA. Throughout this thesis, all additions (subtractions) in context of RC4 are to be considered as 'addition (subtraction) modulo N', and all equalities in context of RC4 are to be considered as 'congruent modulo N'.

1.3.2 Choice of RC4 for analysis and implementation

A closer look at RC4, as described in Figure 1.4 and again presented as a formal algorithm in Table 1.1, leaves one wondering whether these four lines of core code, too simple even for a toy cipher, can generate a pseudorandom keystream as demanded of a stream cipher. That's the beauty of RC4! Since its public reveal through Internet leakage in 1994 [124], the sheer elegance and enigmatic appeal of the cipher has roots in its simple design, which is undoubtedly the simplest for any practical cryptographic algorithm to date.

Table 1.1: The RC4 Algorithm: KSA and PRGA.

Key Scheduling (KSA)	Pseudorandom Generation (PRGA)
Input: Secret Key k. **Output**: S-Box S_0 generated by k. Initialize $S_0^K = \{0, 1, 2, \ldots, N-1\}$, $K[i] = k[i \bmod l]$ and $i_0^K = j_0^K = 0$; **for** $r = 1, \ldots, N$ **do** $\quad j_r^K = j_{r-1}^K + S_{r-1}^K[i_r^K] + K[i_r^K]$; $\quad$ Swap $S_{r-1}^K[i_r^K] \leftrightarrow S_{r-1}^K[j_r^K]$; $\quad i_r^K = i_{r-1}^K + 1$; **end**	**Input**: S-Box S_0, output of KSA. **Output**: Random stream Z. Initialize the counters: $i_0 = j_0 = 0$; **for** $r = 1, 2, \ldots$ **do** $\quad i_r = i_{r-1} + 1$; $\quad j_r = j_{r-1} + S_{r-1}[i_r]$; $\quad$ Swap $S_{r-1}[i_r] \leftrightarrow S_{r-1}[j_r]$; $\quad$ Output $Z_r = S_r[S_r[i_r] + S_r[j_r]]$; **end**

The simplicity in design has attracted everyone towards this cipher. It has been a hit in the software industry for decades, and has been adopted as the core cipher for numerous web and software applications like Microsoft Windows, Apple OCE, Secure SQL, to name a few. The most pervasive application of RC4 however, has been in standardized web and network security protocols.

RC4 in Security Protocols

IEEE 802.11 standard protocol for WiFi security used to be Wired Equivalent Privacy (WEP), which has now been replaced by Wi-Fi Protected Access (WPA). Both WEP and WPA use RC4 as their core module. In case of WEP, the protocol uses RC4 with a pre-shared key appended to a public initialization vector (nonce) for self-synchronization. Using the technique of related key attacks on RC4, this scheme has been broken through passive full-key recovery attacks, and thus WEP is considered insecure in practice.

To mitigate this problem, WEP has been replaced by WPA. The goal of WPA was to resolve all security threats of WEP. However, the original WEP protocol was extensively adopted by the industry, and it was already implemented in several commercial products, both in software and hardware. This rendered a design of WPA from scratch quite impractical and costly. The work-around was to fix the full-key recovery problems of WEP using a patch, as minimal as possible, on top of the original protocol. WPA accomplished this by introducing a key fixing function to feed the RC4 core with different unrelated keys for each packet. In addition to this, WPA incorporated a packet integrity protection scheme to prevent replay and alteration of the initialization vector, which is a main tool in active attacks.

It is nowadays recommended by the Wi-Fi alliance to use WPA2, which uses AES block cipher as the core instead of RC4. However for hardware based applications and products using WEP, and later WPA, it is neither cost effective nor easy to migrate completely away from the RC4 core. Even with the proven weaknesses in WEP, a large number of applications still have an active option for the protocol, and users quite frequently opt for the simplicity of WEP over WPA or WPA2. Thus, RC4 continues to dominate the domain of network security to date, through the most widely used IEEE 802.11 security protocols WEP and WPA.

Another prominent use of RC4 in web security is through Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), which provide communication security over the Internet. The RC4 suite is considered to be one of the best choices for use in SSL/TLS, as it can prevent popular attacks on the protocol that primarily target the CBC cipher suite. Even though a

new attack [5] on the RC4 suite of TLS has been proposed in 2013, it still remains the most popular cipher for the protocol. It is even being debated whether one should fix the recently discovered problems with a patch.

In short, the users and the industry have been obsessed with RC4 for all possible web and network security solutions for more than two decades, and the cipher still remains the most popular, most studied and most debated symmetric key algorithm in practice.

RC4 in Academic Literature

Since 1994, when the design of this cipher came public, cryptanalysts around the world have targeted the simplest modern cipher, in both its theoretical form as well as in its practical applications. Although there had been successful breaches against some practical protocols using RC4, the stand-alone cipher is still quite robust at the core. It is this stunning combination of robustness and simplicity of RC4 that has attracted analysts for the last two decades.

There have been a large number of academic papers on RC4 analysis published during the last two decades at top cryptography conferences and journals. The first attack on RC4 published in the academic literature was a linear statistical distinguisher presented by Golic in EUROCRYPT'97 [52], later extended to an IEEE-IT publication [53] in 1999. Analytic results on the cipher and related security protocols have resulted in three Master's theses [100, 108, 145], two PhD theses [116, 134], and one book [119] to date. In addition to this, parts of several other research theses and monographs have resulted from RC4 cryptanalysis and related studies.

In spite of the vast literature of RC4 analysis since its public appearance in 1994, the field is nowhere close to saturation. RC4 is thought to be 'broken' by some, and yet, attempts at 'breaking' the cipher and related protocols are getting published at top conferences, like FSE'08 [96], CRYPTO'08 [107], FSE'09 [104], SAC'10 [136], SAC'11 [131], FSE'11 [98], EUROCRYPT'11 [137], quite regularly over the last five years. Our work on RC4 analysis is due to appear in the Journal of Cryptology in 2013 [132]. Most recently, the IACR conference FSE'13 has witnessed two contributory papers [70, 135] and an invited talk by Dan Bernstein [14], presenting recent results on analysis of

RC4 and its practical applications in WEP, WPA and TLS. This corroborates the fact that the enigmatic charm of RC4 is still alive and thriving in the community; and that it still offers a plethora of research problems to inspire cryptanalysts worldwide, amateurs and veterans alike.

Intellectual Properties based on RC4

RC4 has been the cipher of choice for numerous practical implementations over the last two decades. It has been projected primarily as a software stream cipher, as a software implementation of RC4 is quite simple for any general purpose processor and any software platform. Detailed comparison of the software performance of RC4 against modern stream ciphers is given in [41].

However, if we take a look at the other side of the coin, the nontrivial aspect of hardware implementation of RC4, we find something interesting. There have been a few hardware designs proposed in the literature, and the motivation has been efficiency in speed versus the hardware footprint. In 2003, a patent by Matthews Jr. [106] was disclosed, which provided an efficient RC4 architecture using hardware pipelining and block RAMs. After a gap of five years, another patent by Matthews Jr. [105] was disclosed in 2008, which proposed a new design for RC4 hardware using pipeline architecture. The patents acquired on hardware designs of RC4 prove the practical viability of its implementations, and allures a researcher to drive for an even better architecture.

Choice of RC4

In view of the discussion so far, RC4 stands out amongst all modern stream ciphers due to the following properties:

- Most widely used stream cipher – especially in web and network security.
- Simplest and oldest cipher to withstand decades of analysis, thus commanding the respect and interest of the cryptanalyst community.
- Well studied, well understood, but still offering bouquet of research problems; even after two decades of analysis and attacks.
- Software cipher with patented hardware designs – unusual and intriguing.

In summary, there is no other stream cipher that is interesting enough in comparison with RC4 if one aims at cryptanalysis and practical implementation. Thus, we choose RC4 for the problems discussed in this thesis, and the motivation for the particular research problems arises as follows.

1.3.3 Motivation for RC4 Analysis

There is an open problem for everyone who wants to explore the existing literature of RC4 analysis, as detailed in Chapter 2. The previous cryptanalytic results on the cipher, covering over fifty prominent research publications, have opened more problems and research directions in this field than for any other stream cipher. The problems discussed in this thesis are motivated as follows.

Effect of keylength on RC4 biases

In SAC 2010, Sepehrdad, Vaudenay and Vuagnoux [136] reported a strong empirical bias in the event $(S_{16}[j_{16}] = 0 \mid Z_{16} = -16)$ and mentioned that no explanation of this bias could be found. A related bias of the same order involving the event $(S_{17}^{K}[16] = 0 \mid Z_{16} = -16)$ has been empirically reported in [134, Section 6.1], and this has been used to mount WEP and WPA attacks on RC4. Our first motivation was to investigate the source of the number 16 in both the cases, which led to the following problem.

Problem 1a. Some of the empirical biases observed by Sepehrdad, Vaudenay and Vuagnoux [136] seem to be related to the length of the secret key used in RC4. If this is the case, is it possible to identify and prove such general relations between the keylength to the keystream biases in RC4?

Detailed study and analysis of this problem have led us to keylength-dependent conditional biases in RC4, a completely new field of RC4 analysis, hitherto unknown to the community. We present a detailed study in Chapter 3.

In a recent invited talk [14] at FSE 2013, Bernstein disclosed an attack on TLS, which in fact is one of the most significant attacks on any RC4 based protocol. In this work [5], the authors AlFardan, Bernstein, Paterson, Poettering and Schuldt identified a strong negative bias in Z_1 towards 129 (for $N = 256$),

which was not observed in the earlier works [112, 132] on the first keystream byte Z_1. We observe (experimentally) that this bias is in fact dependent on the length of the RC4 secret key, and occurs only for certain values of the keylength. To understand this issue better, we take up the following problem.

Problem 1b. Investigate the negative bias of Z_1 towards 129 ($N = 256$), as observed by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5], and identify its keylength dependence characteristics.

We discover an intriguing connection of this bias with "anomalies" in the distribution of the state array S_0 after the RC4 KSA, a mysterious problem open for more than a decade, since the observation by Mantin [100]. In Chapter 3, we present our observations and proofs in connection with the keylength dependence of this bias and that of the anomalies.

Discovery and proof of biases involving state variables

Along a similar line of investigation of other empirical biases presented in SAC 2010 by Sepehrdad, Vaudenay and Vuagnoux [136], we find several biases that depend on the state variables of RC4. These biases are quite strong and may be useful in practical attacks, although none were theoretically justified. In addition, we observe critical non-randomness in the behavior of the index j over the first few rounds of RC4 PRGA. This again, may pose serious security threats, once completely understood. Quite often the quest for theoretical justification of experimental results open up new avenues to explore further. Thus, we raise the following research problems to probe deeper.

Problem 1c. Prove all known significant biases of RC4 involving the state variables, as empirically observed in [136]. In addition, is it possible to identify and prove other interesting biases of similar nature?

Problem 1d. It seems that the index j exhibits certain non-random behavior in the initial rounds of RC4 PRGA. Is it possible to completely characterize the (non-)randomness of index j throughout RC4 PRGA?

Investigation on these problems led to several theoretical results that complement the current literature of RC4 biases. We discuss the results in Chapter 4.

Discovery and proof of keystream biases

In the quest for proving known significant biases, we turn our attention to the biases in the keystream, most effective for practical attacks. We notice that although the best known bias of RC4, for the second output byte Z_2, was proved [103] back in 2001, no serious attempt has been made to find and prove all prominent biases in the other initial bytes of RC4.

In CRYPTO 2002, Mironov [112] observed that the first byte Z_1 of RC4 keystream has a negative bias towards zero, and also found an interesting non-uniform probability distribution (similar to a sine curve) for all other values of this byte. However, the theoretical proof remained open for almost a decade. We take up the challenge.

Problem 1e. Theoretically justify the sinusoidal probability distribution of Z_1 and its negative bias towards zero, as observed by Mironov [112].

In Chapter 5, for the first time in the literature, we derive the complete theoretical distribution of Z_1, and prove all related biases.

Looking back at the FSE 2001 paper by Mantin and Shamir [103], which proved the bias in the second byte Z_2 towards zero, we find that the authors claimed of no such bias existing in any subsequent byte in the keystream. We decide not to take this claim on face-value, but to explore further.

Problem 1f. Identify all significant biases towards zero in the initial bytes of RC4 keystream (Z_3 to Z_{255}), and prove all subsequent results.

During this investigation, contrary to this claim by Mantin and Shamir [103], we find that all the initial bytes Z_3 to Z_{255} of RC4 keystream are biased towards zero. We present a detailed account of this result in Chapter 5.

In the recent work of AlFardan, Bernstein, Paterson, Poettering and Schuldt [5, 14], to appear in USENIX Security Symposium 2013, the authors identified all initial keystream biases in RC4 through extensive experimentation, and utilized those to mount attacks on TLS and WPA. This identified several unproved biases in the initial bytes of the RC4 keystream. To complement and complete the RC4 literature of biases, we target the following.

Problem 1g. Prove all significant biases in the initial bytes of RC4 keystream identified by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5].

Some of the identified biases of RC4 already has theoretical justification in the literature [70, 127, 131, 132]. We attempt only the unproved or the partially proved ones, and present a systematic account of all such proofs in Chapter 5.

It is well-known that biases in the initial rounds of RC4 has no effect if one throws away some initial bytes from the keystream of RC4. This naturally motivates a quest for long-term biases in the RC4 output, if any exists.

Problem 1h. Experimentally discover and subsequently prove biases in RC4 keystream which remain effective even after discarding the initial bytes.

In Chapter 5, we observe and prove a new long-term bias in RC4 keystream.

In summary, we consider the aforesaid eight problems 1a-1h as our target for RC4 analysis, and we present all relevant results in Part I on this thesis. The results thus obtained fit nicely into the flow of the RC4 literature to date, as depicted in Figure 2.1 of Chapter 2.

1.3.4 Motivation for RC4 Implementation

Efficiency in terms of 'keystream throughput' has always been a benchmarking parameter for stream ciphers. The efficiency of the RC4 obviously depends on the efficiency of KSA and PRGA. While the KSA invokes a fixed cost for generating the initial pseudorandom state, the PRGA incurs a variable cost in terms of the number of keystream bytes to be generated. An efficient implementation of RC4 would aim to minimize the cost for per round of KSA and PRGA to provide better throughput.

In this thesis, we focus on efficient high-throughput hardware implementation of the cipher. The main motivation is to test the limits to which RC4, the popular 'software' stream cipher, can be pushed in terms of throughput efficiency on a dedicated hardware platform. If the hardware throughput of RC4 is sufficiently good as per current standards, we will have a stronger case in support of using this time-tested cipher. Furthermore, systematic study of

the exploitable fine-grained parallelism may aid software developers to attempt better performance in modern parallel processors.

Although there have been a few attempts to propose efficient hardware implementation [46, 62, 89] of RC4, the basic issue remained ignored that the design motivation should be initiated by the following question:

> "*In how many clock cycles can a keystream byte be generated at the PRGA stage in an RC4 hardware?*"

To the best of our knowledge, this line of thought has never been studied and exercised in a disciplined manner in the literature, which is quite surprising.

A 3 cycles-per-byte efficient implementation of RC4 on a custom pipelined hardware was first proposed by Kitsos et al [79] in 2003. In the same year, a patent by Matthews Jr. [106] was disclosed, which provided a similar 3 cycles-per-byte architecture using hardware pipelining. After a gap of five years, another patent by Matthews Jr. [105] was disclosed in 2008, which proposed a new design for RC4 hardware using pipeline architecture. This could increase the efficiency of the cipher to obtain 1-byte-per-cycle in RC4 PRGA. To the best of our knowledge, no further efficiency improvement for RC4 hardware has been proposed in the existing literature.

We pose and answer the following research problems in this thesis:

Problem 2a. Is it possible to provide a simpler alternative to the best existing designs for RC4 hardware that would yield the same throughput?

Problem 2b. Is it possible to design a new RC4 hardware that would yield a better throughput compared to the best existing architecture?

We discuss and present solutions to both the problems in this work. Problem 2a has been treated in Chapter 7, while Problem 2b has been solved in Chapter 8.

1.4 Organization of the Thesis

As outlined in Sections 1.3.3 and 1.3.4, we consider the following ten research problems as a backbone for this thesis on analysis and implementation of RC4.

Problem 1a. Some of the empirical biases observed by Sepehrdad, Vaudenay and Vuagnoux [136] seem to be related to the length of the secret key used in RC4. If this is the case, is it possible to identify and prove such general relations between the keylength to the keystream biases in RC4?

Problem 1b. Investigate the negative bias of Z_1 towards 129 ($N = 256$), as observed by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5], and identify its keylength dependence characteristics.

Problem 1c. Prove all known significant biases of RC4 involving the state variables, as empirically observed in [136]. In addition, is it possible to identify and prove other interesting biases of similar nature?

Problem 1d. It seems that the index j exhibits certain non-random behavior in the initial rounds of RC4 PRGA. Is it possible to completely characterize the (non-)randomness of index j throughout RC4 PRGA?

Problem 1e. Theoretically justify the sinusoidal probability distribution of Z_1 and its negative bias towards zero, as observed by Mironov [112].

Problem 1f. Identify all significant biases towards zero in the initial bytes of RC4 keystream (Z_3 to Z_{255}), and prove all subsequent results.

Problem 1g. Prove all significant biases in the initial bytes of RC4 keystream identified by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5].

Problem 1h. Experimentally discover and subsequently prove biases in RC4 keystream which remain effective even after discarding the initial bytes.

Problem 2a. Is it possible to provide a simpler alternative to the best existing designs for RC4 hardware that would yield the same throughput?

Problem 2b. Is it possible to design a new RC4 hardware that would yield a better throughput compared to the best existing architecture?

Structuring the material

This thesis is in two parts. The eight problems 1a–1h are considered our target for RC4 analysis, and we present all relevant results in Part I of this thesis.

The last two problems 2a–2b are studied as a part of RC4 implementation, and all relevant results are presented in Part II of this thesis. In summary, we answer the ten research problems 1a–1h and 2a–2b posed above, in accordance with the structure described in Table 1.2.

Table 1.2: Structure of the Thesis

Chapter 1 – Preliminaries and Motivation	
Part I – Analysis of RC4	Part II – Implementation of RC4
Chapter 2 – Overview	Chapter 6 – Overview
Chapter 3 – Problems 1a, 1b	Chapter 7 – Problem 2a
Chapter 4 – Problems 1c, 1d	Chapter 8 – Problem 2b
Chapter 5 – Problems 1e, 1f, 1g, 1h	
Chapter 9 – Conclusion and Open Problems	

The reader may choose to combine Chapter 1 and Chapter 9 with either Part I or Part II to get two independent self-contained pieces to read.

Publications included in this Thesis

This thesis is built upon two journal papers [129, 132], four conference papers [98, 99, 131, 133], and one IACR ePrint report [128].

The following publications are included in Part I of this thesis. The journal paper [132] is a consolidation and considerable extension of the conference papers [98, 131], while the most recent works [99, 128] are independent of these. The results presented in these papers are organized categorically, and hence assembled and distributed as required, over Chapters 3, 4 and 5 of this thesis.

1. Sourav Sen Gupta, Subhamoy Maitra, Goutam Paul, and Santanu Sarkar. (Non–)random sequences from (non–)random permutations – analysis of RC4 stream cipher. Journal of Cryptology, 2013. To appear. Published online Dec 2012, DOI: 10.1007/s00145-012-9138-1. **Ref.** [**132**]

2. Santanu Sarkar, Sourav Sen Gupta, Goutam Paul, and Subhamoy Maitra. Proving TLS-attack related open biases of RC4. IACR Cryptology ePrint Archive, 2013:502, 2013. **Ref.** [**128**]

3. Subhamoy Maitra and Sourav Sen Gupta. New long-term glimpse of RC4 stream cipher. In Aditya Bagchi and Indrakshi Ray, editors, ICISS, volume 8303 of Lecture Notes in Computer Science, pages 230–238. Springer, 2013. **Ref.** [**99**]

4. Subhamoy Maitra, Goutam Paul, and Sourav Sen Gupta. Attack on broadcast RC4 revisited. In FSE, volume 6733 of Lecture Notes in Computer Science, pages 199–217. Springer, 2011. **Ref.** [**98**]

5. Sourav Sen Gupta, Subhamoy Maitra, Goutam Paul, and Santanu Sarkar. Proof of empirical RC4 biases and new key correlations. In Selected Areas in Cryptography, volume 7118 of Lecture Notes in Computer Science, pages 151–168. Springer, 2011. **Ref.** [**131**]

The following publications are included in Part II of this thesis. The journal paper [129] is a considerable extension of the conference paper [133]. It is mostly the material of [133] that is presented in Chapter 7, while the extended portion in [129] is discussed in Chapter 8 of this thesis.

6. Sourav Sen Gupta, Anupam Chattopadhyay, Koushik Sinha, Subhamoy Maitra, and Bhabani P. Sinha. High-performance hardware implementation for RC4 stream cipher. IEEE Transactions on Computers, 62(4):730–743, 2013. **Ref.** [**129**]

7. Sourav Sen Gupta, Koushik Sinha, Subhamoy Maitra, and Bhabani P. Sinha. One byte per clock: A novel RC4 hardware. In INDOCRYPT, volume 6498 of Lecture Notes in Computer Science, pages 347–363. Springer, 2010. **Ref.** [**133**]

Part I

Analysis of RC4 Stream Cipher

Chapter 2

Overview of RC4 Analysis

The goal of RC4 is to produce a pseudorandom sequence of bytes from the internal permutation, which in turn should be pseudorandom. Hence, one of the main ideas for RC4 cryptanalysis is to investigate for *biases*, that is, statistical weaknesses that can be exploited to computationally distinguish the keystream of RC4 from a truly random sequence of bytes with a considerable probability of success.

The target of an attack on RC4 may be to exploit the non-randomness in the internal state of the stream cipher, or the non-randomness of byte-extraction from the internal permutation. Both ideas have been put to practice in various ways in the literature, and the main theme of attacks on RC4 has been in four major directions, as follows.

1. Weak keys, Key recovery from the internal state, and related attacks [4, 13, 15, 17, 31, 60, 77, 104, 117, 125, 150]
2. Key recovery from the keystream [44, 80, 84, 85, 101, 134–137, 146, 147, 149]
3. State recovery from the keystream [54, 57, 81, 107, 113, 139, 148]
4. Biases, Distinguishers, and related attacks [5, 12, 45, 52, 70, 75, 98, 102, 103, 112, 118, 125, 127, 128, 131, 132, 134, 136, 137]

Figure 2.1 gives a chronological summary of the major RC4 cryptanalytic results to date. We also describe, in brief, each of these major attacks to form a comprehensive overview of the current literature on RC4 analysis.

	Weak keys and Key recovery from state	Key recovery from keystream	State recovery attacks	Biases and Distinguishers
1995	◇ Roos weak keys [125] ◇ Wagner weak keys [150]			◇ Roos biases [125]
1996				◇ Glimpse bias [75]
1997				◇ Golic longterm bias [52]
1998			◇ Branch and bound [81] ◇ Cycle structures [113]	
2000	◇ Related key-pairs [60]		◇ Iterative probabilistic cryptanalysis [54]	◇ Digraph biases [45]
2001		◇ FMS WEP attack [44]		◇ Broadcast attack [103]
2002				◇ Non-random Z_1 [112]
2003			◇ State part-known [139]	
2004		◇ Korek WEP att. [84, 85]		
2005		◇ Mantin WEP att. [101]		◇ Mantin's $AB\mathcal{S}AB$ [102]
2006		◇ Klein WEP attack [80]		

(continued to next page)

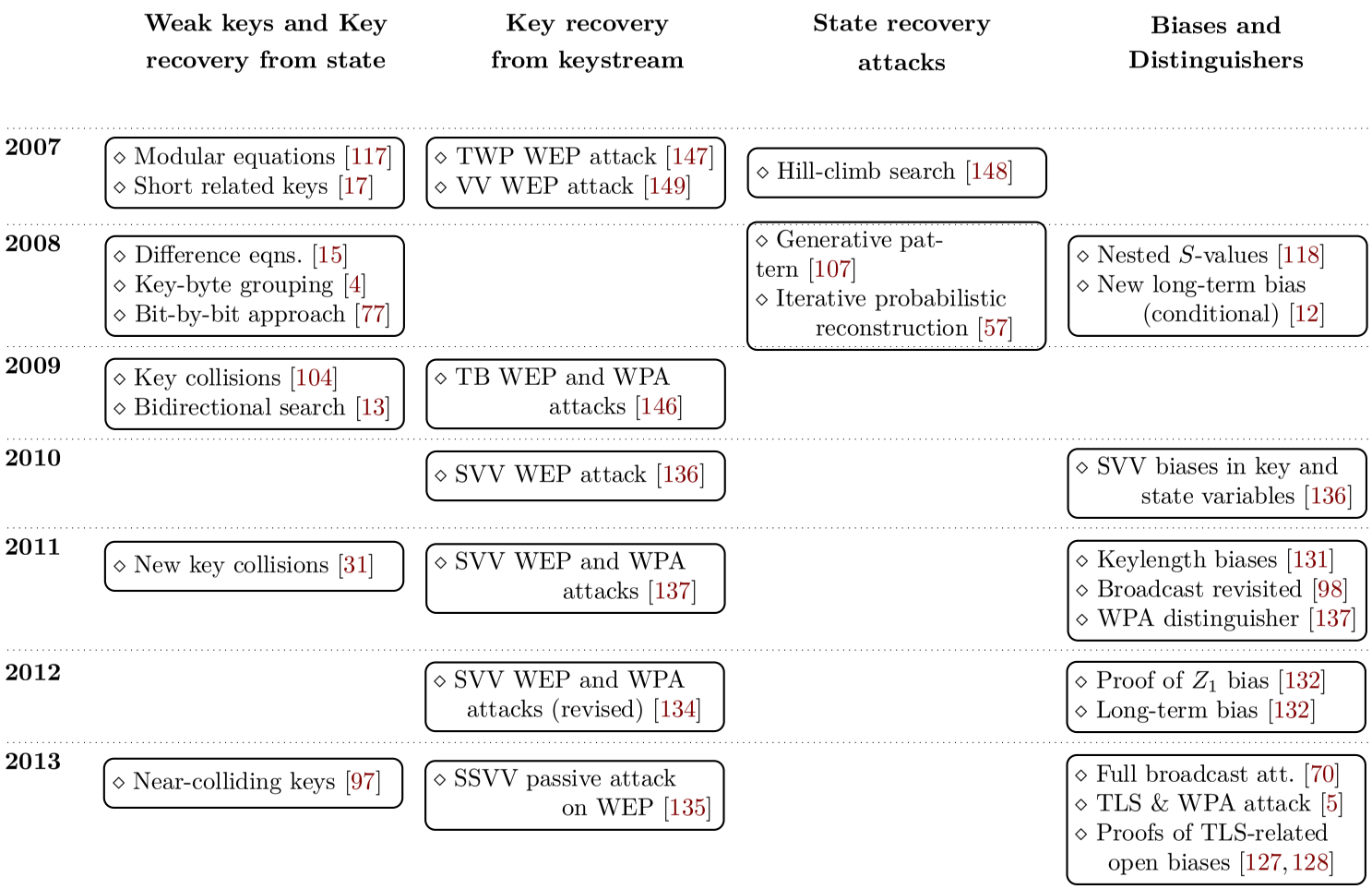

Figure 2.1: Chronological summary of RC4 cryptanalysis from 1995 to 2013.

2.1 Weak keys and Key recovery from state

Weaknesses of RC4 keys and KSA has attracted quite a lot of attention from the community. In particular, Roos [125] and Wagner [150] showed that for specific properties of a 'weak' secret key, certain undesirable biases occur in the internal state and in the keystream. Grosul and Wallach [60] demonstrated that certain related-key pairs generate similar output bytes in RC4, and later Biham and Dunkelman [17] found better related key constructions. Matsui [104] found much stronger key collisions in the cipher in 2009, and recently, there has been some more results on key collision by Chen and Miyaji [31], and partial key collision by Maitra, Paul, Sarkar, Lehmann and Meier [97]. A direct approach for key recovery from the internal permutation of RC4 was first proposed by Paul and Maitra [117], and was later studied by Biham and Carmeli [15], Khazaei and Meier [77], Akgün, Kavak and Demirci [4], and Basu, Maitra, Paul and Talukdar [13]. Brief summary of each result is as follows.

2.1.1 Weak keys

This line of analysis targets a specific set (however small) of keys in RC4 which leave certain characteristic *traces* on the internal state of the cipher after key scheduling (KSA), or on the output keystream bytes. The sets of keys or specific key-patterns with prominent discoverable traces on the state or the keystream are called *weak keys*. Once such prominent traces are observed, an adversary may attempt a key recovery from the state or the keystream in case one of the weak keys are used.

Roos weak keys [125]

In 1995, just after the public 'appearance' of RC4, Roos [125] discovered the first set of weak keys for the cipher. He discovered that, given an RC4 key $K[0], \ldots, K[l]$ with $K[0] + K[1] = 0$, there is a significant probability that the first byte generated by RC4 will be $Z_1 = K[2] + 3$. In fact, experimentally he found that the probability of occurrence of this event ranges between 0.12 and 0.16, depending on the values of $K[0], K[1]$, with a mean of approximately 0.138. This is significantly more than the chance of occurrence due to random

association, which is approximately 0.003906. These findings and related results were theoretically proved a decade later by Paul, Rathi and Maitra [121].

In an addendum to his original report [125], Roos extended this to the first two output bytes. He found that keys starting with "00 00 FD" have 0.14 probability of generating keystream which start with "00 00", and keys starting with "03 FD FC" have 0.05 probability of generating keystream which start with "FF 03". If an 80-bit keys are in use, then these discoveries result in a key recovery attack on RC4 with 50% chance of success using only $2^{67.8}$ trials, as compared to a brute force approach with 2^{79} trials[1]. For 64-bit keys, the complexity of this attack is reduced to $2^{51.8}$ from a brute force complexity of 2^{63} trials[2]. This was the first successful cryptanalytic attempt on RC4, setting the stage for a plethora of research initiatives in this direction.

Wagner weak keys [150]

Motivated by the Roos's post [125], Wagner presented some of his observations [150] in the same direction. He had independently noticed some of the results posted by Roos, and had some new ones as well. Wagner used the observation (independently presented by Roos [125]) that, given an l-byte and some $e < l$, there is around 37% probability that element e of the state table depends only on elements $K[0], \ldots, K[e]$ of the key. He was motivated by a 'simplified-exportable-RC4-SSL-variant' where an adversary can actively choose or modify the first 10 bytes $K[0], \ldots, K[9]$ of a 16-byte key and thus try to guess the remaining bytes $K[10], \ldots, K[15]$. In such a model, the first three keystream bytes will reveal $K[10], K[11], K[12]$ with considerable probability of success, and then the adversary can recover the remaining bytes of the key by brute force. This approach does reduce the attack complexity, but is not of much practical interest, as the model considered here does not match the actual SSL protocol, which uses a hash on the RC4 key before using it.

[1]Full 80-bit key recovery attack using brute force would require 2^{80} trials. However, key recovery with 50% chance of success requires $2^{80}/2 = 2^{79}$ trials in brute force attack.

[2]Key recovery with 50% chance of success requires $2^{64}/2 = 2^{63}$ trials in brute force attack.

2.1.2 Key collisions

The RC4 KSA is not one-to-one in terms of the secret key and the state of the cipher produced by KSA. Thus, it may happen that two different secret keys K_1 and K_2 generate the same state $S_N^{K_1} = S_N^{K_2} = S_0$ through KSA, and produce the same keystream output thereafter. This situation is called *key collision*, and constructing a pair of colliding keys is the goal for this attack.

Grosul and Wallach related key-pairs [60]

In 2000, Grosul and Wallach [60] came up with the first construction and example of key collisions in RC4. However, this was not a complete key collision, as they propose the construction of *related key-pairs* for which the output keystreams match in the initial hundred bytes and then diverge. The idea was to simply construct the second key of the pair by making two complimentary changes in the first key (any random key) with the goal of not disturbing the update process of the state.

Grosul and Wallach first propose a change in the last byte of the key-pair, so that a difference in only three bytes is created in the state after KSA. They also suggest complimentary differential changes $(+\delta, -\delta)$ in the last two bytes of the key-pair, so that the state after KSA is minimally affected. The authors produced practical examples of such related key-pairs for 256-byte (full length) keys, but admitted that their attack was ineffective for shorter and more practical RC4 keylengths. Recently, this idea has been studied again by Maitra, Paul, Sarkar, Lehmann and Meier [97].

Biham and Dunkelman differential key-pairs [17]

In 2007, Biham and Dunkelman [17] proposed the first constructive method to remove the problem of only 256-byte keys in Grosul and Wallach's attack [60]. They considered practical 32-byte keys for RC4, and proposed to introduce the differential changes $(+1, -1)$ in the last two bytes of the related key-pair. Thus, for the first 30 rounds of KSA, the state S^K remains identical for both the keys, and it remains identical even after 32-rounds with probability 2^{-16}. After KSA is over, the probability that the states $S_N^K = S_0$ are the same for

both keys is $(2^{-16})^8 = 2^{-128}$. Note that for a key-pair with length l, this probability is approximately $(2^{16})^{N/l}$, and thus, this related key-pair attack is not too effective for short keys. In fact, it is elusive whether a practical example of colliding keys may at all be obtained by this method.

The main idea of [17] behind obtaining a key-collision was to change the key bytes in two places such that the effect of one difference is nullified by the other during the course of KSA. However, this intuitive idea did not succeed in obtaining practical key collisions. Surprisingly, an actual key collision in RC4 was discovered by creating the difference in only one key byte, as in [104].

Matsui key collisions [104]

In 2009, Matsui [104] finally proposed a practical method of constructing colliding key-pairs of RC4. In sharp contrast with the previous works of Grosul and Wallach [60] and Biham and Dunkelman [17], Matsui made no attempts to introduce self-correcting complimentary differentials in the key-pair. Rather, the keys were selected to have difference in a single byte, so that the initial mismatch in the state is nullified at the end of KSA.

The approach of Matsui [104] considered tracking the *distance of a key-pair*, which is defined as the number of different bytes in the states at a given round of KSA. The author designed an algorithm that tracked this distance over all rounds of KSA and adaptively constructed a related key-pair such that the distance is not more than 2 in any round. This algorithm effectively generated practical key collisions, and Matsui reported an example each of colliding key-pairs of lengths 24, 43 and 64 bytes in his paper [104]. The author also reported a 20-byte near-colliding key-pair for which the states after the completion of KSA differed in only two positions.

Chen and Miyaji 22-byte key collision [31]

The strongest practical results on full-key collisions were published in 2011, by Chen and Miyaji [31]. The motivation of the authors was not to device theoretical constructions of related keys, but to experimentally find colliding key-pairs for short and practical keylengths. The main contribution of this

work is the design of an efficient search algorithm that produces a 22-byte colliding key-pair of RC4, the shortest one to date. However, finding colliding key-pairs for the most practical 16-byte key still seems illusive.

Maitra, Paul, Sarkar, Lehmann and Meier near-colliding keys [97]

In 2013, the most recent work related to key collisions was published by Maitra, Paul, Sarkar, Lehmann and Meier [97]. The authors discover and discuss methods to construct near-colliding key-pairs such that the states generated after KSA differ only by a few bytes. The best near-collision of states presented in this work [97] has 230 (out of $N = 256$) matching state bytes after KSA.

2.1.3 Key recovery from state

As RC4 PRGA is one-to-one and reversible, it is easy to reach the initial state S_0 from any given state of PRGA. However, it is not so trivial to reverse the KSA and obtain the secret key. If one can do this efficiently, it is possible to convert a state-recovery attack on RC4 to a key-recovery attack as well. This line of key recovery analysis was not considered until quite recent times.

Paul and Maitra equation solving approach [117]

In 2007, Paul and Maitra [117] pioneered this area of analysis in their work on key recovery by solving system of modular equations. The motivation for the work arises from the known correlations between key bytes and the PRGA initial state bytes, as presented by Roos [125] back in 1995. Roos' observations and similar results generate a system of modular equations relating sums of consecutive key-bytes to the individual bytes of the PRGA initial state S_0 (refer to [117, 119] for details). The trick is to choose useful equations, with known values from S_0, and solve those systematically for the key-bytes.

However, one should note that the Roos' correlations and similar results connecting the key-bytes and the state-bytes are probabilistic in nature. Thus, the correctness of the solution for the key-bytes probabilistically depends on the correctness of the chosen equations. The work of Paul and Maitra [117] does not assume that majority of the equations are correct, as one may always

cross-check the correctness of a system of equations by running the KSA again. The authors claim through empirical results that one obtains enough correct equations in a significant proportion of cases to solve for the correct key.

Biham and Carmeli difference equation approach [15]

In 2008, Biham and Carmelli [15] improved the approach of Paul and Maitra [117] to achieve more efficient key recovery with the same probability of success. The authors replaced the basic modular equations of [117] by *difference equations*, generated by taking pairwise differences of the original equations. The novelty in this idea lies in the observation that the probability of these difference equations to be correct are more than the probability of correctness of the original ones. So, solving the system of difference equations with the same success probability becomes easier compared to solving the original modular equations, thus providing a more efficient technique for key recovery.

Akgün, Kavak and Demirci key byte grouping approach [4]

In 2008, the work of Akgün, Kavak and Demirci [4] further improved the equation solving approach for key recovery from state. The authors used several different events to guess each value of the j-index, and then proceeded to solve the difference equations of Biham and Carmelli [15].

Instead of solving for each key-byte straight away, Akgün, Kavak and Demirci guessed the sum of key-bytes first, and assigned weighted guess values to a selection of key-bytes. Then the equation solving took place over groups of key-bytes, where some of the bytes are already assigned weighted guesses, and the others are obtained by solving the system of equations. This group-wise divide and conquer approach for equation solving makes the technique even more efficient, and one gets a faster key recovery with same success probability.

Khazaei and Meier bit-by-bit approach [77]

In the same year 2008, Khazaei and Meier [77] presented another improvement on the work of Biham and Dunkelman [15]. They started with the same dif-

ference equations as [15], but considered a bit-by-bit approach to key recovery. The basic idea is to convert the probabilistic difference equations into deterministic equations with noise, and recover the key-bytes in a correlation-based attack using a hypothesis testing approach. This hypothesis testing approach for key recovery can then be extended to a bit-level decision problem, which is solved bit-by-bit to obtain the key. The bit-level decision-making process is performed through a tree-based search algorithm, which claims to have better complexity compared to the previous key recovery techniques.

Basu, Maitra, Paul and Talukdar bidirectional search [13]

In 2009, Basu, Maitra, Paul and Talukdar [13] employed a bidirectional search algorithm for guessing the key-bytes, to introduce some parallelism in the search process. Instead of guessing the key-bytes contiguously, the authors simultaneously guessed the lower-index key-bytes using the lower-index S_0-bytes, and the higher-index key-bytes using the higher-index S_0-bytes. Cross-checking is done for consistency at regular intervals by running KSA with the guessed values of key-bytes, and matching known values of j. This inherently parallel key-recovery process was naturally faster, and the authors of [13] estimated the recovery of 16-byte secret keys with success probability 0.1409, which was almost twice the earlier best known value of 0.0745 reported in [4].

2.2 Key recovery from keystream

Key recovery from the keystream primarily exploits the use of RC4 in WEP and WPA. The analysis by Fluhrer, Mantin and Shamir [44] and Mantin [101] are applicable towards RC4 in WEP mode, and there are quite a few practical attacks [80, 84, 85, 135–137, 146, 149] on the WEP protocol as well. After a practical breach of WEP by Tews, Weinmann and Pyshkin [147] in 2007, the new variant WPA came into the picture. This too used RC4 as a backbone, and the most recent result published Sepehrdad, Vaudenay and Vuagnoux [137] mounts a distinguishing attack as well as a key recovery attack on RC4 in WPA mode, later improved in [134]. A brief summary of each of these attacks is as follows.

2.2.1 Attacks on WEP

WEP, the RC4-based protocol, had initially been the choice for IEEE 802.11 standard, thus making it an attractive target for cryptanalysts. For robustness against packet loss due to transmission errors, the protocol is designed to encrypt the packets independently. However, as RC4 does not inherently support an IV, the WEP protocol generates 64 to 128-bit session keys for each packet, using a 3-byte IV concatenated with a 5 to 13-byte secret key.

In practice, the IV is generated using a counter, incremented to produce slightly different keys to encrypt each packet. As a relevant portion of the plaintext in the WEP protocol is practically constant, and some other bytes are predictable, the adversary has access to a considerable number of known plaintext-ciphertext pairs. Thus the most desirable attack on WEP is to recover the secret key K from the known IV (first three bytes of K) and known values of RC4 keystream bytes obtained from the plaintext-ciphertext pairs.

Fluhrer, Mantin and Shamir related-key attack on WEP [44]

The main idea of related-key attacks on RC4 in the WEP mode was introduced by Fluhrer, Mantin and Shamir [44] in 2001. They described their attack idea in theory, and claimed that approximately 4 million packets were sufficient to recover the WEP secret key with success probability 0.5. This is based on the implementation of WEP using incremental IVs for consecutive packets. In practice, this estimate is close to 5 to 6 million packets, as later verified experimentally by Stubblefield, Ioannidis and Rubin [143, 144].

Korek practical attacks on WEP [84, 85]

In 2004, Korek's work [84, 85] initiated the next generation of WEP attacks, where practical key recovery algorithms took the center stage. In fact, there was no theoretical analysis in [84, 85], and only practical implementations of WEP attacks in the form of the Aircrack-ng software [37] were put forward. The attack complexity was reduced to about 100,000 packets for a success probability of 0.5.

Mantin attack on RC4 and WEP [101]

In 2005, Mantin [101] utilized the well-known Jenkins' correlation or the glimpse bias [75] to mount a key recovery attack on RC4 in the WEP mode. Mantin's attack improved that of Fluhrer, Mantin and Shamir [44] in terms of complexity, and remained effective even if the first $N = 256$ bytes of the keystream were discarded. No practical timings were reported for general WEP scenario, but it was claimed that 16-byte key recovery was possible in 2^{48} steps using 2^{17} short keystreams generated from different chosen IVs. The data complexity was estimated as 2^{22} if the IV was concatenated after the secret key. This work [101] also introduced the notion of fault injection in WEP attacks, and claimed that only 2^{14} faulted keystreams could be used to recover the internal state and the secret key.

Klein attack on WEP [80]

In 2006, Klein's work [80] presented a new attack strategy on WEP, which claimed to improve the complexity to 25000 packets for a 0.5 probability of success. However, these estimates were only theoretical, as no practical implementation of this attack was made in [80]. Later in 2010-11, Sepehrdad, Vaudenay and Vuagnoux [136, 137] implemented this attack and verified that the practical complexity was approximately 60000 packets for Klein's attack [80].

Tews, Weinmann and Pyshkin 60-second attack on WEP [147]

In 2007, the first hands-on practical attack on WEP was demonstrated by Tews, Weinmann and Pyshkin [147], where the attack complexity is reduced to 40000 packets, with a brute force on 10^6 most probable secret keys. No theoretical analysis was presented in this work [147], but the practical results could mount a key recovery attack on WEP in under 60 seconds.

Vaudenay and Vuagnoux attack on WEP [149]

In the same year 2007, Vaudenay and Vuagnoux [149] presented another practical work in the same direction, without any theoretical analysis. This work

could further reduce the complexity of key recovery to about 32700 packets generated using random IVs, where 10^6 most probable secret keys were brute forced. The success probability was 0.5, same as before.

Tews and Beck attack on WEP [146]

In 2009, Tews and Beck [146] reimplemented an optimized version of the Vaudenay and Vuagnoux [149] attack using the Aircrack-ng software in 'interactive mode'. The use of this mode was quite novel, and the complexity of the attack could be reduced to 24200 packets with the same number of brute forced keys and with the same probability of success. Later in 2010-11, Sepehrdad, Vaudenay and Vuagnoux [136, 137] repeated this attack using Aircrack-ng in a non-interactive mode, and could only obtain a complexity of 30000 packets.

Sepehrdad, Vaudenay and Vuagnoux attacks on WEP [134, 136, 137]

During 2010-12, Sepehrdad, Vaudenay and Vuagnoux [134, 136, 137] presented several attacks on WEP, continuously evolving and improving over the earlier results. In [136], the authors discovered several new statistical biases of RC4, and used some of them to mount a key recovery attack on WEP. Carefully chosen biases from the existing literature and the new findings led to a key recovery of WEP with only 9800 packets in less than 20 seconds time. This proved to be much faster in comparison with the previous best attack of Tews and Beck [146]. In [137], Sepehrdad, Vaudenay and Vuagnoux estimated that optimized implementations of the key-recovery attack on WEP can recover the key from only 4000 packets, if one assumes that the first bytes of the plaintexts are known for each packet. In 2012, these results on WEP key recovery were substantially improved in [134] under the name 'Tornado Attack', and the new results arising from this attack were formally presented in [135] by Sepehrdad, Susil, Vaudenay and Vuagnoux.

Sepehrdad, Susil, Vaudenay and Vuagnoux passive attack [135]

This paper by Sepehrdad, Susil, Vaudenay and Vuagnoux [135] is the most recent result on WEP analysis, reporting extremely fast and optimized active

and passive key recovery attacks against the protocol. The motivation of this work is based on the earlier results from [137], where the theoretical estimates for packet complexity was not entirely correct. The authors show in [135] that in practice, the number of packets required for the attack of [137] is considerably higher than the claimed value of 4000.

After correcting the estimates of [137], the authors report an active attack on 104-bit WEP, based on ARP injection, requiring 22500 packets for a success probability of 0.5. This attack is practically tested and verified in Aircrack-ng software, used in non-interactive mode. In the interactive mode, however, Aircrack-ng requires only 19800 packets to recover the 104-bit WEP key through this active attack. The passive attack presented in this work [135] requires only 27500 packets to recover a 104-bit WEP key. This work [135] presents the best known key recovery attacks on WEP to date.

A summary of WEP attacks, in terms of packet complexity, is presented in Table 2.1. The shaded rows in the table refer to attacks that report only theoretical estimates, and provide no practical measure for packet complexity.

Table 2.1: Summary of WEP attacks in terms of packet complexity.

Year	Reference	Type	Packets
2001	FMS theory [44]	Passive (estimate)	4,000,000
	FMS verified [143, 144]	Passive (actual)	5,500,000
2004	Korek [84, 85]	Passive (Aircrack-ng)	100,000
2006	Klein theory [80]	Passive (estimate)	25,000
	Klein verified [136, 137]	Passive (actual)	60,000
2007	TWP attack [147]	Passive (actual)	40,000
2007	VV attack [149]	Passive (actual)	32,700
2009	Tews-Beck [146]	Interactive (Aircrack-ng)	24,200
	TB verified [136, 137]	Non-Int. (Aircrack-ng)	30,000
2010	SVV attack [136]	Passive (estimate)	9,800
2011	SVV attack [137]	Passive (estimate)	4,000
2013	SSVV attack [135]	Passive (actual)	27,500
2013	SSVV attack [135]	Non-Int. (Aircrack-ng)	22,500
2013	SSVV attack [135]	Interactive (Aircrack-ng)	19,800

2.2.2 Attacks on WPA

When WEP was proved to be weak in terms of related-key attacks, the 802.11 standard adapted to the WPA protocol for confidentiality. WPA can be thought of as a wrapper on top of WEP to provide good key mixing features. WPA introduces a key hashing module in the original WEP design to defend against the Fluhrer, Mantin and Shamir attack [44].

It also includes a message integrity feature and a key management scheme to avoid key reuse in the protocol. The key mixing stage takes as input a temporal key, the transmitter address and a 48-bit IV (implemented as a counter), and outputs the main RC4 key, which is claimed to be unrelated for consecutive packets, and not repeated for 2^{48} packets with different IVs. Once this RC4 session key for an individual packet is obtained, the original WEP algorithm comes into effect to encrypt the plaintext, concatenated with the message integrity code. It is an interesting line of analysis to seek weaknesses in the WPA protocol, which is robust against most WEP vulnerabilities.

Tews and Beck data-injection [146]

In 2009, Tews and Beck [146] presented the first practical attack on the WPA-PSK protocol. However, the goal of the attack was to inject data in the encrypted channel, and it was not any form of key recovery attack as such.

Sepehrdad, Vaudenay and Vuagnoux attacks on WPA [134, 137]

During 2011-12, Sepehrdad, Vaudenay and Vuagnoux [134, 137] presented the first practical key recovery attack on WPA. The authors first described a distinguisher of complexity 2^{43} for WPA, with a packet complexity of 2^{40} and probability of success 0.5. Thereafter, based on several partial key recovery approaches, the authors presented a new attack to recover the full 128-bit temporary key of WPA by using only 2^{38} packets. The time complexity of this key-recovery attack was theoretically estimated as 2^{96} in [134, 137].

2.3 State recovery attacks

RC4 has a state-space of size $N! \times N^2$, where the $N!$ term comes from the permutation space of the N-byte S array, and N^2 is due to all possible pairs of values for i, j. In practice, for $N = 256$, the state-space becomes $256! \times 256^2 \approx 2^{1700}$, which makes a state-recovery attack extremely challenging for RC4.

The first important state recovery attack was due to Knudsen, Meier and Preneel [81]. After a series of improvements by Mister and Tavares [113], Golic [54], Shiraishi, Ohigashi and Morii [139], and Tomasevic, Bojanic and Nieto-Taladriz [148], the best attack was published by Maximov and Khovratovich [107]. A contemporary result by Golic and Morgari [57] makes some critical comments on [107], and claims to improve the attack of [107] even further by iterative probabilistic reconstruction of the RC4 internal states. Brief summary of each of these attacks is as follows.

Knudsen, Meier and Preneel branch-and-bound approach [81]

In 1998, Knudsen, Meier and Preneel [81] proposed the first effective state recovery attack on RC4 that reduced the search space considerably compared to an exhaustive search. The main idea was to solve for the four unknown variables $j_r, S_r[i_r], S_r[j_r], t_r = S_r^{-1}[Z_r]$ in each round of RC4, using an adaptive and optimized branch-and-bound method for the search. The theoretical estimate for the complexity of this attack was 2^{779} for $N = 256$, and in case some state values are known beforehand, the complexity decreases even further.

Mister and Tavares cycle structures of RC4 [113]

In the same year 1998, Mister and Tavares [113] published a similar observation, made independently of [81]. The results proposed by the authors for RC4 state recovery followed the principle of branch-and-bound search. The interesting addition to this work [113] was the tracking of possible states using certain cycle-structure of RC4. The cycle-structures were explored and used in attacking smaller versions of RC4; for example, the state of a RC4-like cipher with $N = 32$ could be recovered in 2^{42} steps. However, no practical attack was attempted on the full-scale cipher, and no complexity estimates were stated.

Golic iterative probabilistic recovery [54]

In 2000, Golic [54] presented the first sophisticated probabilistic approach towards state recovery of RC4. An iterative probabilistic algorithm was developed for reconstructing the RC4 initial state from a short segment of known keystream bytes. The reconstruction starts with the basic search approach of Knudsen, Meier and Preneel [81], but uses a much sophisticated and comprehensive probabilistic analysis, involving forward and backward recursions of the posterior probabilities. The proposed attack had similar complexity as that of [81], but required much less number of known keystream bytes.

Shiraishi, Ohigashi and Morii partial known state approach [139]

In 2003, Shiraishi, Ohigashi and Morii [139] attempted a slightly different route to state recovery, based on partially known information about the state. They first showed that the attack by Knudsen, Meier and Preneel [81] would be of search complexity 2^{220} if 112 entries of the state were known. Thereafter, they proposed an improvement of the attack in [81] to reduce the required number of known state entries to 73, keeping the search complexity the same.

Tomasevic, Bojanic and Nieto-Taladriz hill-climbing approach [148]

The work of Tomasevic, Bojanic and Nieto-Taladriz [148] in 2007 was the first to reduce the complexity of the attack by Knudsen, Meier and Preneel [81] for unconditional full-state recovery of RC4. In this work [148], the dependence relation between the unknown variables $j_r, S_r[i_r], S_r[j_r], t_r = S_r^{-1}[Z_r]$ over different rounds of RC4 was modeled as a tree, and then this tree was searched by efficient hill-climbing strategy. The final attack reduced the complexity of full RC4 state recovery to 2^{731}.

Maximov and Khovratovich generative pattern approach [107]

In 2008, the best known state recovery attack on RC4 appeared in the work of Maximov and Khovratovich [107]. They considered certain generative patterns which revealed the values of j over a window of consecutive rounds.

Once j was known for some consecutive rounds, the number of unknown variables in each round of RC4 reduced to two from the original four unknowns $j_r, S_r[i_r], S_r[j_r], t_r = S_r^{-1}[Z_r]$. This considerably simplified the problem, and the search complexity drastically reduced to 2^{241} from 2^{731} as in [148]. The estimate for time complexity was based on theoretical justifications and simulation results of the attack on reduced scale RC4.

Golic and Morgari iterative probabilistic reconstruction [57]

In the same year 2008, Golic and Morgari [57] revisited the attack of Maximov and Khovratovich [107], and presented comprehensive analysis of their work based on the techniques of iterative probabilistic reconstruction. This work [57] claimed that the complexity estimates of [107] were somewhat optimistic, and presented a practical improvement over the existing attack strategy. The improved attack is based on guess-and-determine strategy to reconstruct the RC4 state, and the authors of [57] claimed the data and time complexities to be $2^{41}, 2^{689}$ for the basic version and $2^{211}, 2^{262}$ for the optimized version of the attack, respectively. In a recent result by Golic and Morgari (unpublished; known through personal communication), a combined attack is proposed to improve the data and time complexities further to 2^{222} and 2^{214}, respectively.

Successful state recovery attacks on full-scale RC4 seem to be the most challenging line of research in RC4 analysis, as quite evident from the considerably large void in the third column of Figure 2.1. Finding short cycles in RC4 state evolution is another field in which not much progress has been made to date. The main focus of RC4 cryptanalysis to date has been towards discovery and exploitation of statistical weaknesses in the cipher, primarily in the direction of identifying new biases and corresponding distinguishers.

2.4 Biases and Distinguishers

As a stream cipher, RC4 promises to deliver pseudorandom bytes as keystream output. Thus, any lapse in that goal creates interesting consequences towards the security of the cipher. This is the reason why statistical weaknesses like

biases and their application as distinguishers have attracted the main focus of RC4 cryptanalysis to date. There have been numerous results on RC4 biases over years, and the trend still continues, as evident from Figure 2.1. The focus of RC4 analysis in this thesis is also towards biases and distinguishers.

Most of the existing results are targeted towards specific short-term (involving only the initial few bytes of the output) biases and correlations [5, 52, 70, 75, 98, 103, 112, 125, 127, 128, 132, 134, 136], while there exist only a few important results for long-term (prominent even after discarding an arbitrary number of initial bytes of the output) biases [12, 45, 52, 75, 102, 132].

We present the details of relating a bias to its corresponding distinguisher in Section 2.4.1, and a brief summary of each of these attacks thereafter.

2.4.1 Theory of biases and distinguishers

For a stream cipher, if there is an event such that the probability of occurrence of the event is different from that in case of a uniformly random sequence of bits, the event is said to be *biased*. If there exists a biased event based only on the bits /bytes of the keystream, then such an event gives rise to a *distinguisher*, an algorithm that can computationally differentiate between the keystream of the cipher and a random sequence of bits.

The efficiency of most distinguishers, including the ones we study and propose in this thesis, is judged by the number of samples (keystream or ciphertext bytes from the cipher) required to identify the bias.

Number of samples required to identify a bias

Let E be an event based on some key bits or internal state bits or keystream bits or a combination of them in a stream cipher. Suppose, $\Pr(E) = p$ for a uniformly random sequence of bits, and $\Pr(E) = p(1 + q)$ for the keystream of the stream cipher under consideration. The cryptanalytic motivation of studying a stream cipher is to distinguish these two sequences in terms of the difference in the above probabilities when $q \neq 0$. It requires the formal information theoretic notion of 'relative entropy' between two sequences.

The relative entropy between two discrete probability distributions $P(\cdot)$

and $Q(\cdot)$ is given by the Kullback-Leibler divergence [86]:

$$D_{KL}(P||Q) = \sum_x P(x) \log_2(P(x)/Q(x)),$$

where x runs over all the sample points. For the above-mentioned single event E with probabilities p and $p(1+q)$ in two different distributions $P(\cdot)$ and $Q(\cdot)$, the relative entropy is given by:

$$\begin{aligned} & p \log_2 \left(\frac{p}{p(1+q)} \right) + (1-p) \log_2 \left(\frac{1-p}{1-p(1+q)} \right) \\ & \approx -p \left(\frac{q}{1+q} \right) + (1-p) \left(\frac{pq}{1-p(1+q)} \right) \approx pq^2, \end{aligned}$$

where the approximations hold when q is small. Applying this to n samples from the same distribution, the relative entropy is obtained as npq^2, and according to [25], the bound for false positive rate (α) and false negative rate (β) in distinguishing between $P(\cdot)$ and $Q(\cdot)$ satisfy the following:

$$npq^2 \ \geq \ \beta \log_2 \frac{\beta}{1-\alpha} + (1-\beta) \log_2 \frac{1-\beta}{\alpha}.$$

Thus for a given false positive rate α and a false negative rate β, one needs roughly $O(1/pq^2)$ many samples to perform the distinguishing test. For the special case $\alpha = \beta$, this relation reduces to

$$n \geq \frac{1}{pq^2} \cdot (1-2\alpha) \cdot \log_2 \left(\frac{1-\alpha}{\alpha} \right).$$

In our context, *false positive* means that the test sequence is actually from the stream cipher but we accept it to be random, and *false negative* means that the test sequence is actually random but we accept it to be the keystream of a specific stream cipher.

In particular, $n \geq 1/pq^2$ signifies $\alpha \approx 0.22$, i.e., a success probability of approximately 0.78. Since $0.78 > 0.5$ is a reasonably good success probability, $O(1/pq^2)$ many samples are considered enough to reliably apply the distinguisher. The theoretical estimate for the complexity of a distinguisher also appears in the work of Mantin and Shamir [103], in form of the following technical result (originally Theorem 2 in [103]).

Proposition 2.1 (Mantin and Shamir [103]). *Let X, Y be distributions, and suppose that the event e happens in X with probability p and in Y with probability $p(1+q)$. Then for small p and q, $O(1/pq^2)$ samples suffice to distinguish X from Y with a constant probability of success.*

This result, when applied to the distribution of the keystream of a stream cipher and a random sequence of bytes, gives an estimate of the number of samples needed to confirm a bias (either through simulation or from practical data). We shall use this notion of *sample complexity* while judging the effectiveness of any bias discussed throughout this thesis.

It is worth noting at this point that Proposition 2.1 is true only for small values of p and q, and the result does not hold if the base event occurs with a large probability p. However, a special case of interest in stream ciphers is the case of binary events, i.e., events with two outcomes, where $p = 0.5$ is large. This special case had previously been considered by Golic [48, 52] towards the investigation for linear statistical weaknesses in stream ciphers.

2.4.2 Biases related to the secret key

As a matter of great interest to RC4 cryptanalysts was to recover the secret key from the output keystream or the internal state, it was also of prime importance to identify any biases or correlations relating the secret key to the others. There is a large number of examples of such biases in the literature.

Roos' key correlations [125]

In 1995, Roos [125] was the first to correlate the secret key bytes to the bytes of the PRGA initial state S_0. He experimentally observed that $S_0[E]$ for any given $E = 0, 1, \ldots, N-1$ depends only on the key bytes $K[0], \ldots, K[E]$ with a higher than expected probability of 0.37. More precisely, Roos postulated that $S_0[E]$ is most likely equal to be $K[0] + K[1] + \cdots + K[E] + E(E+1)/2$, just after the completion of KSA. He also stated in [125] that given an RC4 key K with $K[0] + K[1] = 0$, there is a significant probability that $Z_1 = K[2] + 3$. This experimental observation was the first bias found in the cipher [125], and the result was later proved by Paul, Rathi and Maitra [121].

Sen Gupta, Maitra, Paul and Sarkar keylength bias [131, 132]

In 2011, we identified [131] a relation between the length of the secret key to certain biases in the RC4 keystream, while trying to find theoretical justification of an observation made by Sepehrdad, Vaudenay and Vuagnoux [136]. This was the first proof for any keylength dependent bias in RC4, and resulted in a keylength distinguisher for the cipher. This distinguisher, in turn, provided an efficient strategy to guess the length of the secret key from the keystream output. The result was further studied by us in [132], and the details are presented in Chapter 3 of this thesis.

Isobe, Ohigashi, Watanabe and Morii extended keylength bias [70]

Following the identification of the first keylength dependent bias in our previous works [131, 132], Isobe, Ohigashi, Watanabe and Morii [70] extended the result in 2013 to identify a general class of keylength dependent biases in RC4. They used the biases (along with other short-term biases of RC4) to mount an efficient full plaintext recovery attack on the broadcast version of the cipher that uses a 16-byte secret key. Although the authors attempted a partial proof of this result [70], a conclusive proof of the extended keylength dependent biases was presented in our recent work [128]. We discuss the general results of [70] and [128] in Chapter 3 of this thesis.

Sarkar, Sen Gupta, Paul and Maitra keylength dependent bias of first byte and anomalies [128]

In 2013, we attempted [128] the proofs of all significant biases used by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5] towards an attack on TLS. While proving certain negative bias in $Z_1 = 129$ (for $N = 256$) observed in [5], we noticed in a recent work [128] that the negative bias of Z_1 towards 129 occurs prominently only for certain lengths of the secret key of RC4. This is the first observation of any keylength dependent bias in the first byte of the RC4 keystream.

In the same work [128], we also discovered that this bias in Z_1 may be related to the long-standing mysterious problem of "anomalies" in the distri-

bution of the state array after the RC4 KSA. In this connection, we could prove the anomaly in $S_0[128] = 127$, a problem open for more than a decade since the observation of anomaly pairs by Mantin [100] in 2001. Detailed results are presented in Chapter 3 of this thesis.

2.4.3 Biases related to state variables

Statistical weaknesses that relate the output keystream bytes of RC4 to its internal state are also of prime importance due to their direct application towards state-recovery attacks on the cipher. The main results are as follows.

Jenkins correlation and glimpse [75]

In 1996, Jenkins [75] first noticed a relation between the output keystream bytes and the internal state of RC4. He observed that at any round r of RC4, the relations $S_r[j_r] = i_r - Z_r$ and $S_r[i_r] = j_r - Z_r$ hold with probability $2/N$, twice that of the chance of random association. However, he never proved the result. In 2001, the work of Mantin and Shamir [103] brought the Jenkins' bias in the public literature, and Mantin's thesis [100] provided the first detailed explanation for this bias. This was the first bias of RC4 to provide a *glimpse* into the hidden state from the output.

Mantin non-uniform distribution of initial PRGA state [100]

In 2001, Mantin [100] identified and proved the interesting probability distribution of $S_0[u] = v$ for the initial PRGA state, where u, v ranged over all values $0, \ldots, N-1$. This results showed for the first time that the internal state S_0 right after the completion of KSA is not even close to uniform, as one would expect from a good key scheduling process. The result was proved again in 2007 by Paul, Maitra and Srivastava [120] by an alternative technique. To date, Mantin's non-uniform S_0 distribution [100] has been the most useful internal state bias of RC4, giving rise to several non-uniformities in the keystream.

Sepehrdad, Vaudenay and Vuagnoux spectral search [134, 136]

In 2010, Sepehrdad, Vaudenay and Vuagnoux [136] performed an almost exhaustive search on the space of all relations between $i_r, j_r, S_r[i_r], S_r[j_r], Z_r$ to identify all linear correlations in a single round of the cipher. Their spectral technique proved quite effective in enlisting a host of new biases in RC4 that relate the internal state to the output bytes of the cipher. The work was further discussed and extended in [134]. The authors did not prove many of the prominent biases they found, as their goal was to use these biases to mount practical attacks on WEP and WPA [134, 136, 137].

Sen Gupta, Maitra, Paul and Sarkar state-related biases [131]

In 2011, we attempted [131] a complete theoretical justification for all prominent biases identified by Sepehrdad, Vaudenay and Vuagnoux [136], and proved almost all of them. This systematic approach towards analyzing the state-related biases from [136] did not only provide proper theoretical base for the existing empirical biases, but also identified some new ones. During the same time, some other state-related results were presented in our work [98], and all the aforesaid results were further studied by us in [132]. The details are presented in Chapter 4 of this thesis.

2.4.4 Short-term biases in the keystream bytes

There exist many traits of non-random behavior in the initial keystream bytes of RC4, especially in the first N output bytes. Some of these result from the non-uniform distribution of S_0, the initial PRGA state, as observed by Mantin [100]. Some others result out of the simple permutation update and byte extraction procedure of the cipher. There has been a few attempts at analyzing the general shuffle-exchange paradigm of RC4 and its drawbacks [112, 123], while most of the effort has been targeted towards identifying the initial keystream biases and their potential application towards practical attacks. Most of these initial keystream biases are short-term, in the sense that they do not linger in future rounds of the PRGA, while some persist (often periodically) in the later rounds as well. In this section, we deal with only the

short-term biases of RC4, and summarize the prominent results as follows.

Mantin and Shamir second byte bias [103]

In 2001, Mantin and Shamir [103] identified one of the most celebrated keystream biases of RC4 – bias of the second keystream byte towards zero. They proved that the event $Z_2 = 0$ occurs with probability $2/N$, twice that of the chance of random association. Observation of such a simple yet elegant bias in the initial keystream byte of RC4 pioneered a completely new direction of analysis. This bias of the second byte [103] provided a quite efficient distinguisher of RC4, with a complexity of only $O(N)$.

In addition to this, the second byte bias allowed Mantin and Shamir to mount a practical attack on RC4 in the broadcast setting, where the same plaintext is broadcast to several receivers using different random keys for encryption. In such a case, the second byte of the plaintext can be successfully recovered from the knowledge of only about $\Omega(N)$ ciphertexts [103].

Mironov first byte sinusoidal distribution [112]

In 2002, Mironov [112] thoroughly studied the shuffle-exchange paradigm of RC4, took a close look at the first keystream byte of RC4, and identified a negative bias towards zero. This bias was of the order of $1/N^2$, not as prominent as the Mantin and Shamir [103] bias, but nonetheless, was an interesting observation. He also noticed that the probability distribution of $Z_1 = v$ for $v = 0, \ldots, N-1$ exhibits a non-uniform sinusoidal feature. This was only an experimental observation, without any proofs attempted [112].

Maitra, Paul and Sen Gupta short-term biases towards zero [98]

Mantin and Shamir's [103] claimed in 2001 that no significant bias towards zero exists in the initial keystream bytes of RC4 other than the one exhibited by Z_2. However in 2011, after a decade of this claim, we refuted [98] it by proving that all initial bytes of RC4, from Z_3 to Z_{255}, exhibit considerably significant biases, of the order of $1/N^2$, towards zero. We could successfully identify and prove [98] the biases in $Z_r = 0$ for $r = 3, \ldots, 255$. A detailed account of this

result is presented in Chapter 5 of this thesis. These biases were later used by Isobe, Ohigashi, Watanabe and Morii [70] towards a general attack on the broadcast mode of RC4.

Sen Gupta, Maitra, Paul and Sarkar proof of first byte bias [132]

In 2013, after almost a decade of the identification of Mironov [112] first byte bias, we theoretically proved [132] the complete distribution of Z_1 for full keylength $l = N = 256$, as well as its negative bias towards zero. The details are presented in Chapter 5 of this thesis.

Sarkar second byte negative bias [127]

In 2013, another short-term bias was identified and proved by Sarkar [127], which showed that the second byte Z_2 is negatively biased towards the value 2. This in fact is the most prominent bias in Z_2 after the $Z_2 = 0$ positive bias proved by Mantin and Shamir [103]. Sarkar [127] also mention the existence of a negative bias in $Z_2 = N/2 + 1$ and positive biases in $Z_r = r$ for $r = 3, \ldots, N - 1$, but no proofs were presented in this work.

Isobe, Ohigashi, Watanabe and Morii full broadcast attack [70]

In 2013, as a follow up of [98, 132], Isobe, Ohigashi, Watanabe and Morii [70] attempted an experimental identification of the best biases in each initial keystream byte $Z_1, Z_2, \ldots, Z_{257}$ of RC4. They could discover and prove some new short-term biases in RC4 [70] in addition to rediscovering the known biases [98, 103, 112, 132]. Two major new biases proved in [70] are in the events $(Z_1 = 0 \mid Z_2 = 0)$ and $Z_3 = 131$. However, the most important short-term bias identified and proved in this work was the general positive bias pattern in $Z_r = r$ for $r = 3, \ldots, N - 1$, which was independently observed, but not proved, by Sarkar [127].

The prime target of this work [70] was RC4 in broadcast mode. The authors could mount the first practical full-plaintext recovery attack on broadcast RC4 by combining all known and new biases, specifically the best ones for each initial keystream byte of the cipher, with the long-term digraph repetition bias

of Mantin [102]. Isobe, Ohigashi, Watanabe and Morii [70] prove that almost all of the first 257 bytes of the plaintext can be recovered, with probability more than 0.8, using only 2^{32} ciphertexts of broadcast RC4. If the digraph repetition bias [102] is used, then all later bytes of the plaintext can also be recovered from about 2^{34} ciphertexts. They validate their theory by providing experimental attacks on RC4 in broadcast setting, where they estimate the recovery of first 2^{50} bytes of the plaintext, with probability close to 1, using only 2^{34} ciphertexts generated by random keys. Some critical comments on these results have been made in a related recent work [5].

AlFardan, Bernstein, Paterson, Poettering, Schuldt TLS-attack [5]

In the same year 2013, another prominent attempt was made by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5, 14] to experimentally identify all initial keystream biases in RC4. They ran extensive experiments, using more than 2^{44} random keys, to generate a list of approximately 65536 single-byte short-term biases of RC4, including the previous ones [70,98,103,112,132]. This search provides a comprehensive list of non-random behavior of the initial keystream bytes (bytes 1 to $N = 256$) of RC4 when a 16-byte key is used.

The main goal of this analysis [5] was not to provide theoretical justification for the biases, but to exploit those in a practical message recovery attack against the TLS protocol that uses RC4 for confidentiality. The authors could use all of the above-mentioned 65536 initial short-term biases of RC4 to mount a plaintext recovery attack on the TLS protocol that reliably recovers the first 256 bytes of the plaintext from the knowledge of only 2^{28} to 2^{32} ciphertexts generated using random keys, with no prior plaintext knowledge. The authors show that plaintext recovery for RC4 in TLS is also possible from arbitrary positions of the plaintext, provided that enough (around 2^{34}) encryptions of the same plaintext bytes are available.

The same principle was exploited in the full version of the research paper [5] to mount a plaintext-recovery attack against WPA. The authors claim that the first 130 bytes of the plaintext can be reliably recovered with the knowledge of about 2^{30} frames of the WPA/TKIP protocol. This attack on TLS and WPA by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5] is one of

the most extensive attacks on any practical RC4 protocol to date, with far-reaching consequences.

Sarkar, Sen Gupta, Paul and Maitra proofs of TLS biases [128]

In 2013, immediately after the practical breach of RC4 security in TLS by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5, 14], the proofs of almost all open (unproved or partially proved in literature) and significant TLS-related biases were presented by us in a recent work [128]. We provided proofs for the open biases in $Z_2 = 129$, $Z_2 = 172$, $Z_4 = 2$, $Z_{256} = 0$, $Z_{257} = 0$, and the detailed results are presented in Chapter 5 of this thesis.

RC4 short-term landscape generated from the data of [5, 14]

The extensive experimental results by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5] identified several non-random behaviors in the short-term output keystream of RC4. Figure 2.2 presents a 3D model of the probabilities $\Pr(Z_r = v)$ for $r = 1, \ldots, N$ and $v = 0, \ldots, N-1$, which we call the 'RC4 landscape' of initial keystream bytes.

Note that this landscape is for the most practical version of RC4 that uses a 16-byte key, and is not identical for RC4 initial keystream patterns generated by secret keys of various other lengths. For example, the non-random peaks and troughs present in the 16-byte key landscape reduce to a certain extent if one uses a full length $N = 256$ bytes key.

The visible vertical walls and spikes in Figure 2.2 identify the prominent short-term bias patterns in the RC4 landscape. The main ones are for the events $Z_2 = 0$ (largest positive spike), $Z_2 = 2$ (largest negative spike), $Z_1 = v$ where $v = 0, \ldots, N-1$ (sinusoidal vertical wall on the left side), $Z_r = 0$ (decreasing vertical wall on the right side), $Z_r = r$ (decreasing vertical wall at the center) and $Z_r = -r$ (decreasing series of spikes at the center), where $r = 1, \ldots, N$. The patterns for $Z_1 = v$, $Z_r = 0$, $Z_r = r$ and $Z_r = -r$ are presented independently in Figure 2.3.

The proofs for many of these major non-random events are present in the literature, as discussed earlier. The biases in $Z_2 = 0$ and $Z_2 = 2$ have been

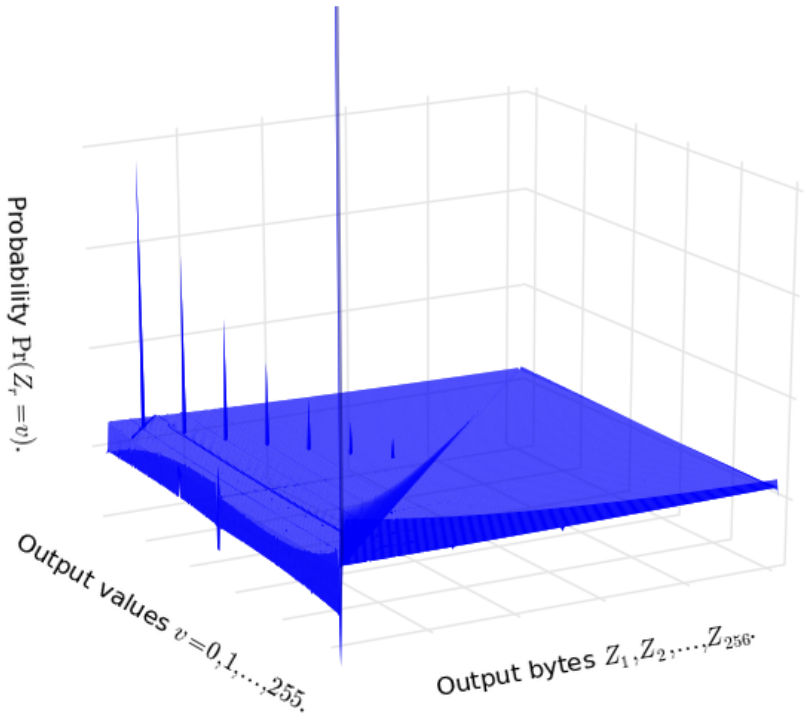

Figure 2.2: The RC4 landscape of initial keystream bytes (data from [5, 14]).

proved by Mantin and Shamir [103] in 2001 and Sarkar [127] in 2013 respectively. The sinusoidal pattern of Z_1 for full-length key, including the negative biases in $Z_1 = 0, 1$, have been proved by us [132] in 2013, and the general proof for $Z_r = 0$ had been presented by us [98] in 2011. We also proved [131] the $Z_r = -r$ case for $r = 16$ (keylength) in 2011; and we proved the general pattern for $Z_r = -r$ in a recent work [128]. In 2013, Isobe, Ohigashi, Watanabe and Morii [70] proved the general bias pattern for $Z_r = r$.

The slightly weaker single-byte bias $Z_3 = 131$ has been proved by Isobe, Ohigashi, Watanabe and Morii [70]; and the biases in $Z_2 = 129$ and $Z_2 = 172$ have been proved by us [128] in 2013. These biases are pictorially depicted as visible spikes in Figure 2.4.

In addition to these, we have also proved the biases in $Z_4 = 2$, $Z_{256} = 0$ and $Z_{257} = 0$ in our recent work [128]. A consolidated account of the current state-of-the-art scenario, in terms of identified and proved short-term keystream biases, is presented in Table 2.2 at the end of this chapter.

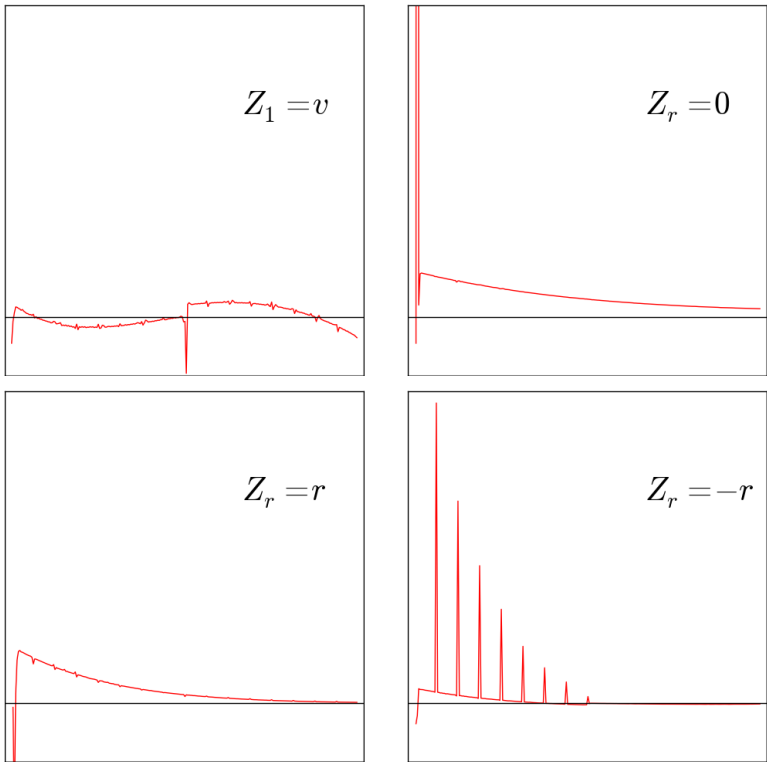

Figure 2.3: Prominent bias patterns in RC4 initial bytes (data from [5, 14]).

2.4.5 Long-term biases in the keystream bytes

In view of the numerous short-term biases infecting the initial keystream bytes of RC4, many researchers have suggested to discard the initial N to $6N$ bytes of the keystream, and use the ones thereafter for encrypting the plaintext. If this practice is adopted, then an attacker will be interested to find long-term biases of RC4, that is, biases which remain in the keystream even after an arbitrary number of initial bytes are discarded. There has been a few prominent results in this direction, summarized as follows.

Golic bit-wise bias [52]

In 1997, Golic [52] was the first to systematically study the linear statistical weaknesses in the RC4 keystream, and to propose a long-term bias of the cipher. He identified that the second binary derivative of the LSB output sequence is correlated to 1 with an approximate correlation coefficient of $15/N^3$. It means that the LSB of $Z_r \oplus Z_{r+2}$ for all $r \geq 1$ is biased towards 1, or in other

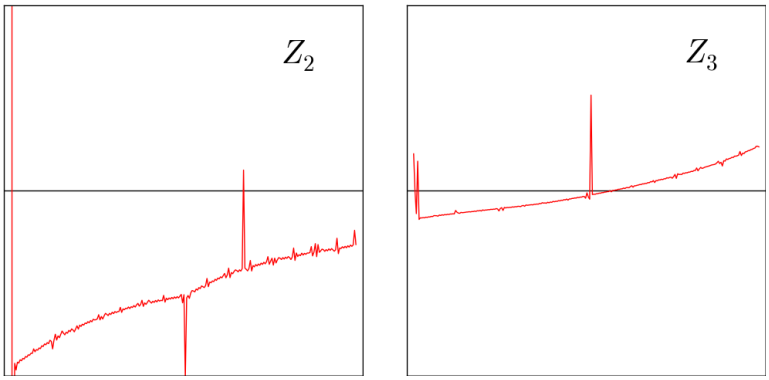

Figure 2.4: Single-byte biases in Z_2 and Z_3 of RC4 (data from [5, 14]).

words, the LSBs of the non-consecutive bytes Z_r and Z_{r+2} mismatch with a probability more than that of random coincidence. This bias resulted in the first long-term distinguisher of RC4, with a theoretical complexity to $2^{40.2}$.

Fluhrer and McGrew digraph bias [45]

In 2000, Fluhrer and McGrew [45] analyzed the work of Golic [52] to show that the practical complexity estimates for the distinguisher in [52] is close to $2^{44.7}$. The authors also introduced the notion of biases in successive pairs of bytes, called digraphs, in their work [45], and performed a comprehensive analysis of the digraph probabilities in the long-term behavior of RC4 keystream. They found a list of biased digraphs which persist in the long run, with probabilities different from the chance of random association. Using these digraph biases, Fluhrer and McGrew [45] could mount an efficient distinguishing attack on RC4 that required only $2^{30.6}$ keystream bytes.

Mantin digraph repetition bias [102]

In 2005, Mantin [102] presented the best known long-term bias and distinguisher for RC4, which remains effective even after arbitrary number of initial keystream bytes are discarded. This work got motivated by the digraph occurrence analysis of [45], and attempted the analysis of the digraph repetition pattern in RC4 keystream. Mantin [102] showed that the occurrence of a repeated digraph string $AB\mathcal{S}AB$, where A, B are bytes and $\mathcal{S}$ is a random string of small length, is more frequent in RC4 keystream compared to a uniformly

random stream of bytes.

The bias in digraph repetition for RC4 is more than $1/N^2$, and moreover, the digraph biases for strings $\mathcal{S}$ of different lengths could be combined to produce a stronger cumulative bias. This cumulative bias results in the strongest known long-term distinguisher of RC4, with a complexity of just 2^{26} ciphertext bytes. This long-term bias of Mantin [102] has found one of its most prominent applications in the recent work of Isobe, Ohigashi, Watanabe and Morii [70], where a full-plaintext recovery attack has been mounted on RC4 broadcast scheme exploiting the $AB\mathcal{S}AB$ bias.

Basu, Ganguly, Maitra and Paul conditional bias [12]

In 2008, Basu, Ganguly, Maitra and Paul [12] presented a complete characterization for one-step operation of the RC4 PRGA, and identified that under certain conditions, two consecutive bytes Z_r and Z_{r+1} of the RC4 keystream are equal with probability more than that of random association. This result produces a new long-term (conditional) distinguisher of RC4 with complexity estimates of approximately 2^{40} ciphertext bytes [12].

Sen Gupta, Maitra, Paul and Sarkar periodic bias [132]

In 2013, we identified and proved [132] the first byte-wise long-term correlation between non-consecutive bytes of the RC4 keystream, which persists even after discarding an arbitrary number of initial keystream bytes. While Golic [52] proved a bit-wise correlation between non-consecutive bytes Z_r and Z_{r+2}, our work [132] identified a correlation between the non-consecutive periodic bytes Z_{wN} and Z_{wN+2} of RC4, where N is the size of the permutation. We proved a bias in the event $(Z_{wN+2} = 0 \wedge Z_{wN} = 0)$, which in turn resulted in a new long-term distinguisher for RC4. The detailed results of [132] are presented in Chapter 5 of this thesis.

2.4.6 Significance of proofs for RC4 biases

Identification of biases and distinguishers has been the most prominent trend of RC4 cryptanalysis, and proofs of the observed biases play a crucial role.

Fundamentally, it is necessary to prove and study biases in RC4 due to the undeniable significance of theoretical justification. In cryptanalysis, one uses the biases to mount practical attacks on a cipher. However, as in any scientific discipline based on the foundations of mathematics and computer science, the cryptanalytic attempts based on biases require proper theoretical support to validate the idea behind the attack. Theory helps to generalize the experimental observations and often identifies further applications of the biases to understand the intrinsic characteristics of the cipher.

Discovering RC4 biases through experiments and proofs

In general, there are two ways to discover a bias in the RC4 keystream – through experimentation, and via a theoretical approach. One has to run extensive experiments to identify significant biases in the keystream bytes of RC4. There is a strong history of this approach, including prominent recent examples like Sepehrdad, Vaudenay and Vuagnoux spectral search [134, 136], Isobe, Ohigashi, Watanabe and Morii search [70] for optimal bias set, and AlFardan, Bernstein, Paterson, Poettering and Schuldt search [5] for all single-byte initial keystream biases. This method has proved to be successful in discovering several unknown biases in the initial keystream bytes.

However, experimental method for discovering biases is restricted by computational limits of the experiment platform. For example, it is extremely improbable (if not impossible) to exhaustively enumerate all significant biases in the initial keystream bytes of RC4 (with $N = 256$) where dependence between two or more bytes are considered. In such cases, proving the biases theoretically may help, as in the process of proving the existing biases, one needs to consider some additional events and often ends up discovering new biases. This leads to further insight into the cipher. In the course of this thesis, we shall discuss several biases that would possibly not have been discovered otherwise. These include the keylength dependent biases identified and proved by us [131, 132] while analyzing certain empirical biases of [136], and the keylength dependent bias in the first keystream byte and its relation with anomalies, identified by us [128] while analyzing certain biases of [5, 14].

Identified and/or proved keystream biases of RC4

In this thesis, we try to present theoretical justification for experimentally observed and unproved significant non-randomnesses in RC4.

A consolidated account of the current state-of-the-art in terms of identified and/or proved keystream biases is presented in Table 2.2, with references to our proofs in this thesis. The shaded row represents the only open (identified but unproved) significant bias in the keystream bytes of RC4.

Table 2.2: Identified and/or proved keystream biases of RC4.

Biased event	Type of bias	Observed	Proved	In thesis
	Isolated short-term biases			
$Z_1 = 0$	Negative	[112]	[132]	Chap. 5
$Z_1 = 1$	Negative	[132]	[132]	Chap. 5
$Z_1 = 129$	Negative if l divides 128	[5, 128]	Open	–
$Z_2 = 0$	Positive	[103]	[103]	–
$Z_2 = 2$	Negative	[5, 127]	[127]	–
$Z_2 = 129$	Negative	[5, 127]	[128]	Chap. 5
$Z_2 = 172$	Positive	[5]	[128]	Chap. 5
$Z_3 = 131$	Positive	[5, 70]	[70]	–
$Z_4 = 2$	Positive	[5]	[128]	Chap. 5
$Z_{256} = 0$	Negative	[5, 70]	[128]	Chap. 5
$Z_{257} = 0$	Positive	[70]	[128]	Chap. 5
	Patterns of short-term biases			
$Z_1 = v$	Sinusoidal ($v = 0, \ldots, 255$)	[112]	[132]	Chap. 5
$Z_r = 0$	Positive ($r = 3, \ldots, N-1$)	[98]	[98]	Chap. 5
$Z_r = r$	Positive ($r = 3, \ldots, N-1$)	[5, 70]	[70]	–
$Z_l = -l$	Positive	[131]	[132]	Chap. 3
$Z_{xl} = -xl$	Positive if $x = 1, \ldots, N/l$	[70]	[128]	Chap. 3
	Long-term keystream biases			
$Z_r \oplus Z_{r+2}$	Bitwise (LSB) correlation	[52]	[52]	–
$AB\mathcal{S}AB$	Digraph repetition	[102]	[102]	–
To be filled	Digraph biases	[45]	[45]	–
$Z_r = Z_{r+1}$	Conditioned on $2Z_r = i_r$	[12]	[12]	–
$Z_{wN} = Z_{wN+2}$	Conditioned on $Z_{wN} = 0$	[132]	[132]	Chap. 5

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Biases Based on RC4 Keylength

In this chapter, we present a family of biases in RC4 that are dependent on the length of the secret key, and thereby try to predict the keylength of RC4. This chapter deals with the following problems in RC4 analysis, as mentioned earlier in Section 1.4 of Chapter 1, towards the organization of this thesis.

Problem 1a. Some of the empirical biases observed by Sepehrdad, Vaudenay and Vuagnoux [136] seem to be related to the length of the secret key used in RC4. If this is the case, is it possible to identify and prove such general relations between the keylength to the keystream biases in RC4?

Problem 1b. Investigate the negative bias of Z_1 towards 129 ($N = 256$), as observed by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5], and identify its keylength dependence characteristics.

Problem 1a is studied and solved in our papers [128, 131], and the study of Problem 1b is presented in our recent work [128].

Notation: Throughout this chapter (and the thesis), we denote the length of the secret key k as l, and in connection with the expanded key K, we define:

$$f_y \equiv \sum_{i=0}^{y} K[i] + \frac{y(y+1)}{2} \pmod{N} \quad \text{for } 0 \leq y \leq N-1.$$

Further, '$x : A \xrightarrow{\alpha} B \xrightarrow{\beta} \cdots$' denotes that the value x transits from position A to B (of S) with probability α, from B to the next with probability β, etc.

Probabilistic model: Throughout this chapter, we shall assume the probabilistic model of uniform random keys to prove the results on keylength dependent biases in KSA and initial rounds of PRGA. We assume actual RC4 next-state-function for the evolution of S and i, j, and no randomness assumptions are made on the initial state S_0 of PRGA.

3.1 Keylength dependent biases

Our motivation arises from the conditional bias $\Pr(S_{16}[j_{16}] = 0 \mid Z_{16} = -16) \approx 0.038488$ observed by Sepehrdad, Vaudenay and Vuagnoux [136]. They also mentioned in [136, Section 3] that no explanation for this bias could be found. For direct exploitation in WEP and WPA attacks, a related KSA version of this bias (of the same order) was reported in [134, Section 6.1] for the event $(S_{17}^K[16] = 0 \mid Z_{16} = -16)$.

While exploring these conditional biases in RC4 PRGA, we ran extensive experiments (1 billion runs of RC4 with randomly chosen keys in each case) with $N = 256$ and keylength $5 \leq l \leq 32$. We could observe that the biases actually correspond to the keylength l:

$$\begin{aligned} \Pr\left(S_l[j_l] = 0 \mid Z_l = -l\right) &\approx \eta_l^{(1A)}/256, \\ \Pr\left(S_{l+1}^K[l] = 0 \mid Z_l = -l\right) &\approx \eta_l^{(1B)}/256, \end{aligned} \tag{3.1}$$

where each of $\eta_l^{(1A)}$ and $\eta_l^{(1B)}$ decreases from 12 to 7 (approx.) as l increases from 5 to 32. In this section, we present first proofs of these two biases.

We also observe and prove a *family of new conditional biases*. Experimenting with 1 billion runs of RC4 in each case, we observed that:

$$\begin{aligned} \Pr\left(Z_l = -l \mid S_l[j_l] = 0\right) &\approx \eta_l^{(2)}/256, \\ \Pr\left(S_l[l] = -l \mid S_l[j_l] = 0\right) &\approx \eta_l^{(3)}/256, \\ \Pr\left(t_l = -l \mid S_l[j_l] = 0\right) &\approx \eta_l^{(4)}/256, \\ \Pr\left(S_l[j_l] = 0 \mid t_l = -l\right) &\approx \eta_l^{(5)}/256, \end{aligned} \tag{3.2}$$

where $\eta_l^{(2)}$ decreases from 12 to 7 (approx.), each of $\eta_l^{(3)}$ and $\eta_l^{(4)}$ decreases from

34 to 22 (approx.), and $\eta_l^{(5)}$ decreases from 30 to 20 (approx.) as l increases from 5 to 32. Note that as per our earlier notation, t_l denotes the index of extracting the keystream byte Z_l from the S permutation, i.e., $S_l[t_l] = Z_l$.

We also find a *keylength distinguisher* for RC4, based on the following event.

$$(Z_l = -l) \quad \text{for } 5 \leq l \leq 32. \tag{3.3}$$

3.1.1 Technical results required to prove the biases

For the proofs of the biases in this section we need some additional technical results that we present here. Some of these results would also be referred for our results in subsequent sections. We start with [100, Theorem 6.2.1], restated as Proposition 3.1 below.

Proposition 3.1. *At the end of RC4 KSA, for* $0 \leq u \leq N-1$, $0 \leq v \leq N-1$,

$$\Pr(S_0[u] = v) = \begin{cases} \frac{1}{N}\left(\left(\frac{N-1}{N}\right)^v + \left(1 - \left(\frac{N-1}{N}\right)^v\right)\left(\frac{N-1}{N}\right)^{N-u-1}\right), & \text{if } v \leq u; \\ \frac{1}{N}\left(\left(\frac{N-1}{N}\right)^{N-u-1} + \left(\frac{N-1}{N}\right)^v\right), & \text{if } v > u. \end{cases}$$

Now, we extend the above result to the end of the first round of the PRGA. Since the KSA ends with $i^K = N-1$ and the PRGA begins with $i = 1$, skipping the index 0 of RC4 permutation, this extension is non-trivial, as would be clear from the proof of Lemma 3.2. This is a revised version of [132, Lemma 1].

Lemma 3.2. *After the first round of RC4 PRGA, the probability* $\Pr(S_1[u] = v)$ *is given by*

$$\Pr(S_1[u] = v)$$
$$= \begin{cases} \Pr(S_0[1] = 1) + \sum_{X \neq 1} \Pr(S_0[1] = X \ \wedge \ S_0[X] = 1), & u = 1, v = 1; \\ \sum_{X \neq 1, v} \Pr(S_0[1] = X \ \wedge \ S_0[X] = v), & u = 1, v \neq 1; \\ \Pr(S_0[1] = u) + \sum_{X \neq u} \Pr(S_0[1] = X \ \wedge \ S_0[u] = u), & u \neq 1, v = u; \\ \sum_{X \neq u, v} \Pr(S_0[1] = X \ \wedge \ S_0[u] = v), & u \neq 1, v \neq u. \end{cases}$$

Proof. First, let us represent the probability as

$$\Pr(S_1[u] = v) = \sum_{X=0}^{N-1} \Pr(S_0[1] = X \ \wedge \ S_1[u] = v).$$

The goal is to reduce all probabilities in terms of expressions over S_0. After the first round of RC4 PRGA, all positions of S_0, except for $i_1 = 1$ and $j_1 = S_0[1] = X$, remain fixed in S_1. So, we need to be careful about the cases where $X = 1, u, v$. Let us separate these cases and write

$$\begin{aligned}
&\Pr(S_1[u] = v) \\
&= \Pr(S_0[1] = 1 \ \wedge \ S_1[u] = v) + \Pr(S_0[1] = u \ \wedge \ S_1[u] = v) \\
&+ \Pr(S_0[1] = v \ \wedge \ S_1[u] = v) + \sum_{X \neq 1,u,v} \Pr(S_0[1] = X \ \wedge \ S_1[u] = v). \qquad (3.4)
\end{aligned}$$

Now, depending on the values of u, v, we get a few special cases. In the first PRGA round,

$$S_1[u] = \begin{cases} S_1[i_1] = S_0[j_1] = S_0[S_0[1]], & u = i_1 = 1; \\ S_1[j_1] = S_0[i_1] = S_0[1] = u, & u = j_1 = S_0[1]; \\ S_0[u], & \text{otherwise.} \end{cases}$$

This indicates that one needs to consider two special cases, $u = 1$ and $u = v$, separately. However, there is an overlap within these two cases at the point $(u = 1, v = 1)$, which in turn, should be considered on its own. In total, we have fours cases to consider for Equation (3.4), as shown in Figure 3.1.

Common point $u = 1, v = 1$: In this case, $S_0[1] = X = 1$ implies no swap, resulting in $S_1[u] = S_1[1] = S_0[1]$. If $X \neq 1$, we have $S_1[u] = S_1[1] = S_0[X]$. Thus, Equation (3.4) reduces to

$$\begin{aligned}
&\Pr(S_1[1] = 1) \\
&= \Pr(S_0[1] = 1 \ \wedge \ S_0[1] = 1) + \sum_{X \neq 1} \Pr(S_0[1] = X \ \wedge \ S_0[X] = 1) \\
&= \Pr(S_0[1] = 1) + \sum_{X \neq 1} \Pr(S_0[1] = X \ \wedge \ S_0[X] = 1).
\end{aligned}$$

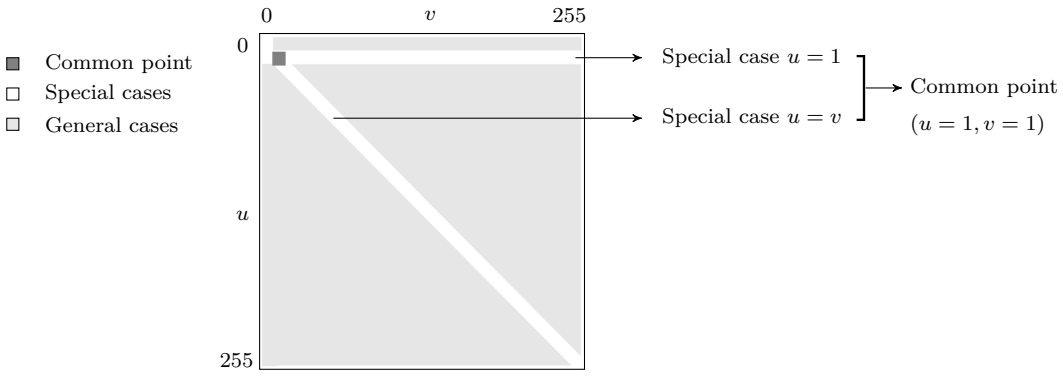

Figure 3.1: u, v dependent special cases and range of sums for evaluation of $\Pr(S_1[u] = v)$ in terms of S_0.

Special case $u = 1, v \neq 1$*:* In this case, $S_0[1] = X = 1$ implies $S_1[u] = S_1[1] = S_0[1]$, as before, and $S_0[1] = X = v$ implies $S_1[u] = S_1[1] = S_0[v]$. If $X \neq 1, v$, we have $S_1[u] = S_1[1] = S_0[X]$. Thus,

$$\begin{aligned}
&\Pr(S_1[1] = v) \\
&= \Pr(S_0[1] = 1 \ \wedge \ S_0[1] = v) + \Pr(S_0[1] = 1 \ \wedge \ S_0[v] = 1) \\
&+ \sum_{X \neq 1,v} \Pr(S_0[1] = X \ \wedge \ S_0[X] = v) \\
&= \ 0 + 0 + \sum_{X \neq 1,v} \Pr(S_0[1] = X \ \wedge \ S_0[X] = v).
\end{aligned}$$

Special case $u \neq 1, v = u$*:* In this case, $S_0[1] = X = 1$ implies no swap, resulting in $S_1[u] = S_0[u]$. Again, $S_0[1] = X = u$ implies $S_1[u] = S_0[1]$, and if $X \neq 1, u$, we have $S_1[u] = S_0[u]$. Thus,

$$\begin{aligned}
&\Pr(S_1[u] = u) \\
&= \Pr(S_0[1] = 1 \ \wedge \ S_0[u] = u) + \Pr(S_0[1] = u \ \wedge \ S_0[1] = u) \\
&+ \sum_{X \neq 1,u} \Pr(S_0[1] = X \ \wedge \ S_0[u] = u) \\
&= \Pr(S_0[1] = u) + \sum_{X \neq u} \Pr(S_0[1] = X \ \wedge \ S_0[u] = u).
\end{aligned}$$

General case $u \neq 1, v \neq u$*:* In this case, $S_0[1] = X = 1$ implies no swap, resulting in $S_1[u] = S_0[u]$. Again, $S_0[1] = X = u$ implies $S_1[u] = S_0[1]$,

and if $X \neq 1, u$, we have $S_1[u] = S_0[u]$. Thus,

$$\begin{aligned}
&\Pr(S_1[u] = v) \\
&= \Pr(S_0[1] = 1 \ \wedge \ S_0[u] = v) + \Pr(S_0[1] = u \ \wedge \ S_0[1] = v) \\
&+ \Pr(S_0[1] = v \ \wedge \ S_0[u] = v) + \sum_{X \neq 1,u,v} \Pr(S_0[1] = X \ \wedge \ S_0[u] = v) \\
&= \Pr(S_0[1] = 1 \ \wedge \ S_0[u] = v) + 0 \\
&+ 0 + \sum_{X \neq 1,u,v} \Pr(S_0[1] = X \ \wedge \ S_0[u] = v) \\
&= \sum_{X \neq u,v} \Pr(S_0[1] = X \ \wedge \ S_0[u] = v).
\end{aligned}$$

Combining all the above cases together, we obtain the desired result. □

The probabilities depending on S_0 can be derived from Proposition 3.1. The estimation of the joint probabilities $\Pr(S_0[u] = v \ \wedge \ S_0[u'] = v')$ is also required for our next result, i.e., Theorem 3.3, as well as for our results in Section 5.1. This estimation is explained in detail in Section 5.1.3.

In Theorem 3.3, we find the probability distribution of $S_{u-1}[u] = v$, just before index i touches the position u during PRGA. This is a generalization of [131, Theorem 4].

Theorem 3.3. *In RC4 PRGA, for* $3 \leq u \leq N-1$,

$$\begin{aligned}
\Pr(S_{u-1}[u] = v) \approx &\Pr(S_1[u] = v) \left(1 - \frac{1}{N}\right)^{u-2} \\
&+ \sum_{t=2}^{u-1} \sum_{w=0}^{u-t} \frac{\Pr(S_1[t] = v)}{w! \cdot N} \left(\frac{u-t-1}{N}\right)^{w} \left(1 - \frac{1}{N}\right)^{u-3-w}.
\end{aligned}$$

Proof. From Lemma 3.2, we know that the event $\Pr(S_1[u] = v)$ is positively biased for all u. Hence the natural path for investigation is as follows:

$$\begin{aligned}
\Pr(S_{u-1}[u] = v) = &\Pr(S_{u-1}[u] = v \mid S_1[u] = v) \cdot \Pr(S_1[u] = v) \\
&+ \Pr(S_{u-1}[u] = v \mid S_1[u] \neq v) \cdot \Pr(S_1[u] \neq v).
\end{aligned}$$

Case $(S_1[u] = v)$*:* Index i varies from 2 to $(u-1)$ during the evolution of S_1 to S_{u-1}, and hence never touches the u-th index. Thus, the index u will retain

its value $S_1[u]$ if index j does not touch it. The probability of this event is $(1-1/N)^{u-2}$ over all the intermediate rounds. Hence we get:

$$\Pr(S_{u-1}[u] = v \mid S_1[u] = v) \cdot \Pr(S_1[u] = v) = \left(1 - \frac{1}{N}\right)^{u-2} \cdot \Pr(S_1[u] = v).$$

Case ($S_1[u] \neq v$)*:* Suppose that $S_1[t] = v$ for some $t \neq u$. In such a case, only a swap between the positions u and t during rounds 2 to $(u-1)$ of PRGA can result in $(S_{u-1}[u] = v)$. If index i does not touch the t-th location, then the value at $S_1[t]$ can only go to some position behind $i \leq u-1$, and can never reach $S_{u-1}[u]$. Thus we must have i touching the t-th position, i.e., $2 \leq t \leq u-1$.

Now suppose that it requires $(w+1)$ hops for v to reach from $S_1[t]$ to $S_{u-1}[u]$. The transfer will never happen if the position t swaps with any index which is not touched by i later. This fraction of favorable positions start from $(u-t-1)/N$ for the first hop and decreases approximately to $(u-t-1)/(lN)$ at the l-th hop. It is also required that j does not touch the position u for the remaining $(u-3-w)$ rounds. Thus, the second part of the probability for a specific position t is:

$$\begin{aligned}
&\Pr(S_1[t] = v) \left(\prod_{l=1}^{w} \frac{u-t-1}{lN}\right) \left(1 - \frac{1}{N}\right)^{u-3-w} \\
&= \frac{\Pr(S_1[t] = v)}{w! \cdot N} \left(\frac{u-t-1}{N}\right)^{w} \left(1 - \frac{1}{N}\right)^{u-3-w}.
\end{aligned}$$

Finally, the number of hops is bounded as $1 \leq w+1 \leq u-t+1$ (here $w+1=1$ or $w=0$ denotes a single-hop transfer), depending on the initial gap between t and u positions. Summing over all t, k with their respective bounds, we get the desired expression for $\Pr(S_{u-1}[u] = v)$. □

3.1.2 Proofs of the keylength dependent biases

Observation of the biases (3.2) was first reported in [131, Section 3], but without any proof. In this section, we present complete proofs of all these biases. Although the biases are all conditional in nature, for ease of understanding we first compute the associated joint probabilities and then discuss how the conditional probabilities can be computed. All the biases that we are inter-

ested in are related to $(S_{l+1}^K[l-1] = -l \ \wedge \ S_{l+1}^K[l] = 0)$. So we first derive the probability for this event.

Lemma 3.4. *Suppose that l is the length of the secret key of RC4. Then*

$$\Pr(S_{l+1}^K[l-1] = -l \ \wedge \ S_{l+1}^K[l] = 0) \approx \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right)\alpha_l,$$
$$\textit{where } \alpha_l = \frac{1}{N}\left(1 - \frac{3}{N}\right)^{l-2}\left(1 - \frac{l+1}{N}\right).$$

Proof. The major path that leads to the target event is as follows.

- In the first round of the KSA, when $i_1^K = 0$ and $j_1^K = K[0]$, the value 0 is swapped into the index $S^K[K[0]]$ with probability 1.
- The index $j_1^K = K[0] \notin \{l-1, l, -l\}$, so that the values $l-1, l, -l$ at these indices respectively are not swapped out in the first round of the KSA. We as well require $K[0] \notin \{1, \ldots, l-2\}$, so that the value 0 at index $K[0]$ is not touched by these values of i^K during the next $l-2$ rounds of the KSA. This happens with probability $\left(1 - \frac{l+1}{N}\right)$.
- From round 2 to $l-1$ (i.e., for $i^K = 1$ to $l-2$) of the KSA, none of $j_2^K, \ldots, j_{l-1}^K$ touches the three indices $\{l, -l, K[0]\}$. This happens with probability $\left(1 - \frac{3}{N}\right)^{l-2}$.
- In round l of the KSA, when $i_l^K = l-1$, j_l^K becomes $-l$ with probability $\frac{1}{N}$, thereby moving $-l$ into index $l-1$.
- In round $l+1$ of the KSA, when $i_{l+1}^K = l$, j_{l+1}^K becomes $j_l^K + S_l^K[l] + K[l] = -l + l + K[0] = K[0]$, and as discussed above, this index contains the value 0. Hence, after the swap, $S_{l+1}^K[l] = 0$. Since $K[0] \neq l-1$, we have $S_{l+1}^K[l-1] = -l$.

Considering the above events to be independent, the probability that all of above occur together is given by $\alpha_l = \frac{1}{N}\left(1 - \frac{3}{N}\right)^{l-2}\left(1 - \frac{l+1}{N}\right)$. If the above path does not occur, then the target event happens due to random association, with probability $\frac{1}{N^2}$, thus contributing a probability of $(1-\alpha_l)\frac{1}{N^2}$. Adding the two contributions, the result follows. □

Now we may derive the joint probabilities associated with the conditional events of (3.2), as follows.

Theorem 3.5. *Suppose that l is the length of the secret key of RC4. Then*

$$\Pr(S_l[l] = -l \wedge S_l[j_l] = 0) = \Pr(t_l = -l \wedge S_l[j_l] = 0) \approx \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right)\beta_l,$$
$$\textit{where } \beta_l = \frac{1}{N}\left(1 - \frac{1}{N}\right)\left(1 - \frac{2}{N}\right)^{N-3}\left(1 - \frac{3}{N}\right)^{l-2}\left(1 - \frac{l+1}{N}\right).$$

Proof. From the proof of Lemma 3.4, consider the major path with probability α_l for the event $(S_{l+1}^K[l-1] = -l \ \wedge \ S_{l+1}^K[l] = 0)$. For the remaining $N - l - 1$ rounds of the KSA and for the first $l - 2$ rounds of the PRGA (i.e., for a total of $N - 3$ rounds), none of the values of j^K (corresponding to the KSA rounds) or j (corresponding to the PRGA rounds) should touch the indices $\{l - 1, l\}$. This happens with a probability of $\left(1 - \frac{2}{N}\right)^{N-3}$.

Now, in round $l - 1$ of PRGA, $i_{l-1} = l - 1$, from where the value $-l$ moves to index j_{l-1} due to the swap. In the next round, $i_l = l$ and $j_l = j_{l-1} + S_{l-1}[l] = j_{l-1}$, provided the value 0 at index l had not been swapped out by j_{l-1}, the probability of which is $1 - \frac{1}{N}$. So during the next swap, the value $-l$ moves from index j_l to index l and the value 0 moves from index l to j_l. The probability of the above major path leading to the event $(S_l[l] = -l \ \wedge \ S_l[j_l] = 0)$ is given by $\beta_l = \alpha_l \left(1 - \frac{2}{N}\right)^{N-3}\left(1 - \frac{1}{N}\right)$. If this path does not occur, then there is always a chance of $\frac{1}{N^2}$ for the target event to happen due to random association. Adding the two contributions and substituting the value of α_l from Lemma 3.4, the result follows.

Further, as $t_l = S_l[l] + S_l[j_l]$, the event $(S_l[l] = -l \wedge S_l[j_l] = 0)$ is equivalent to the event $(t_l = -l \ \wedge \ S_l[j_l] = 0)$, and hence the result. □

Theorem 3.6. *Suppose that l is the length of the secret key of RC4. Then*

$$\Pr(Z_l = -l \ \wedge \ S_l[j_l] = 0) \approx \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right)\gamma_l,$$
$$\textit{where } \gamma_l = \frac{1}{N^2}\left(1 - \frac{l+1}{N}\right)\sum_{x=l+1}^{N-1}\left(1 - \frac{1}{N}\right)^{x}\left(1 - \frac{2}{N}\right)^{x-l}\left(1 - \frac{3}{N}\right)^{N-x+2l-4}.$$

Proof. From the PRGA update rule, we have $j_l = j_{l-1} + S_{l-1}[l]$. Hence, $S_l[j_l] = S_{l-1}[l] = 0$ implies $j_l = j_{l-1}$ as well as $Z_l = S_l[S_l[l] + S_l[j_l]] = S_l[S_{l-1}[j_l] + 0] =$

$S_l[S_{l-1}[j_{l-1}]] = S_l[S_{l-2}[l-1]]$. Thus, the event $(Z_l = -l \ \wedge \ S_l[j_l] = 0)$ is equivalent to the event $(S_l[S_{l-2}[l-1]] = -l \ \wedge \ S_{l-1}[l] = 0)$.

From the proof of Lemma 3.4, consider the major path with probability α_l for the joint event $(S^K_{l+1}[l-1] = -l \wedge S^K_{l+1}[l] = 0)$. This constitutes the first part of our main path leading to the target event. The second part, having probability α'_l, can be constructed as follows.

- For an index $x \in [l+1, N-1]$, we have $S^K_x[x] = x$. This happens with probability $\left(1 - \frac{1}{N}\right)^x$.
- For the KSA rounds $l+2$ to x, the j^K values do not touch the indices $l-1$ and l. This happens with probability $\left(1 - \frac{2}{N}\right)^{x-l-1}$.
- In round $x+1$ of KSA, when $i^K_{x+1} = x$, j^K_{x+1} becomes $l-1$ with probability $\frac{1}{N}$. Due to the swap, the value x moves to $S^K_{x+1}[l-1]$ and the value $-l$ moves to $S^K_{x+1}[x] = S^K_{x+1}[S^K_{x+1}[l-1]]$.
- For the remaining $N-x-1$ rounds of the KSA and for the first $l-1$ rounds of the PRGA, none of the j^K or j values should touch the indices $\{l-1, S[l-1], l\}$. This happens with a probability of $\left(1 - \frac{3}{N}\right)^{N-x+l-2}$.
- So far, we have $(S_{l-1}[S_{l-2}[l-1]] = -l \ \wedge \ S_{l-1}[l] = 0)$. Now, we should also have $j_l \notin \{l-1, S[l-1]\}$ for $S_l[S_{l-2}[l-1]] = S_{l-1}[S_{l-2}[l-1]] = -l$. The probability of this condition is $\left(1 - \frac{2}{N}\right)$.

Assuming all the individual events in the above path to be mutually independent, we get

$$\alpha'_l = \frac{1}{N} \sum_{x=l+1}^{N-1} \left(1 - \frac{1}{N}\right)^x \left(1 - \frac{2}{N}\right)^{x-l} \left(1 - \frac{3}{N}\right)^{N-x+l-2}.$$

Thus, the probability of the entire path is given by $\gamma_l = \alpha_l \cdot \alpha'_l$, that is,

$$\gamma_l = \frac{1}{N^2}\left(1 - \frac{l+1}{N}\right) \sum_{x=l+1}^{N-1} \left(1 - \frac{1}{N}\right)^x \left(1 - \frac{2}{N}\right)^{x-l} \left(1 - \frac{3}{N}\right)^{N-x+2l-4}.$$

If this path does not occur, then there is always a chance of $\frac{1}{N^2}$ for the target event to happen due to random association. Adding the two contributions, we get the result. □

In order to calculate the conditional probabilities of (3.2), we need to compute the marginals $\delta_l = \Pr(S_l[j_l] = 0)$ and $\tau_l = \Pr(t_l = -l)$. Our experimental observations reveal that in $5 \leq l \leq 32$, δ_l does not change much with l, and has a slightly negative bias: $\delta_l \approx 1/N - 1/N^2$. On the other hand, as l varies from 5 to 32, τ_l changes approximately from $1.13/N$ to $1.08/N$. We can derive the exact expression for δ_l as a corollary to Theorem 3.3, and an expression for τ_l using δ_l.

Corollary 3.7. *For any keylength l, with $3 \leq l \leq N-1$,*

$$\Pr(S_l[j_l] = 0) = \delta_l \approx \Pr(S_1[l] = 0)\left(1 - \frac{1}{N}\right)^{l-2} + \sum_{t=2}^{l-1}\sum_{w=0}^{l-t} \frac{\Pr(S_1[t] = 0)}{w! \cdot N}\left(\frac{l-t-1}{N}\right)^{w}\left(1 - \frac{1}{N}\right)^{l-3-w}.$$

Proof. Note that $S_l[j_l]$ is assigned the value of $S_{l-1}[l]$ due to the swap in round l. Hence, by substituting $u = l$ and $v = 0$ in Theorem 3.3, we get the result. □

Theorem 3.8. *Suppose that l is the length of the secret key of RC4. Then*

$$\tau_l = \Pr(t_l = -l) \approx \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right)\beta_l + (1 - \delta_l)\frac{1}{N},$$

where β_l is given in Theorem 3.5 and δ_l is given in Corollary 3.7.

Proof. We can write

$$\Pr(t_l = -l) = \Pr(t_l = -l \ \wedge \ S_l[j_l] = 0) + \Pr(t_l = -l \ \wedge \ S_l[j_l] \neq 0),$$

where the first term is given by Theorem 3.5. When $S_l[j_l] \neq 0$, the event $(t_l = -l)$ can be assumed to occur due to random association. Hence the second term can be computed as

$$\Pr(S_l[j_l] \neq 0) \cdot \Pr(t_l = -l \mid S_l[j_l] \neq 0) \approx (1 - \delta_l)\frac{1}{N}.$$

Adding the two terms, we get the result. □

Theoretical values for both δ_l and τ_l match closely with the experimental ones for all values of l.

Computing the conditional biases in (3.2): When we divide the joint probabilities $\Pr(S_l[l] = -l \ \wedge \ S_l[j_l] = 0)$ and $\Pr(t_l = -l \ \wedge \ S_l[j_l] = 0)$ of Theorem 3.5, and $\Pr(Z_l = -l \wedge S_l[j_l] = 0)$ of Theorem 3.6 by the appropriate marginals $\delta_l = \Pr(S_l[j_l] = 0)$ of Corollary 3.7 and $\tau_l = \Pr(t_l = -l)$ of Theorem 3.8, we get theoretical values for all the biases in (3.2). The theoretical values closely match with the experimental observations.

3.1.3 Bias in $(Z_l = -l)$ and keylength prediction

First, we prove the bias in (3.3) and thereby show how to predict the length l of RC4 secret key. Next, we use the marginal probability $\Pr(Z_l = -l)$ to derive the conditional probabilities of (3.1).

Theorem 3.9. *Suppose that l is the length of the secret key of RC4. Then*

$$\Pr(Z_l = -l) \ \approx \ \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right)\gamma_l + (1 - \delta_l)\frac{1}{N},$$

where γ_l is given in Theorem 3.6 and δ_l is given in Corollary 3.7.

Proof. We can write

$$\Pr(Z_l = -l) = \Pr(Z_l = -l \ \wedge \ S_l[j_l] = 0) + \Pr(Z_l = -l \ \wedge \ S_l[j_l] \neq 0),$$

where the first term is given by Theorem 3.6. When $S_l[j_l] \neq 0$, the event $(Z_l = -l)$ can be assumed to occur due to random association. Hence the second term can be computed as

$$\Pr(S_l[j_l] \neq 0) \cdot \Pr(Z_l = -l \mid S_l[j_l] \neq 0) \approx (1 - \delta_l)\frac{1}{N}.$$

Adding the two terms, we get the result. □

It is important to note that the estimate of $\Pr(Z_l = -l)$ is always greater than $1/N + 1/N^2 \approx 0.003922$ for $N = 256$ and $5 \leq l \leq 32$. In Figure 3.2, we plot the theoretical as well as the experimental values of $\Pr(Z_l = -l)$ against l for $5 \leq l \leq 32$, where the experiments have been run over 1 billion trials of RC4 PRGA, with randomly generated keys.

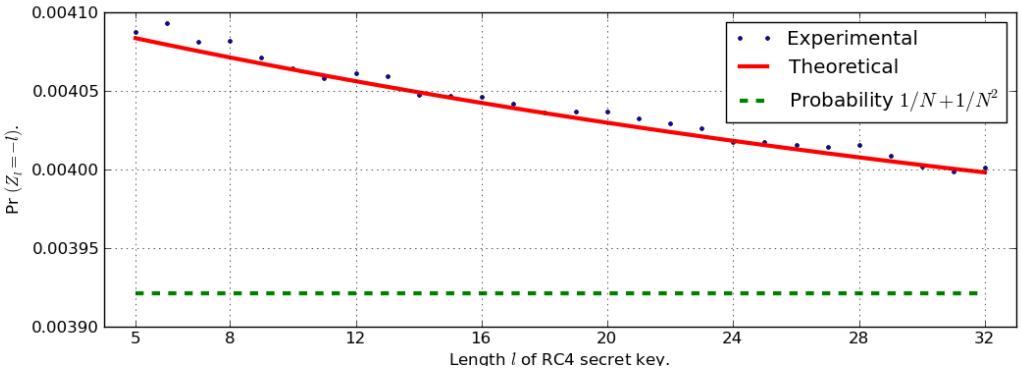

Figure 3.2: Distribution of $\Pr(Z_l = -l)$ for different lengths $5 \leq l \leq 32$ of the RC4 secret key.

Keylength distinguisher: From this estimate, we immediately get a distinguisher of RC4 that can effectively distinguish the output keystream of the cipher from a random sequence of bytes. For the event $E : (Z_l = -l)$, the bias proved in Theorem 3.9 can be written as $p(1+q)$, where $p = 1/N$ and $q > 1/N$ for $5 \leq l \leq 32$ and $N = 256$. Thus, the number of samples required to distinguish RC4 from random sequence of bits with a constant probability of success is approximately $\frac{1}{pq^2} = N^3$. Using this distinguisher, one may predict the length l of RC4 secret key from the output keystream.

Proofs of the keylength-dependent biases in (3.1): To prove the conditional biases in (3.1), we first compute the associated joint probabilities $\Pr\left(S_l[j_l] = 0 \wedge Z_l = -l\right)$ and $\Pr\left(S_{l+1}^K[l] = 0 \wedge Z_l = -l\right)$, and then use the marginal $\Pr(Z_l = -l)$ to obtain the final results. The first joint probability is already computed in Theorem 3.6, and the second one is computed as follows.

Theorem 3.10. *Suppose that l is the length of the secret key of RC4. Then*

$$\begin{aligned}&\Pr(Z_l = -l \ \wedge \ S_{l+1}^K[l] = 0)\\ &\approx \left(\frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right)\alpha_l\right) \cdot \alpha_l' + \left(1 - \frac{1}{N} - \left(1 - \frac{1}{N^2}\right)\alpha_l\right) \cdot \frac{1}{N^2},\end{aligned}$$

where α_l is given in Lemma 3.4 and α_l' is given in Theorem 3.6.

Proof. We consider the main path in this case to be $\Pr(S_{l+1}^K[l-1] = -l \ \wedge S_{l+1}^K[l] = 0)$, which occurs with probability $\frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right)\alpha_l$, as in Lemma 3.4.

We also need to compute $\Pr(S_{l+1}^K[l-1] = -l)$. Since i^K in round $l+1$ has touched the index l, the value at this position can be assumed to be random. Thus, we may assume $\Pr(S_{l+1}^K[l] = 0) \approx \frac{1}{N}$, and hence

$$\begin{aligned}
&\Pr(S_{l+1}^K[l-1] = -l) \\
&= \Pr(S_{l+1}^K[l-1] = -l \ \wedge \ S_{l+1}^K[l] = 0) + \Pr(S_{l+1}^K[l-1] = -l \ \wedge \ S_{l+1}^K[l] \neq 0) \\
&= \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right)\alpha_l + \Pr(S_{l+1}^K[l] \neq 0) \cdot \Pr(S_{l+1}^K[l-1] = -l \mid S_{l+1}^K[l] \neq 0) \\
&\approx \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right)\alpha_l + \left(1 - \frac{1}{N}\right)\frac{1}{N} = \frac{1}{N} + \left(1 - \frac{1}{N^2}\right)\alpha_l.
\end{aligned}$$

Now, we may compute the main probability $\Pr(Z_l = -l \ \wedge \ S_{l+1}^K[l] = 0)$, as

$$\begin{aligned}
&\Pr(Z_l = -l \wedge S_{l+1}^K[l] = 0 \wedge S_{l+1}^K[l-1] = -l) \\
&+ \Pr(Z_l = -l \wedge S_{l+1}^K[l] = 0 \wedge S_{l+1}^K[l-1] \neq -l) \\
&= \Pr(S_{l+1}^K[l] = 0 \wedge S_{l+1}^K[l-1] = -l) \cdot \Pr(Z_l = -l \mid \\
&\qquad\qquad S_{l+1}^K[l] = 0 \wedge S_{l+1}^K[l-1] = -l) \\
&+ \Pr(S_{l+1}^K[l-1] \neq -l) \cdot \Pr(Z_l = -l \wedge S_{l+1}^K[l] = 0 \mid S_{l+1}^K[l-1] \neq -l).
\end{aligned}$$

From Lemma 3.4 and proof of Theorem 3.6, the first part is approximated by $\left(\frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right)\alpha_l\right) \cdot \alpha_l'$. In the second part, we assume that when $S_{l+1}^K[l-1] \neq -l$, with probability $1 - \frac{1}{N} - \left(1 - \frac{1}{N^2}\right)\alpha_l$, then the event $(Z_l = -l \wedge S_{l+1}^K[l] = 0)$ happens due to random association, with probability $\frac{1}{N^2}$. Adding the contributions from the two parts as above, we obtain the result. □

If we divide $\Pr\left(S_l[j_l] = 0 \wedge Z_l = -l\right)$, as obtained from Theorem 3.6, and $\Pr\left(S_{l+1}^K[l] = 0 \wedge Z_l = -l\right)$, as obtained from Theorem 3.10, by $\Pr(Z_l = -l)$ of Theorem 3.9, we get the desired conditional probabilities for the events $(S_l[j_l] = 0 \mid Z_l = -l)$ and $\left(S_{l+1}^K[l] = 0 \mid Z_l = -l\right)$ respectively. These theoretical estimates closely match with our experimental observations. For an example, in case of $l = 16$, from simulations with 1 billion randomly generated secret keys, we obtained the experimental values of the above probabilities as 9.7/256 and 9.5/256 (approx.) respectively, whereas the theoretical values are close to 9.6/256 for both cases.

3.2 Extended keylength dependent biases

In [132, Section 2], we presented a family of biases in RC4 that are dependent on the length of the secret key. The most important of those biases was a keylength distinguisher based on the positive bias in the event $(Z_l = -l)$, where l is the length of RC4 secret key in bytes. This result was discussed in the previous section.

Subsequently, in [70, Section 3.4], Isobe, Ohigashi, Watanabe and Morii observed that similar bias also exists in the class of events $(Z_{xl} = -xl)$ for positive integer $x = 1, \ldots, \lfloor N/l \rfloor$. To prove these biases, they explored certain paths involving the expression $f_y = y(y+1)/2 + \sum_{x=0}^{y} K[x]$. However, they could not prove all the paths and substituted experimental values to compute what they referred to as *semi-theoretical values*. They also commented that

> *"Since semi-theoretical value are partially based on experimental results, we can not claim that complete theoretical proof of these bias are given."*

We observe that instead of following the approach of [70], if one follows the approach in [132], then the theoretical derivation of the extended keylength dependent biases become much simpler. In this section, we generalize all the keylength dependent biases of [132] for any keylength $l \in [3, N-1]$ and any integer $x = 1, 2, \ldots, \lfloor \frac{N}{l} \rfloor$, and thereby complete the proof of extended keylength distinguisher that was left open in [70]. As a result, the biases in [132] (presented in the previous section) become special cases of our results presented in this section, if we take $x = 1$.

Note that though the general proof follows the same approach as in [132], the extension is not obvious. A general proof always imply the special cases, but the converse need not be true. We experimentally verified all the intermediate claims and assumptions related to the events involving xl, and we found them to be consistent with our theoretical claims.

We will require some existing results for our proofs in this section. Proposition 3.1 from the previous section (originally [100, Theorem 6.2.1]) and Lemma 3.2 (revised version of [132, Lemma 1]) will be needed, along with the following result, which is a revised version of [132, Theorem 1].

Proposition 3.11. *In RC4 PRGA, for* $3 \leq u \leq N-1$ *and* $0 \leq v \leq N-1$,

$$\Pr(S_{u-1}[u] = v) \approx \Pr(S_1[u] = v)\left(1 - \frac{1}{N}\right)^{u-2} + \sum_{y=2}^{u-1}\sum_{w=0}^{u-y} \frac{\Pr(S_1[y] = v)}{w! \cdot N}\left(\frac{u-y-1}{N}\right)^{w}\left(1 - \frac{1}{N}\right)^{u-3-w}.$$

3.2.1 Proofs of extended keylength dependent biases

All the biases that we are interested in are related to $(S^K_{xl+1}[xl-1] = -xl \wedge S^K_{xl+1}[xl] = 0)$, where x is an integer between 1 and $\lfloor \frac{N}{l} \rfloor$. So we first derive the probability for this event in Lemma 3.12.

Lemma 3.12. *Suppose that* l *is the length of the secret key of RC4. Then for* $1 \leq x \leq \lfloor \frac{N}{l} \rfloor$, *we have* $\Pr(S^K_{xl+1}[xl-1] = -xl \wedge S^K_{xl+1}[xl] = 0) \approx \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right)\alpha_{x,l}$, *where* $\alpha_{x,l} = \frac{1}{N}\left(1 - \frac{3}{N}\right)^{xl-2}\left(1 - \frac{xl+1}{N}\right)$.

Proof. The major path that leads to the target event is as follows.

- In the first round of the KSA, when $i^K_1 = 0$ and $j^K_1 = K[0]$, the value 0 is swapped into the index $S^K[K[0]]$ with probability 1.
- The index $j^K_1 = K[0] \notin \{xl-1, xl, -xl\}$, so that the values $xl-1, xl, -xl$ at these indices respectively are not swapped out in the first round of the KSA. We as well require $K[0] \notin \{1, \ldots, xl-2\}$, so that the value 0 at index $K[0]$ is not touched by these values of i^K during the next $xl-2$ rounds of the KSA. This happens with probability $\left(1 - \frac{xl+1}{N}\right)$.
- From round 2 to $xl-1$ (i.e., for $i^K = 1$ to $xl-2$) of the KSA, none of $j^K_2, \ldots, j^K_{xl-1}$ touches the three indices $\{xl, -xl, K[0]\}$. This happens with probability $\left(1 - \frac{3}{N}\right)^{xl-2}$.
- In round xl of the KSA, when $i^K_{xl} = xl-1$, j^K_{xl} becomes $-xl$ with probability $\frac{1}{N}$, thereby moving $-xl$ into index $xl-1$.
- In round $xl+1$ of the KSA, when $i^K_{xl+1} = xl$, j^K_{xl+1} becomes $j^K_{xl} + S^K_{xl}[xl] + K[xl] = -xl + xl + K[0] = K[0]$, and as discussed above, this index contains the value 0. Hence, after the swap, $S^K_{xl+1}[xl] = 0$. Since $K[0] \neq xl-1$, we have $S^K_{xl+1}[xl-1] = -xl$.

Considering the above events to be independent, the probability that all of above occur together is given by $\alpha_{x,l} = \frac{1}{N}\left(1-\frac{3}{N}\right)^{xl-2}\left(1-\frac{xl+1}{N}\right)$. If the above path does not occur, then the target event happens due to random association with probability $\frac{1}{N^2}$, thus contributing a probability of $(1-\alpha_{x,l})\frac{1}{N^2}$. Adding the two contributions, the result follows. □

Theorem 3.13. *Suppose that l is the length of the secret key of RC4. Then for $1 \le x \le \lfloor \frac{N}{l} \rfloor$, we have* $\Pr(S_{xl}[xl] = -xl \ \wedge \ S_{xl}[j_{xl}] = 0) = \Pr(t_{xl} = -xl \ \wedge \ S_{xl}[j_{xl}] = 0) \approx \frac{1}{N^2} + \left(1-\frac{1}{N^2}\right)\beta_{x,l}$, *where* $\beta_{x,l} = \frac{1}{N}\left(1-\frac{1}{N}\right)\left(1-\frac{2}{N}\right)^{N-3}\left(1-\frac{3}{N}\right)^{xl-2}\left(1-\frac{xl+1}{N}\right)$.

Proof. From the proof of Lemma 3.12, consider the major path with probability $\alpha_{x,l}$ for the event $(S^K_{xl+1}[xl-1] = -xl \ \wedge \ S^K_{xl+1}[xl] = 0)$. For the remaining $N - xl - 1$ rounds of the KSA and for the first $xl - 2$ rounds of the PRGA (i.e., for a total of $N - 3$ rounds), none of the values of j^K (corresponding to the KSA rounds) or j (corresponding to the PRGA rounds) should touch the indices $\{xl-1, xl\}$. This happens with a probability of $\left(1-\frac{2}{N}\right)^{N-3}$.

Now, in round $xl - 1$ of PRGA, $i_{xl-1} = xl - 1$, from where the value $xl - 1$ moves to index j_{xl-1} due to the swap. In the next round, $i_{xl} = xl$ and $j_{xl} = j_{xl-1} + S_{xl-1}[xl] = j_{xl-1}$, provided the value 0 at index xl had not been swapped out by j_{xl-1}, the probability of which is $1 - \frac{1}{N}$. So during the next swap, the value $-xl$ moves from index j_{xl} to index xl and the value 0 moves from index xl to j_{xl}. The probability of the above major path leading to the event $(S_{xl}[xl] = -xl \wedge S_{xl}[j_{xl}] = 0)$ is given by $\beta_{x,l} = \alpha_{x,l}\left(1-\frac{2}{N}\right)^{N-3}\left(1-\frac{1}{N}\right)$. If this path does not occur, then there is always a chance of $\frac{1}{N^2}$ for the target event to happen due to random association. Adding the two contributions and substituting the value of $\alpha_{x,l}$ from Lemma 3.12, the result follows.

Further, as $t_{xl} = S_{xl}[xl] + S_{xl}[j_{xl}]$, the event $(S_{xl}[xl] = -xl \ \wedge \ S_{xl}[j_{xl}] = 0)$ is equivalent to the event $(t_{xl} = -xl \ \wedge \ S_{xl}[j_{xl}] = 0)$, and hence the result. □

Theorem 3.14. *Suppose that l is the length of the secret key of RC4. Then for $1 \le x \le \lfloor \frac{N}{l} \rfloor$, we have*

$$\Pr(Z_{xl} = -xl \ \wedge \ S_{xl}[j_{xl}] = 0) \approx \frac{1}{N^2} + \left(1-\frac{1}{N^2}\right)\gamma_{x,l},$$

where $\gamma_{x,l} = \frac{1}{N^2}\left(1-\frac{xl+1}{N}\right)\sum_{u=xl+1}^{N-1}\left(1-\frac{1}{N}\right)^{u}\left(1-\frac{2}{N}\right)^{u-xl}\left(1-\frac{3}{N}\right)^{N-u+2xl-4}$.

Proof. From the PRGA update rule, we have $j_{xl} = j_{xl-1} + S_{xl-1}[xl]$. Hence, $S_{xl}[j_{xl}] = S_{xl-1}[xl] = 0$ implies $j_{xl} = j_{xl-1}$ as well as $Z_{xl} = S_{xl}[S_{xl}[xl] + S_{xl}[j_{xl}]] = S_{xl}[S_{xl-1}[j_{xl}] + 0] = S_{xl}[S_{xl-1}[j_{xl-1}]] = S_{xl}[S_{xl-2}[xl-1]]$. Thus, the event $(Z_{xl} = -xl \wedge S_{xl}[j_{xl}] = 0)$ is equivalent to the event $(S_{xl}[S_{xl-2}[xl-1]] = -xl \wedge S_{xl-1}[xl] = 0)$.

From the proof of Lemma 3.12, consider the major path with probability α_{xl} for the joint event $(S^K_{xl+1}[xl-1] = -xl \wedge S^K_{xl+1}[xl] = 0)$. This constitutes the first part of our main path leading to the target event. The second part, having probability $\alpha'_{x,l}$, can be constructed as follows.

- For an index $u \in [xl+1, N-1]$, we have $S^K_u[u] = u$. This happens with probability $\left(1 - \frac{1}{N}\right)^u$.
- For the KSA rounds $xl+2$ to u, the j^K values do not touch the indices $xl-1$ and xl. This happens with probability $\left(1 - \frac{2}{N}\right)^{u-xl-1}$.
- In round $u+1$ of KSA, when $i^K_{u+1} = u$, j^K_{u+1} becomes $xl-1$ with probability $\frac{1}{N}$. Due to the swap, the value u moves to $S^K_{u+1}[xl-1]$ and the value $-xl$ moves to $S^K_{u+1}[u] = S^K_{u+1}[S^K_{u+1}[xl-1]]$.
- For the remaining $N-u-1$ rounds of the KSA and for the first $xl-1$ rounds of the PRGA, none of the j^K or j values should touch the indices $\{xl-1, S[xl-1], xl\}$. This happens with a probability of $\left(1 - \frac{3}{N}\right)^{N-u+xl-2}$.
- So far, we have $(S_{xl-1}[S_{xl-2}[xl-1]] = -xl \wedge S_{xl-1}[xl] = 0)$. Now, we should also have $j_{xl} \notin \{xl-1, S[xl-1]\}$ for $S_{xl}[S_{xl-2}[xl-1]] = S_{xl-1}[S_{xl-2}[xl-1]] = -xl$. The probability of this condition is $\left(1 - \frac{2}{N}\right)$.

Assuming all the individual events in the above path to be mutually independent, we get $\alpha'_{x,l} = \frac{1}{N}\sum_{u=xl+1}^{N-1}\left(1-\frac{1}{N}\right)^u\left(1-\frac{2}{N}\right)^{u-xl}\left(1-\frac{3}{N}\right)^{N-u+xl-2}$. Thus, the probability of the entire path is given by $\gamma_{x,l} = \alpha_{x,l} \cdot \alpha'_{x,l}$, that is

$$\gamma_{x,l} = \frac{1}{N^2}\left(1 - \frac{xl+1}{N}\right)\sum_{u=xl+1}^{N-1}\left(1-\frac{1}{N}\right)^u\left(1-\frac{2}{N}\right)^{u-xl}\left(1-\frac{3}{N}\right)^{N-u+2xl-4}.$$

If this path does not occur, then probability of occurrence is $\frac{1}{N^2}$ due to random association. Adding the two contributions, we get the result. □

Theorem 3.15. *For any length $3 \leq l \leq N-1$ of the secret key, and any integer $1 \leq x \leq \lfloor \frac{N}{l} \rfloor$, the probability $\Pr(S_{xl}[j_{xl}] = 0)$ is given by*

$$\delta_{x,l} \approx \Pr(S_1[xl] = 0) \left(1 - \frac{1}{N}\right)^{xl-2} + \sum_{y=2}^{xl-1} \sum_{w=0}^{xl-y} \frac{\Pr(S_1[y] = 0)}{w! \cdot N} \left(\frac{xl - y - 1}{N}\right)^{w} \left(1 - \frac{1}{N}\right)^{xl-3-w}.$$

Proof. Note that $S_{xl}[j_{xl}]$ is assigned the value of $S_{xl-1}[xl]$ due to the swap in round xl. Hence, by substituting $u = xl$ and $v = 0$ in Proposition 3.11, we get the result. □

Theorem 3.16. *Suppose that l is the length of the secret key of RC4. Then for $1 \leq x \leq \lfloor \frac{N}{l} \rfloor$, we have*

$$\tau_{x,l} = \Pr(t_{xl} = -xl) \approx \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right) \beta_{x,l} + (1 - \delta_{x,l}) \frac{1}{N},$$

where $\beta_{x,l}$ is given in Theorem 3.13 and $\delta_{x,l}$ is given in Theorem 3.15.

Proof. We can write $\Pr(t_{xl} = -xl) = \Pr(t_{xl} = -xl \ \wedge \ S_{xl}[j_{xl}] = 0) + \Pr(t_{xl} = -xl \ \wedge \ S_{xl}[j_{xl}] \neq 0)$, where the first term is given by Theorem 3.13. When $S_{xl}[j_{xl}] \neq 0$, the event $(t_{xl} = -xl)$ can be assumed to occur due to random association. Hence the second term can be computed as

$$\Pr(S_{xl}[j_{xl}] \neq 0) \cdot \Pr(t_{xl} = -xl \mid S_{xl}[j_{xl}] \neq 0) \approx (1 - \delta_{x,l}) \frac{1}{N},$$

where $\delta_{x,l}$ is as in Theorem 3.15. Adding the two terms, we get the result. □

By dividing the joint probabilities $\Pr(S_{xl}[xl] = -xl \ \wedge \ S_{xl}[j_{xl}] = 0)$ and $\Pr(t_{xl} = -xl \ \wedge \ S_{xl}[j_{xl}] = 0)$ of Theorem 3.13, and $\Pr(Z_{xl} = -xl \ \wedge \ S_{xl}[j_{xl}] = 0)$ of Theorem 3.14 by the appropriate marginals $\delta_{x,l} = \Pr(S_{xl}[j_{xl}] = 0)$ of Theorem 3.15 and $\tau_{x,l} = \Pr(t_{x,l} = -xl)$ of Theorem 3.16, we get the theoretical values of the following conditional biases:

$$\Pr(S_{xl}[xl] = -xl \mid S_{xl}[j_{xl}] = 0) = \Pr(t_{xl} = -xl \mid S_{xl}[j_{xl}] = 0),$$
$$\Pr(S_{xl}[j_{xl}] = 0 \mid t_{xl} = -xl), \text{ and } \Pr(Z_{xl} = -xl \mid S_{xl}[j_{xl}] = 0).$$

3.2.2 Keylength dependent bias in $(Z_{xl} = -xl)$

Theorem 3.17. *Suppose that l is the length of the secret key of RC4. Then for $1 \le x \le \lfloor \frac{N}{l} \rfloor$,*

$$\Pr(Z_{xl} = -xl) \approx \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right)\gamma_{x,l} + (1 - \delta_{x,l})\frac{1}{N},$$

where $\gamma_{x,l}$ is given in Theorem 3.14 and $\delta_{x,l}$ is given in Theorem 3.15.

Proof. We can write $\Pr(Z_{xl} = -xl) = \Pr(Z_{xl} = -xl \wedge S_{xl}[j_{xl}] = 0) + \Pr(Z_{xl} = -xl \wedge S_{xl}[j_{xl}] \neq 0)$, where the first term is given by Theorem 3.14. When $S_{xl}[j_{xl}] \neq 0$, the event $(Z_{xl} = -xl)$ can be assumed to occur due to random association. Hence the second term can be computed as $\Pr(S_{xl}[j_{xl}] \neq 0) \cdot \Pr(Z_{xl} = -xl \mid S_{xl}[j_{xl}] \neq 0) \approx (1 - \delta_{x,l})\frac{1}{N}$, where $\delta_{x,l}$ is as in Theorem 3.15. Adding the two terms, we get the result. □

By dividing the joint probability $\Pr(Z_{xl} = -xl \wedge S_{xl}[j_{xl}] = 0)$ of Theorem 3.14 by $\Pr(Z_{xl} = -xl)$ of Theorem 3.17, we get the theoretical value of the probability $\Pr(S_{xl}[j_{xl}] = 0 \mid Z_{xl} = -xl)$.

In Figure 3.3, we compare the experimental values of $(Z_{xl} = -xl)$, obtained from the data of [5,14], with our theoretical values derived from Theorem 3.17, for keylength $l = 16$ and $x = 1, 2, \ldots, 15$. We have obtained similar results for other keylengths as well, and Figures 3.4 to 3.9 represent some of the results.

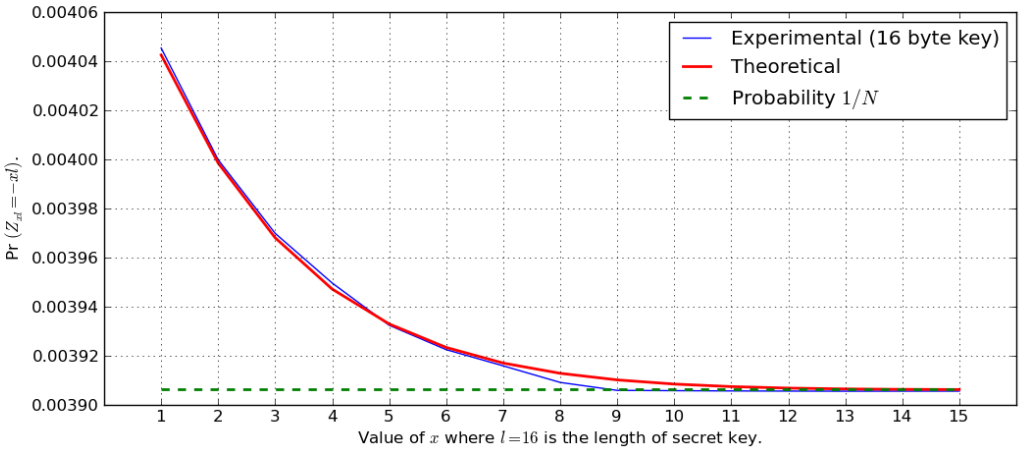

Figure 3.3: Bias in $(Z_{xl} = -xl)$ for keylength $l = 16$ and $x = 1, 2, \ldots, 15$.

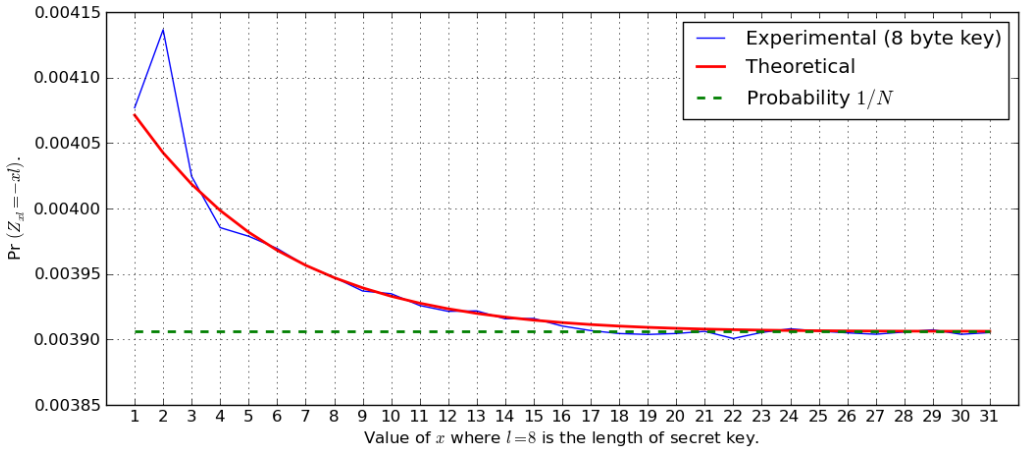

Figure 3.4: Bias in $(Z_{xl} = -xl)$ for keylength $l = 8$ and $x = 1, 2, \ldots, 31$.

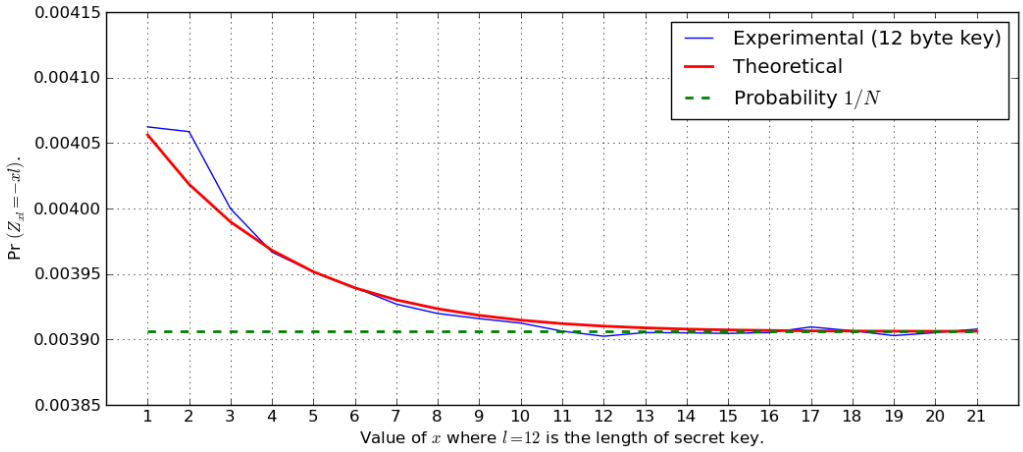

Figure 3.5: Bias in $(Z_{xl} = -xl)$ for keylength $l = 12$ and $x = 1, 2, \ldots, 21$.

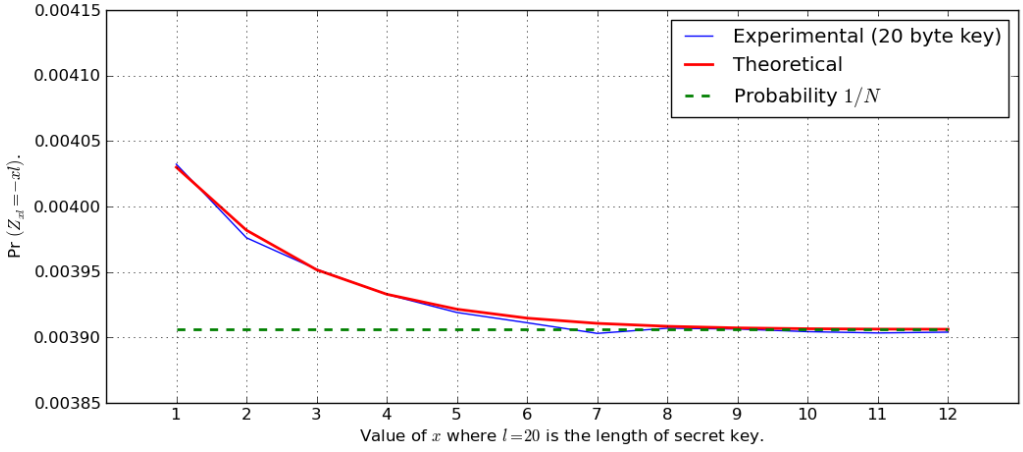

Figure 3.6: Bias in $(Z_{xl} = -xl)$ for keylength $l = 20$ and $x = 1, 2, \ldots, 12$.

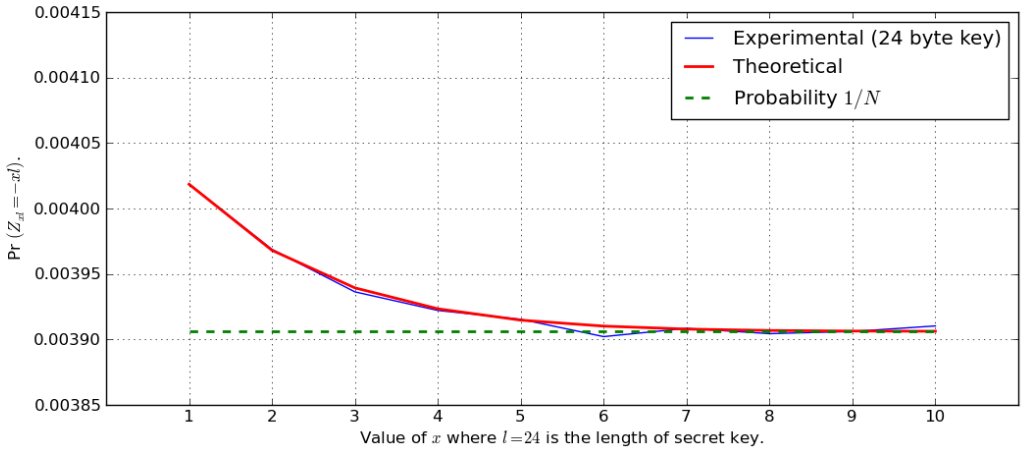

Figure 3.7: Bias in $(Z_{xl} = -xl)$ for keylength $l = 24$ and $x = 1, 2, \ldots, 10$.

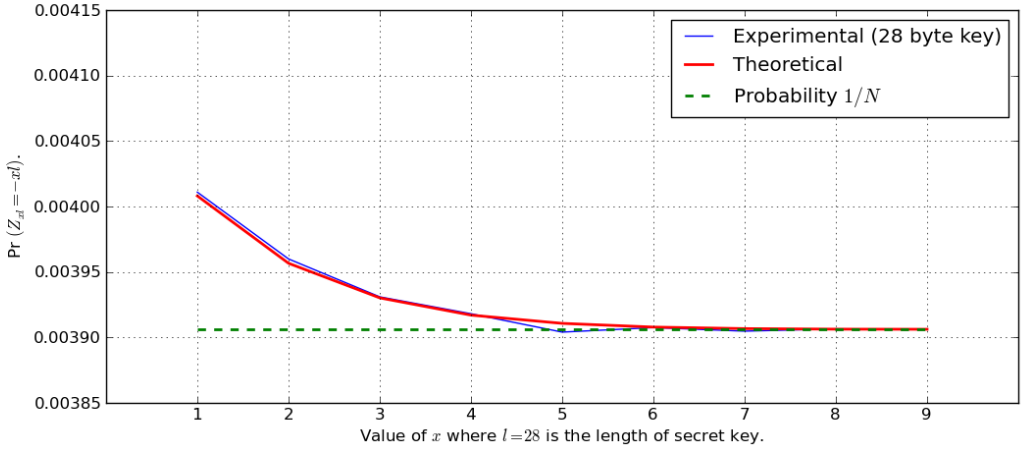

Figure 3.8: Bias in $(Z_{xl} = -xl)$ for keylength $l = 28$ and $x = 1, 2, \ldots, 9$.

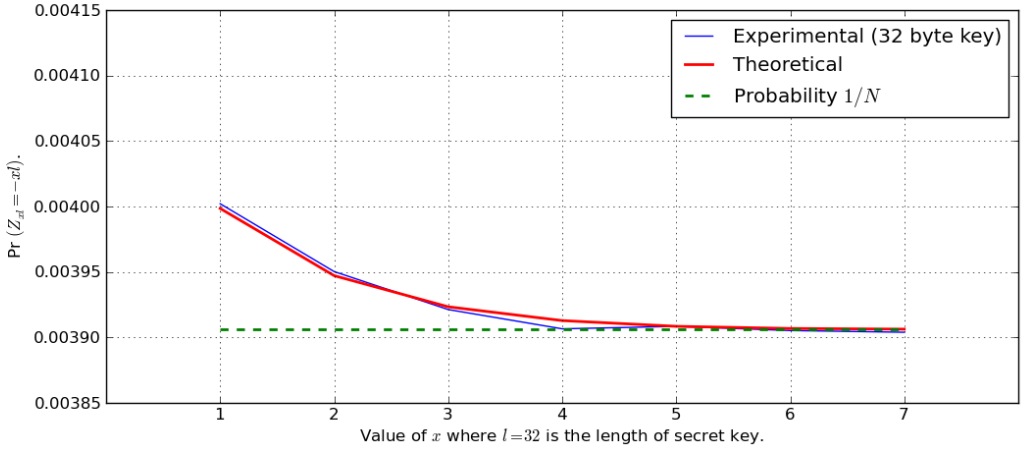

Figure 3.9: Bias in $(Z_{xl} = -xl)$ for keylength $l = 32$ and $x = 1, 2, \ldots, 7$.

The theoretical values for probabilities $\Pr(Z_{xl} = -xl)$ derived in this section closely match the experimental data, as depicted in the figures above. Only in case of Figures 3.4 and 3.5, for $l = 8$ and $l = 12$ respectively, we find a deviation of the experimental data from our theoretical estimates. This deviation however, is prominent only at a single data-point $x = 2$; and we leave it outside the scope of this thesis for a potential future work.

3.3 Keylength dependent bias in first byte

In this section, we attempt at solving the mysterious negative bias in the event $(Z_1 = 129)$, which was observed by [5,14], but not in [112,132]. We notice that the length of the secret key used in the experiments of [5,14] was consistently $l = 16$, whereas the same for [112,132] might have been different. This hinted that the bias in $(Z_1 = 129)$ may be keylength dependent. Our experiments revealed that the negative bias of $(Z_1 = 129)$ is prominent only for keylength l equal to non-trivial factors of 256, that is, for $l = 2, 4, 8, 16, 32, 64, 128$. This behavior, depicted in Figure 3.10, is amazingly similar to the keylength dependence of $\Pr(S_0[128] = 127)$, as in Figure 3.11.

The graphical representation of the complete probability distribution of Z_1 presented in [132, Fig.9] had a typographic error stating that the experimental values are 'with 16 byte keys', whereas the experimental values were actually recorded for full-length 256 byte secret keys. This is why the curve for Z_1 in [132] missed the prominent, but keylength dependent, bias in $(Z_1 = 129)$. We correct the typographic error in Figure 5.2 of Chapter 5.

Our experiments with these specific keylengths $l = 2, 4, \ldots, 128$ revealed that the negative bias in $(S_0[128] = 127)$ is of the same kind. This bias had been pointed out quite a few years ago [100, 120] as an 'anomaly' in the otherwise smooth distribution of $S_0[u] = v$, but it was never observed as a keylength dependent phenomenon. In fact, dependence of keystream biases on the secret keylength l was first proved in [132], for any keylength l, but no such pattern for specific keylengths was discovered earlier.

In this section, we settle the mysterious open issue of the $(S_0[128] = 127)$ anomaly, and then analyze its connection with the bias of $(Z_1 = 129)$, if any.

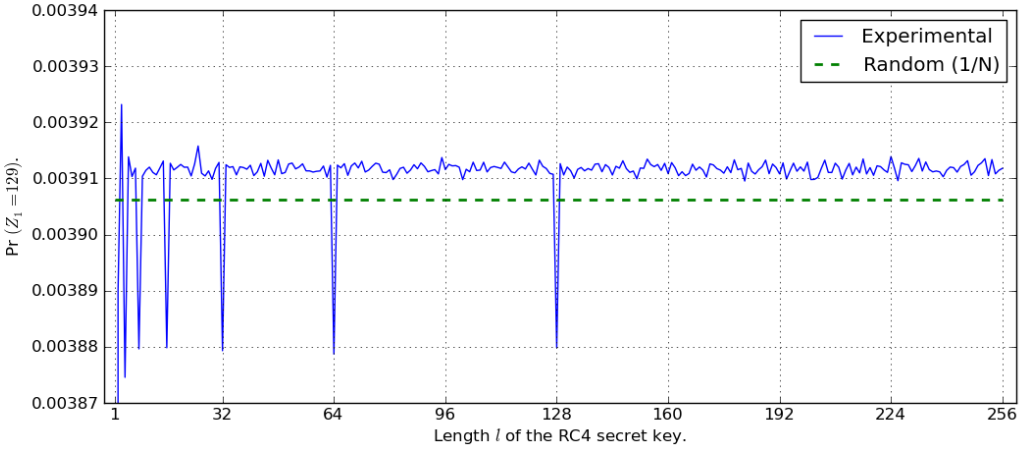

Figure 3.10: Bias in the event $(Z_1 = 129)$ for keylength $1 \leq l \leq 256$.

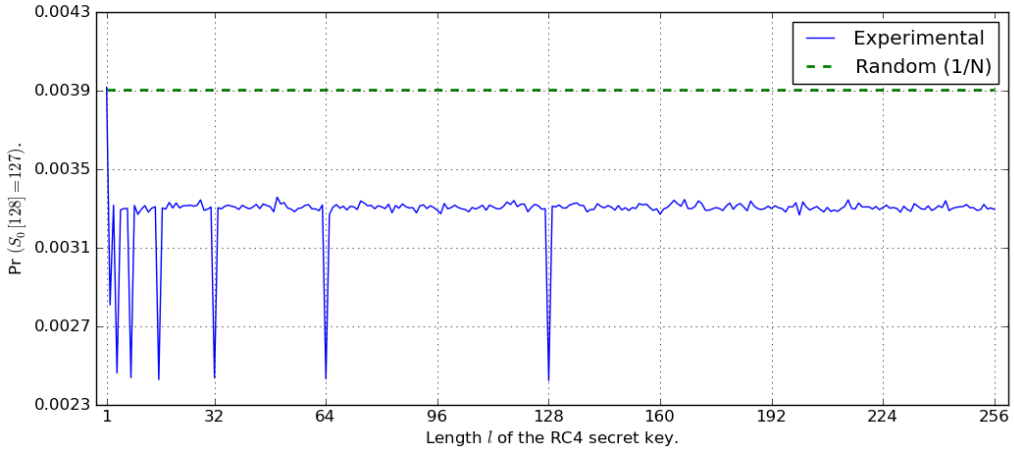

Figure 3.11: Bias in the event $(S_0[128] = 127)$ for keylength $1 \leq l \leq 256$.

3.3.1 Proof of anomaly in $(S_0[128] = 127)$

We will require the following technical results to prove the main theorem later.

Lemma 3.18. *In practical RC4 with $N = 256$, for $1 \leq r \leq N$,*

$$\Pr(S_{r-1}^K[r] = r) \approx 1/N + (1 - 1/N)^r.$$

Proof. We know that S_0^K is the identity permutation of $\{0, \ldots, N-1\}$, and thus $S_0^K[r] = r$. This value will remain at the same index till round $(r-1)$ if none of $j_1^K, j_2^K, \ldots, j_{r-1}^K$ touches the index r, which occurs with probability $(1 - 1/N)^{r-1}$, or otherwise due to random association, with probability $1/N$.

Hence, we get $\Pr(S_{r-1}^K[r] = r)$ approximately equal to

$$(1 - 1/N)^{r-1} \cdot 1 + (1 - (1 - 1/N)^{r-1}) \cdot (1/N) = 1/N + (1 - 1/N)^r.$$

Thus the result. □

Lemma 3.19. *In practical RC4 with $N = 256$,*

$$\Pr(S_{127}^K[128] = -K[128]) \approx 0.4/N,$$

if and only if l, the length of the secret key, is a non-trivial factor of $N = 256$.

Proof. Let us consider the following two paths.

Path 1. Consider $S_{127}^K[128] = 128$. In this case, we surely require $K[128] = -128 = 128$ (modulo $N = 256$). Now, if $l = 2, 4, \ldots, 128$, then $K[128] = K[0] = 128$. This implies $j_1^K = j_0^K + S_0^K[0] + K[0] = 0 + 0 + 128 = 128$, which in turn results in $S_1^K[0] = 128$ and $S_1^K[128] = 0$ after swap in the first round. As i^K does not touch index locations 0 or 128 during rounds 2 to 127, we can not have $S_{127}^K[128] = 128$, a contradiction. If l does not divide 128, then $K[128]$ may not be equal to $K[0]$, and in this case $S_{127}^K[128] = 128$ may occur with probability $1/N$.

Thus, $\Pr(S_{127}^K[128] = -K[128] \mid S_{127}^K[128] = 128) = 0$ if $l = 2, 4, \ldots, 128$, and $\Pr(S_{127}^K[128] = -K[128] \mid S_{127}^K[128] = 128) \approx 1/N$, otherwise.

Path 2. In case $S_{127}^K[128] \neq 128$, there is no special behavior dependent on the keylength l, and we may assume that $\Pr(S_{127}^K[128] = -K[128] \mid S_{127}^K[128] \neq 128) \approx 1/N$.

Combining the two paths, we get

$$\begin{aligned}
&\Pr(S_{127}^K[128] = -K[128]) \\
&= \Pr(S_{127}^K[128] = -K[128] \mid S_{127}^K[128] = 128) \cdot \Pr(S_{127}^K[128] = 128) \\
&\quad + \Pr(S_{127}^K[128] = -K[128] \mid S_{127}^K[128] \neq 128) \cdot \Pr(S_{127}^K[128] = 128) \\
&\approx 0 \cdot (156/N) + (1/N) \cdot (1 - 156/N) \approx 0.4/N,
\end{aligned}$$

if $l = 2, 4, \ldots, 128$, where $\Pr(S_{127}^K[128] = 128) \approx 156/N$ is by Lemma 3.18 with $r = 128$. For all other values of l, we get

$$\Pr(S_{127}^K[128] = -K[128]) \approx (1/N) \cdot (156/N) + (1/N) \cdot (1 - 156/N) = 1/N.$$

Integrating the two cases, we get the result. □

Theorem 3.20. *In practical RC4 with $N = 256$,*

$$\Pr(S_0[128] = S_N^K[128] = 127) \approx 0.63/N,$$

if and only if l, the length of the secret key, is a non-trivial factor of $N = 256$.

Proof. Let us first compute $\Pr(S_{128}^K[128] = 127)$, using the following paths.

Path 1. Consider $S_{127}^K[128] = -K[128]$. In this case, $j_{128} = j_{127} + S_{127}^K[128] + K[128] = j_{127}$. So, $S_{128}^K[128] = S_{127}^K[j_{128}] = S_{127}^K[j_{127}] = S_{126}^K[127]$. Now, by Lemma 3.18 with $r = 127$, we get $\Pr(S_{126}^K[127] = 127) \approx 156/N$. Thus, $\Pr(S_{128}^K[128] = 127 \mid S_{127}^K[128] = -K[128]) \approx 156/N$.

Path 2. Consider $S_{127}^K[128] \neq -K[128]$. In this case, $S_{128}^K[128] = S_{126}^K[X]$ for some $X \neq 127$. Thus by normalization over the probability values $\Pr(S_{126}^K[X] = 127)$ for $X \neq 127$, we get $\Pr(S_{128}^K[128] = 127 \mid S_{127}^K[128] \neq -K[128]) \approx (1 - 156/N)/(N - 1) \approx 0.4/N$.

Combining the two paths as above, we get

$$\begin{aligned}
&\Pr(S_{128}^K[128] = 127) \\
&= \Pr(S_{128}^K[128] = 127 \mid S_{127}^K[128] = -K[128]) \cdot \Pr(S_{127}^K[128] = -K[128]) \\
&\quad + \Pr(S_{128}^K[128] = 127 \mid S_{127}^K[128] \neq -K[128]) \cdot \Pr(S_{127}^K[128] \neq -K[128]) \\
&\approx (156/N) \cdot (0.4/N) + (0.4/N) \cdot (1 - 0.4/N) \approx 0.64/N,
\end{aligned}$$

if $l = 2, 4, \ldots, 128$. For all other values of l, we get $\Pr(S_{128}^K[128] = 127) \approx (156/N) \cdot (1/N) + (0.4/N) \cdot (1 - 1/N) \approx 1/N$. In both cases, the value of $\Pr(S_{127}^K[128] = -K[128])$ comes from Lemma 3.19.

Once we have $S_{128}^K[128] = 127$, we know that $S_0[128] = S_N^K[128] = 127$ if none of $j_{129}, \ldots, j_N$ touches the index 128. If otherwise $S_{128}^K[128] \neq 127$ and the

value 127 is in any index less than 128, then $S_N^K[128] \neq 127$. If $S_{128}^K[128] \neq 127$ and the value 127 is in any index I greater than 128, then $S_N^K[128] = 127$ may occur due to the following association.

> Indices $j_{129}, \ldots, j_{I-1}$ do not touch location I before $i = I$.
> When $i = I$, we have $j = 128$, so that the appropriate swap occurs.
> None of $j_{I+1}, \ldots, j_N$ touches location 128 after the previous event.

This path entails a approximate probability $(1/N) \cdot (1 - 1/N)^{127}$ for each I, and the total probability of the aforesaid association, over $I = 129, \ldots, 255$, becomes approximately $0.24/N$. Thus for $l = 2, 4, \ldots, 128$, we have

$$\begin{aligned}
&\Pr(S_N^K[128] = 127) \\
&= \Pr(S_{128}^K[128] = 127) \cdot (1 - 1/N)^{128} + \Pr(S_{128}^K[128] \neq 127) \cdot (0.24/N) \\
&\approx (0.64/N) \cdot (155/N) + (1 - 0.64/N) \cdot (0.24/N) \approx 0.63/N.
\end{aligned}$$

For other values of l, we get $\Pr(S_0[128] = S_N^K[128] = 127)$ following the distribution of $S_0[u] = v$ predicted by Mantin [100, 103]. Hence the 'anomaly'. □

The theoretical results regarding the anomaly in $(S_0[128] = 127)$, as above, produce a numerical value of $0.63/N = 0.002461$ for $N = 256$. This value closely matches with the experimental results, both from our own experiments (0.002453), as well as the value 0.002440 reported in the literature [119, 120].

This settles a long-standing mysterious issue in RC4 literature, and hints at the possibility that all 'anomalies' or deviations of probabilities in the distribution of S_0 from that predicted by Mantin [100], may actually result from intricate keylength dependencies.

3.3.2 Study of the bias in $(Z_1 = 129)$

Experimentally, we find that $\Pr(Z_1 = 129 \mid S_0[128] = 127) \approx 1/N - 0.5/N^2$ and $\Pr(Z_1 = 129 \mid S_0[128] \neq 127) \approx 1/N - 2/N^2$. Thus, using the anomaly,

one may estimate $\Pr(Z_1 = 129)$ as

$$\begin{aligned}
&\Pr(Z_1 = 129 \mid S_0[128] = 127) \Pr(S_0[128] = 127) \\
&+ \Pr(Z_1 = 129 \mid S_0[128] \neq 127) \Pr(S_0[128] \neq 127) \\
&\approx (1/N - 0.5/N^2) \cdot (0.63/N) + (1/N - 2/N^2) \cdot (1 - 0.63/N) \approx 1/N - 2/N^2.
\end{aligned}$$

However, it may be the case that the anomaly is not directly influencing the bias in $(Z_1 = 129)$. Investigation and proof of all causal paths towards proving the negative bias in $(Z_1 = 129)$ remains an interesting open question, which requires an independent rigorous analysis, beyond the scope of this thesis.

In summary, we attempted the proof of the bias in $(Z_1 = 129)$, but could not settle it completely. However, we discovered that this bias is a new 'keylength dependent' bias of RC4, which is prominent only for certain keylengths $l = 2, 4, 8, \ldots, 128$. In the process, we tried to relate it with the long-standing open issue of 'anomalies' in RC4 initial state, and could prove an important anomaly regarding the bias in $(S_0[128] = 127)$. Our work reveals that a thorough analysis of the 'anomaly pairs' of RC4 initial PRGA state S_0 is necessary, not only for their independent theoretical interest, but also to investigate their potential implications towards keystream biases.

Chapter 4

Biases Involving State Variables of RC4

In this chapter, we prove some empirically observed biases that involve the state variables i, j and S along with the output keystream Z. This chapter deals with the following problems in RC4 analysis, as mentioned earlier in Section 1.4 of Chapter 1.

Problem 1c. Prove all known significant biases of RC4 involving the state variables, as empirically observed in [136]. In addition, is it possible to identify and prove other interesting biases of similar nature?

Problem 1d. It seems that the index j exhibits certain non-random behavior in the initial rounds of RC4 PRGA. Is it possible to completely characterize the (non-)randomness of index j throughout RC4 PRGA?

Problems 1c is studied and solved in our papers [131, 132], and the study of Problem 1d is presented in our works [98, 132].

4.1 Proof of biases involving state variables

In connection with Problems 1c and 1d, we investigate some significant empirical biases discovered and reported by Sepehrdad, Vaudenay and Vuagnoux [136]. We provide theoretical justification only for the biases which are of the approximate order of $2/N$ or more, as in Table 4.1.

Table 4.1: Significant biases observed in [136] and proved in this chapter.

Type of Bias	Label as in [136]	Biased events observed in [136][a]	Probabilities reported in [136]
Bias at Specific Initial Rounds	"New_004"	$j_2 + S_2[j_2] = S_2[i_2] + Z_2$	$2/N$
	"New_noz_007"	$j_2 + S_2[j_2] = 6$	$2.37/N$
	"New_noz_009"	$j_2 + S_2[j_2] = S_2[i_2]$	$2/N$
	"New_noz_014"	$j_1 + S_1[i_1] = 2$	$1.94/N$
Bias at All Rounds (round-independent)	"New_noz_001"	$j_r + S_r[i_r] = i_r + S_r[j_r]$	$2/N$
	"New_noz_002"	$j_r + S_r[j_r] = i_r + S_r[i_r]$	$2/N$
Bias at All Initial Rounds, $1 \leq r \leq N-1$ (round-dependent)	"New_000"	$S_r[t_r] = t_r$	$1.9/N$ at $r = 3$
	"New_noz_004"	$S_r[i_r] = j_r$	$1.9/N$ at $r = 3$
	"New_noz_006"	$S_r[j_r] = i_r$	$2.34/N$ at $r = 3$

[a] Note that the authors of [136] denoted the PRGA variables by primed indices, but we do not use that notation.

4.1.1 Biases at specific initial rounds

In this section, we shall assume two main models to prove the results; that of uniform random keys with actual RC4 next-state-function for the evolution of S and i, j through KSA (we refer to this model as S_0 of RC4), and that of uniformly random initial permutation S_0 in PRGA (we refer to this model as random S_0). Each result specifies the model used to compute the values.

We first prove the bias labeled "New_noz_014" in [136, Figure 3 and Figure 4] and Table 4.1.

Theorem 4.1. *After the first round* $(r = 1)$ *of RC4 PRGA,*

$$\Pr(j_1 + S_1[i_1] = 2) = \Pr(S_0[1] = 1) + \sum_{X \neq 1} \Pr(S_0[X] = 2 - X \ \wedge \ S_0[1] = X).$$

Proof. We have $j_1 + S_1[i_1] = S_0[1] + S_0[j_1] = S_0[1] + S_0[S_0[1]]$. We compute the desired probability using the following two conditional paths depending on the value of $j_1 = S_0[1]$:

$$\begin{aligned}
&\Pr(j_1 + S_1[i_1] = 2) \\
&= \Pr(S_0[1] + S_0[S_0[1]] = 2 \mid S_0[1] = 1) \cdot \Pr(S_0[1] = 1) \\
&\quad + \sum_{X \neq 1} \Pr(S_0[1] + S_0[S_0[1]] = 2 \mid S_0[1] = X) \cdot \Pr(S_0[1] = X) \\
&= \Pr(1 + S_0[1] = 2 \mid S_0[1] = 1) \cdot \Pr(S_0[1] = 1) \\
&\quad + \sum_{X \neq 1} \Pr(X + S_0[X] = 2 \mid S_0[1] = X) \cdot \Pr(S_0[1] = X) \\
&= 1 \cdot \Pr(S_0[1] = 1) + \sum_{X \neq 1} \Pr(S_0[X] = 2 - X \ \wedge \ S_0[1] = X).
\end{aligned}$$

Hence the result. □

If we consider the RC4 permutation after the KSA, the probabilities involving S_0 in the expression for $\Pr(j_1 + S_1[i_1] = 2)$ should be evaluated using Proposition 3.1 and the joint probability should be estimated in the same manner as in Section 5.1.3, giving a total probability of approximately $1.937/N$ for $N = 256$. This closely matches the observed value $1.94/N$. If we assume that RC4 PRGA starts with a random initial permutation S_0, the probability turns out to be approximately $2/N - 1/N^2 \approx 1.996/N$ for $N = 256$, i.e., almost

twice that of a random occurrence.

Next, we prove the biases "New_noz_007", "New_noz_009" and "New_004", as labeled in [136] and Table 4.1.

Theorem 4.2. *After the second round* $(r = 2)$ *of PRGA, the following probability relations hold between index* j_2 *and state variables* $S_2[i_2], S_2[j_2]$*:*

$$\Pr(j_2 + S_2[j_2] = 6) \approx \Pr(S_0[1] = 2) + \sum_{X \text{ even}, X \neq 2} (2/N) \cdot \Pr(S_0[1] = X), \quad (4.1)$$

$$\Pr(j_2 + S_2[j_2] = S_2[i_2]) \approx 2/N - 1/N^2, \quad (4.2)$$

$$\Pr(j_2 + S_2[j_2] = S_2[i_2] + Z_2) \approx 2/N - 1/N^2. \quad (4.3)$$

Proof. We have $j_2 + S_2[j_2] = (j_1 + S_1[i_2]) + S_1[i_2] = S_0[1] + 2 \cdot S_1[2]$ in RC4 PRGA. Now for Equation (4.1), we have the following paths depending on the value of $j_1 = S_0[1]$:

$$\begin{aligned}\Pr(j_2 + S_2[j_2] = 6) = {} & \Pr(S_0[1] + 2 \cdot S_1[2] = 6 \mid S_0[1] = 2) \cdot \Pr(S_0[1] = 2) \\ & + \sum_{X \neq 2} \Pr(S_0[1] + 2 \cdot S_1[2] = 6 \mid S_0[1] = X) \cdot \Pr(S_0[1] = X).\end{aligned}$$

We explore the conditional events in each of the above paths as follows:

$$\begin{aligned} S_0[1] = 2 \;\Rightarrow\; & S_0[1] + 2 \cdot S_1[2] = 2 + 2 \cdot S_1[j_1] \\ & = 2 + 2 \cdot S_0[i_1] = 2 + 2 \cdot S_0[1] = 6, \\ S_0[1] = X \neq 2 \;\Rightarrow\; & S_0[1] + 2 \cdot S_1[2] = X + 2 \cdot S_1[2]. \end{aligned}$$

To satisfy $X + 2 \cdot S_1[2] = 6$ in the second path, the value of X must be even and for each such value of X, the variable $S_1[2]$ can take two different values, namely $(3 + N/2 - X/2)$ and $(3 + N - X/2)$ modulo N. Thus, we have

$$\Pr(j_2 + S_2[j_2] = 6) = 1 \cdot \Pr(S_0[1] = 2) + \sum_{X \text{ even}, X \neq 2} (2/N) \cdot \Pr(S_0[1] = X).$$

In case of Equation (4.2), we have the following conditional paths depending

on the value of $S_1[2]$:

$$\begin{aligned}
&\Pr(j_2 + S_2[j_2] = S_2[i_2]) \\
&= \Pr(S_0[1] + 2 \cdot S_1[2] = S_1[j_2] \mid S_1[2] = 0) \cdot \Pr(S_1[2] = 0) \\
&\quad + \Pr(S_0[1] + 2 \cdot S_1[2] = S_1[j_2] \mid S_1[2] \neq 0) \cdot \Pr(S_1[2] \neq 0).
\end{aligned}$$

In the first case, the condition holds with probability 1, since $S_1[2] = 0$ implies

$$\begin{aligned}
&S_0[1] + 2 \cdot S_1[2] = S_0[1], \text{ and} \\
&S_1[j_2] = S_1[S_0[1] + S_1[2]] = S_1[S_0[1]] = S_1[j_1] = S_0[i_1] = S_0[1].
\end{aligned}$$

For all other cases in the second path, with $S_1[2] = X \neq 0$, we can assume the condition to hold with probability approximately $1/N$. Thus, we have:

$$\Pr(j_2 + S_2[j_2] = S_2[i_2]) \approx 1 \cdot (1/N) + (1/N) \cdot (1 - 1/N) = 2/N - 1/N^2.$$

For Equation (4.3), the condition is almost identical to the condition of Equation (4.2) apart from the inclusion of Z_2. However, our first path $S_1[2] = 0$ gives $\Pr(Z_2 = 0 \mid S_1[2] = 0) = 1$ (as in [103]), which implies the following:

$$\Pr(j_2 + S_2[j_2] = S_2[i_2] + Z_2 \mid S_1[2] = 0) = \Pr(j_2 + S_2[j_2] = S_2[i_2] \mid S_1[2] = 0).$$

In all other cases with $S_1[2] \neq 0$, we assume the conditions to match uniformly at random. Therefore:

$$\Pr(j_2 + S_2[j_2] = S_2[i_2] + Z_2) \approx (1/N) \cdot 1 + (1 - 1/N) \cdot (1/N) = 2/N - 1/N^2. \tag{4.4}$$

Hence the result. □

In case of Equation (4.1), if we assume S_0 to be the initial state for RC4 PRGA, and substitute all probabilities involving S_0 using Proposition 3.1, we get the total probability equal to $2.36/N$ for $N = 256$. This value closely matches with the observed probability $2.37/N$. If we assume S_0 to be a random permutation in (4.1), we get probability $2/N - 2/N^2 \approx 1.992/N$ for $N = 256$. The theoretical results are summarized in Table 4.2 along with the experimentally observed probabilities from [136].

Table 4.2: Theoretical and observed biases at specific initial rounds of RC4 PRGA.

Label [136]	Event	Observed Probability (reported in [136])	Theoretical Probability (for $N = 256$)	
			S_0 of RC4	Random S_0
"New_noz_014"	$j_1 + S_1[i_1] = 2$	$1.94/N$	$1.937/N$	$1.996/N$
"New_noz_007"	$j_2 + S_2[j_2] = 6$	$2.37/N$	$2.363/N$	$1.992/N$
"New_noz_009"	$j_2 + S_2[j_2] = S_2[i_2]$	$2/N$	$1.996/N$	$1.996/N$
"New_noz_004"	$j_2 + S_2[j_2] = S_2[i_2] + Z_2$	$2/N$	$1.996/N$	$1.996/N$

4.1.2 Round-independent biases at all initial rounds

In this section, we turn our attention to the biases labeled "New_noz_001" and "New_noz_002." In [136] it was observed that both of these biases exist for all initial rounds ($1 \leq r \leq N-1$) of RC4 PRGA. In Theorem 4.3 below, we prove a more general result. We show that actually these biases do not change with r and they continue to persist at the same order of $2/N$ at any arbitrary round of PRGA. Thus, the probabilities for "New_noz_001" and "New_noz_002" from [136] are special cases (for $1 \leq r \leq N-1$) of Theorem 4.3.

We shall assume the probabilistic model of uniformly random permutation S anytime during PRGA, where the pointer j is assumed to be uniformly random and independent of the permutation.

Theorem 4.3. *At any round $r \geq 1$ of RC4 PRGA, the following two relations hold between the indices i_r, j_r and the state variables $S_r[i_r], S_r[j_r]$:*

$$\Pr(j_r + S_r[j_r] = i_r + S_r[i_r]) \approx 2/N, \tag{4.5}$$

$$\Pr(j_r + S_r[i_r] = i_r + S_r[j_r]) \approx 2/N. \tag{4.6}$$

Proof. We denote the events as $E_1 : (j_r + S_r[j_r] = i_r + S_r[i_r])$ and $E_2 : (j_r + S_r[i_r] = i_r + S_r[j_r])$. For both the events, we shall take the conditional paths as follows for computing the probabilities:

$$\Pr(E_1) = \Pr(E_1 \mid i_r = j_r) \cdot \Pr(i_r = j_r) + \Pr(E_1 \mid i_r \neq j_r) \cdot \Pr(i_r \neq j_r),$$
$$\Pr(E_2) = \Pr(E_2 \mid i_r = j_r) \cdot \Pr(i_r = j_r) + \Pr(E_2 \mid i_r \neq j_r) \cdot \Pr(i_r \neq j_r).$$

We have $\Pr(i_r = j_r) \approx 1/N$ and $\Pr(E_1 \mid i_r = j_r) = \Pr(E_2 \mid i_r = j_r) = 1$. In the case where $i_r \neq j_r$, we have $S_r[j_r] \neq S_r[i_r]$, as S_r is a permutation. Thus in case $i_r \neq j_r$, the values of $S_r[i_r]$ and $S_r[j_r]$ can be chosen in $N(N-1)$ ways (drawing from a permutation without replacement) to satisfy the events E_1, E_2. This gives the total probability for each event E_1, E_2 as

$$\begin{aligned} &\Pr(E_1) \approx \Pr(E_2) \\ &\approx 1 \cdot \frac{1}{N} + \sum_{j_r \neq i_r} \frac{1}{N(N-1)} = \frac{1}{N} + (N-1) \cdot \frac{1}{N(N-1)} = \frac{2}{N}. \end{aligned} \tag{4.7}$$

Hence the result. □

Our theoretical results match the probabilities reported in [136, Figure 2] for the initial rounds $1 \leq r \leq N-1$. One may note that the biases in Theorem 4.3 look somewhat similar to Jenkin's correlations [75]:

$$\Pr(Z_r = j_r - S_r[i_r]) \approx 2/N \quad \text{and} \quad \Pr(Z_r = i_r - S_r[j_r]) \approx 2/N.$$

However, the biases proved in Theorem 4.3 do not contain the keystream byte Z_r, and one may check that the results do not follow directly from Jenkin's correlations [75] either.

4.1.3 Round-dependent biases at all initial rounds

Next, we consider the biases that are labeled as "New_000", "New_noz_004" and "New_noz_006" in [136, Figure 2]. We prove the biases for rounds 3 to 255 in RC4 PRGA, and we show that all of these decrease in magnitude with increase in r, as observed experimentally in [136].

In this section, we shall assume the probabilistic model of uniform random keys to prove the results. We assume actual RC4 next-state-function for the evolution of S and i, j, and no randomness assumptions are made on the initial state S_0 of PRGA.

The bias labeled "New_noz_006" in [136] can be derived as a corollary to Theorem 3.3 as follows.

Corollary 4.4. *For PRGA rounds* $3 \leq r \leq N-1$,

$$\Pr(S_r[j_r] = i_r) \approx \Pr(S_1[r] = r) \left(1 - \frac{1}{N}\right)^{r-2} + \sum_{t=2}^{r-1} \sum_{w=0}^{r-t} \frac{\Pr(S_1[t] = r)}{w! \cdot N} \left(\frac{r-t-1}{N}\right)^{w} \left(1 - \frac{1}{N}\right)^{r-3-w}.$$

Proof. $S_r[j_r]$ is assigned the value at $S_{r-1}[r]$ due to the swap in round r. Hence substituting $u = r$ and $v = i_r = r$ in Theorem 3.3, we get the result. □

In Figure 4.1, we illustrate the experimental observations (each data point represents the average obtained from over 100 million experimental runs with 16-byte key in each case) and the theoretical values for the distribution of

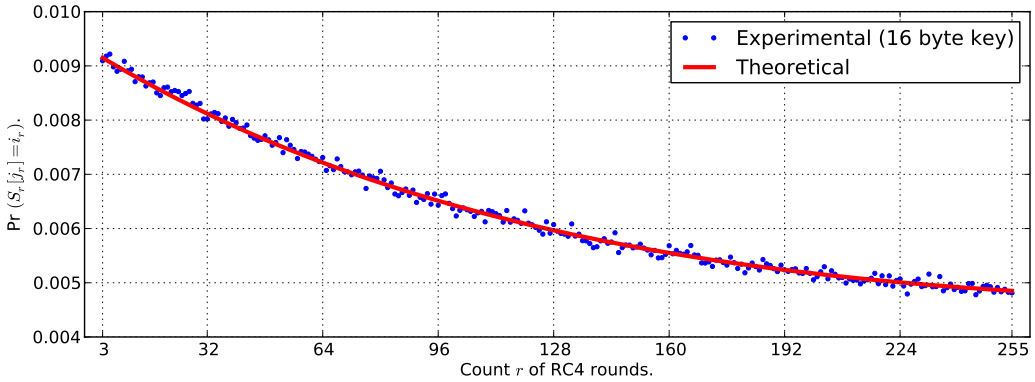

Figure 4.1: Distribution of $\Pr(S_r[j_r] = i_r)$ for initial rounds $3 \leq r \leq 255$ of RC4 PRGA.

$\Pr(S_r[j_r] = i_r)$ over the initial rounds $3 \leq r \leq 255$ of RC4 PRGA. It is evident that our theoretical formula, as derived in Corollary 4.4, matches the experimental observations.

Next we take a look at the other two round-dependent biases of RC4, observed in [136]. We state the related result in Theorem 4.5, corresponding to observations "New_noz_004" and "New_000".

Theorem 4.5. *For PRGA rounds* $3 \leq r \leq N-1$,

$$\begin{aligned}
&\Pr(S_r[i_r] = j_r) \approx \Pr(S_r[t_r] = t_r) \\
&\approx \frac{r}{N^2} + \sum_{X=r}^{N-1} \frac{1}{N} \left(\Pr(S_1[X] = X) \left(1 - \frac{1}{N}\right)^{r-2} \right. \\
&\quad \left. + \sum_{u=2}^{r-1} \sum_{w=0}^{r-u} \frac{\Pr(S_1[u] = r)}{w! \cdot N} \left(\frac{r-u-1}{N}\right)^{w} \left(1 - \frac{1}{N}\right)^{r-3-w} \right).
\end{aligned}$$

Proof. We can write the two events under consideration as $E_3 : (S_{r-1}[j_r] = j_r)$ and $E_4 : (S_r[t_r] = t_r)$, where j_r and t_r can be considered as pseudorandom variables for all $3 \leq r \leq N-1$. We consider the following conditional paths for the first event E_3, depending on the range of values j_r may take:

$$\Pr(E_3) = \sum_{X=0}^{r-1} \Pr(E_3 | j_r = X) \cdot \Pr(j_r = X) + \sum_{X=r}^{N-1} \Pr(E_3 | j_r = X) \cdot \Pr(j_r = X).$$

Case I. In this case, we assume that j_r takes a value X between 0 and $r-1$. Each position in this range is touched by index i, and may also be touched

by index j. Thus, irrespective of any initial condition, we may assume that $\Pr(E_3 \mid j_r = X) \approx 1/N$ in this case. Hence, this part contributes:

$$\sum_{X=0}^{r-1} \Pr(E_3 \mid j_r = X) \cdot \Pr(j_r = X) \approx \sum_{X=0}^{r-1} \frac{1}{N} \cdot \frac{1}{N} = \frac{r}{N^2}.$$

Case II. Here we suppose that j_r assumes a value $r \leq X \leq N-1$. In this case, the probability calculation can be split into two paths, as follows:

$$\begin{aligned}\Pr(E_3 \mid j_r = X) = \Pr(E_3 \mid j_r = X \wedge S_1[X] = X) \cdot \Pr(S_1[X] = X) \\ + \Pr(E_3 \mid j_r = X \wedge S_1[X] \neq X) \cdot \Pr(S_1[X] \neq X).\end{aligned}$$

If $S_1[X] = X$, similarly to the logic in Theorem 3.3, we get the following:

$$\Pr(E_3 \mid j_r = X \wedge S_1[X] = X) \cdot \Pr(S_1[X] = X) \approx \Pr(S_1[X] = X) \left(1 - \frac{1}{N}\right)^{r-2}.$$

If we suppose that $S_1[u] = X$ for some $u \neq X$, then one may note the following two sub-cases:

- Sub-case $2 \leq u \leq r-1$: The probability for this path is similar to that in the proof of Theorem 3.3:

$$\sum_{u=2}^{r-1} \sum_{w=0}^{r-u} \frac{\Pr(S_1[u] = r)}{w! \cdot N} \left(\frac{r-u-1}{N}\right)^w \left(1 - \frac{1}{N}\right)^{r-3-w}.$$

- Sub-case $r \leq u \leq N-1$: In this case the value X will always be behind the position of $i_r = r$, whereas $X > r$ as per assumption, i.e., the value X can never reach index position X from initial position u. Thus the probability is 0 in this case.

Assuming $\Pr(j_r = X) = 1/N$ for all X, and combining all contributions from the above-mentioned cases, we get the value of $\Pr(S_{r-1}[j_r] = j_r) = \Pr(S_r[i_r] = j_r)$, as desired.

In case of $\Pr(S_r[t_r] = t_r)$, t_r is a random variable just like j_r, and may take all values from 0 to $N-1$ with approximately the same probability $1/N$. Thus we can approximate $\Pr(S_r[t_r] = t_r) \approx \Pr(S_{r-1}[j_r] = j_r)$ to obtain the desired expression. □

Remark 4.6. *The approximation* $\Pr(S_r[t_r] = t_r) \approx \Pr(S_{r-1}[j_r] = j_r)$, *as in Theorem 4.5, is particularly close for higher values of* r *because the effect of a single state change from* S_{r-1} *to* S_r *is low in such a case. For smaller values of* r, *it is more accurate to approximate* $\Pr(S_{r-1}[t_r] = t_r) \approx \Pr(S_{r-1}[j_r] = j_r)$ *and critically analyze the effect of the* r*-th round of PRGA thereafter.*

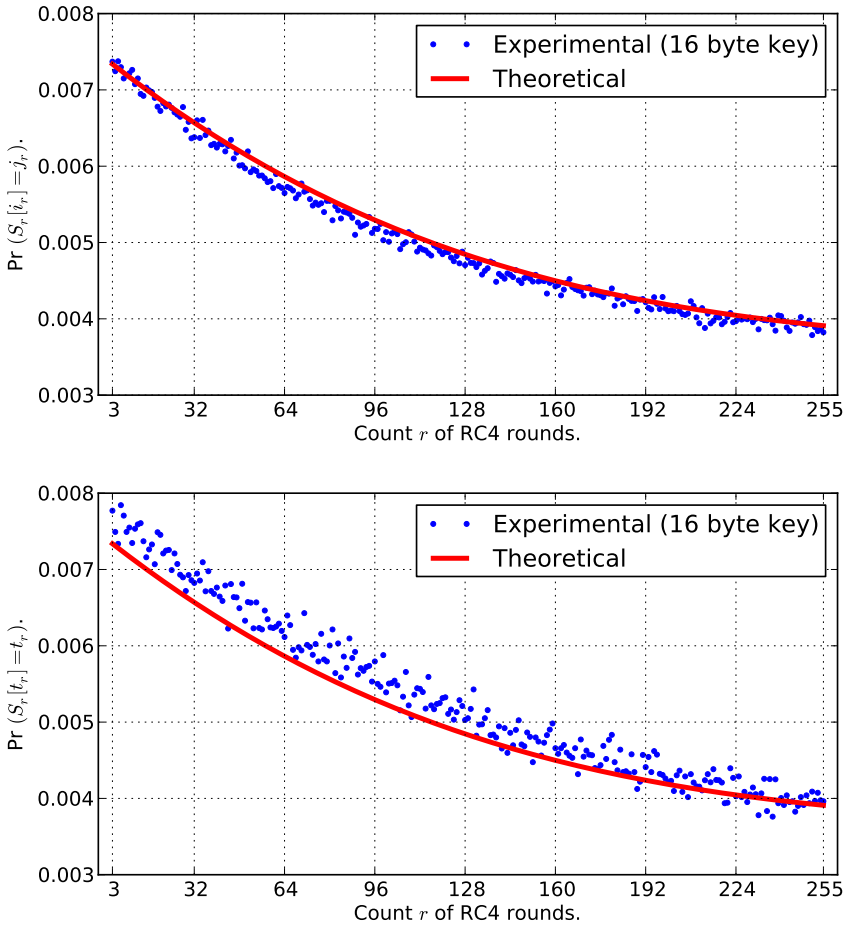

Figure 4.2: Distributions of $\Pr(S_r[i_r] = j_r)$ and $\Pr(S_r[t_r] = t_r)$ for initial rounds $3 \leq r \leq 255$ of RC4 PRGA.

In Figure 4.2, we show the experimental observations (averages taken over 100 million runs with 16-byte key) and the theoretical values for the distributions of $\Pr(S_r[i_r] = j_r)$ and $\Pr(S_r[t_r] = t_r)$ over the initial rounds $3 \leq r \leq 255$ of RC4 PRGA. It is evident that our theoretical formulae closely match with the experimental observations in both the cases.

4.2 (Non-)Randomness of j at initial rounds

Two indices, i and j, are used in RC4 PRGA – the first is deterministic and the second one is pseudorandom. Index j depends on the values of i and $S[i]$ simultaneously, and the pseudorandomness of the permutation S causes the pseudorandomness in j. In this section, we attempt to analyze the pseudorandom behavior of j more clearly.

In this section, we shall assume the probabilistic model of uniform random keys to prove the results. We assume actual RC4 next-state-function for the evolution of S and i, j, and no randomness assumptions are made on the initial state S_0 of PRGA.

In RC4 PRGA, we know that for $r \geq 1$, $i_r = r \bmod N$ and $j_r = j_{r-1} + S_{r-1}[i_r]$, starting with $j_0 = 0$. Thus, we can recursively write the values of j at different rounds $1 \leq r \leq N-1$:

$$
\begin{aligned}
&j_0 = 0,\\
&j_1 = S_0[1],\\
&j_2 = S_0[1] + S_1[2],\\
&\cdots\\
&j_r = j_{r-1} + S_{r-1}[i_r] = S_0[1] + S_1[2] + \cdots + S_{r-1}[r] = \sum_{x=1}^{r} S_{x-1}[x].
\end{aligned}
$$

4.2.1 Non-randomness of j_1

In the first round of PRGA, $j_1 = S_0[1]$ follows a probability distribution which is determined by S_0. According to Proposition 3.1, we have:

$$
\begin{aligned}
&\Pr(j_1 = v) = \Pr(S_0[1] = v)\\
&= \begin{cases}
\dfrac{1}{N}, & \text{if } v = 0;\\
\dfrac{1}{N}\left(\dfrac{N-1}{N} + \dfrac{1}{N}\left(\dfrac{N-1}{N}\right)^{N-2}\right), & \text{if } v = 1;\\
\dfrac{1}{N}\left(\left(\dfrac{N-1}{N}\right)^{N-2} + \left(\dfrac{N-1}{N}\right)^{v}\right), & \text{if } v > 1.
\end{cases}
\end{aligned}
$$

This clearly tells us that j_1 is *not* random. This is also portrayed in Figure 4.3.

4.2.2 Non-randomness of j_2

In the second round of PRGA, however, we have $j_2 = S_0[1] + S_1[2]$, which demonstrates better randomness, as per the following discussion. We have:

$$\Pr(j_2 = v) = \Pr(S_0[1] + S_1[2] = v) = \sum_{w=0}^{N-1} \Pr(S_0[1] = w \wedge S_1[2] = v - w). \quad (4.8)$$

The following cases may arise with respect to Equation (4.8).

- Case I: Suppose that $j_1 = S_0[1] = w = 2$. Then, $S_1[i_2] = S_1[2] = S_1[j_1] = S_0[i_1] = S_0[1] = 2$. In this case, we have:

$$\Pr(j_2 = v) = \begin{cases} \Pr(S_0[1] = 2), & \text{if } v = 4; \\ 0, & \text{otherwise.} \end{cases}$$

- Case II: Suppose that $j_1 = S_0[1] = w \neq 2$. Then $S_0[2]$ will not get swapped in the first round, and hence $S_1[2] = S_0[2]$. In this case, $\Pr(S_0[1] = w \wedge S_1[2] = v - w) = \Pr(S_0[1] = w \wedge S_0[2] = v - w)$.

We substitute the results obtained from these cases to Equation (4.8) to obtain:

$$\Pr(j_2 = v) = \begin{cases} \Pr(S_0[1] = 2) & \\ \quad + \sum_{w \neq 2} \Pr(S_0[1] = w \ \wedge \ S_0[2] = v - w), & \text{if } v = 4; \\ \sum_{w \neq 2} \Pr(S_0[1] = w \ \wedge \ S_0[2] = v - w), & \text{if } v \neq 4. \end{cases} \quad (4.9)$$

Equation (4.9) completely specifies the exact probability distribution of j_2, where the exact values of the probabilities $\Pr(S_0[x] = y)$ can be substituted from Proposition 3.1 with the adjustment as in Section 5.1.3 for estimating the joint probabilities. However, the expression suffices to exhibit the non-randomness of j_2 in the RC4 PRGA, having a large bias for $v = 4$. We found that the theoretical probabilities from Equation (4.9) match almost exactly with the experimental data plotted in Figure 4.3. For the sake of clarity, we do not show the theoretical curve in Figure 4.3.

4.2.3 Randomness of j_r for $r \geq 3$

It is possible to compute the explicit probability distributions of $j_r = \sum_{x=1}^{r} S_{x-1}[x]$ for $3 \leq r \leq 255$ as well. We do not present the complicated expressions for $\Pr(j_r = v)$ for $r \geq 3$ here, but it turns out that j_r becomes closer to be random as r increases.

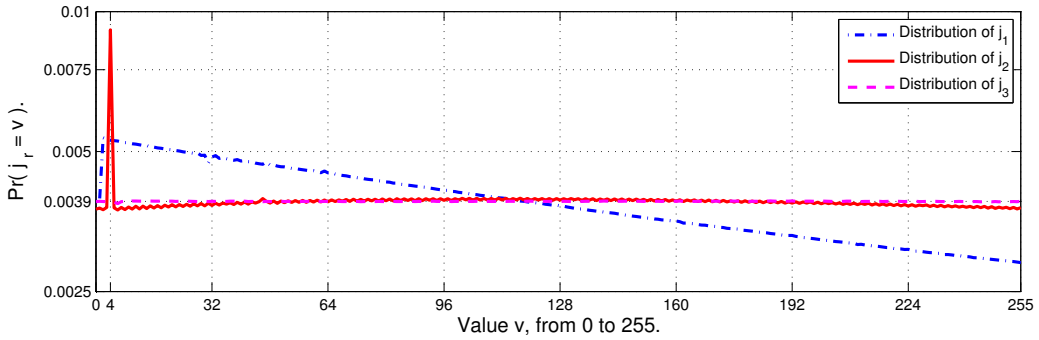

Figure 4.3: Probability distribution of j_r for $1 \leq r \leq 3$.

The probability distributions of j_1, j_2 and j_3 are shown in Figure 4.3, where the experiments have been run over 1 billion trials of RC4 PRGA, with randomly generated keys of size 16 bytes. One may note that the randomness in j_2 is more than that of j_1 (apart from the case $v = 4$), and j_3 is almost uniformly random. This trend continues for the later rounds of PRGA as well. However, we do not plot the graphs for the probability distributions of j_r with $r \geq 4$, as these distributions are almost identical to that of j_3, i.e., almost uniformly random in behavior.

4.2.4 Correlation between Z_2 and $S_2[2]$

We now explore the bias in $(j_2 = 4)$ more deeply and establish a correlation between the state S_2 and the keystream. Let us first evaluate $\Pr(j_2 = 4)$:

$$\Pr(j_2 = 4) = \Pr(S_0[1] = 2) + \sum_{w \neq 2} \Pr(S_0[1] = w \wedge S_0[2] = 4 - w)$$
$$= \frac{1}{N}\left[\left(\frac{N-1}{N}\right)^{N-2} + \left(\frac{N-1}{N}\right)^{2}\right] + \sum_{w \neq 2} \Pr(S_0[1] = w \wedge S_0[2] = 4 - w).$$

Following Proposition 3.1 and the estimation of joint probabilities as in Section 5.1.3, the sum in the above expression evaluates approximately to $0.965268/N$ for $N = 256$. Thus, we get:

$$\Pr(j_2 = 4) \approx \frac{1}{N}\left[\left(\frac{N-1}{N}\right)^{N-2} + \left(\frac{N-1}{N}\right)^{2}\right] + \frac{0.965268}{N} \approx \frac{7/3}{N}.$$

This closely matches with our experimental observation, as depicted in Figure 4.3. To exploit this bias in $(j_2 = 4)$, we focus on the event $(S_2[i_2] = 4 - Z_2)$ or $(S_2[2] = 4 - Z_2)$, and prove the following.

Theorem 4.7. *After the second round of RC4 PRGA with $N = 256$,*

$$\Pr\left(S_2[2] = 4 - Z_2\right) \approx \frac{1}{N} + \frac{4/3}{N^2}.$$

Proof. We can write Z_2 in terms of the state variables as follows:

$$Z_2 = S_2[S_2[i_2] + S_2[j_2]] = S_2[S_1[j_2] + S_1[i_2]] = S_2[S_1[j_2] + S_1[2]].$$

Thus, we can write the probability of the target event $(S_2[2] = 4 - Z_2)$ as

$$\begin{aligned}
\Pr(S_2[2] = 4 - Z_2) &= \Pr(S_2[i_2] = 4 - S_2[S_1[j_2] + S_1[2]]) \\
&= \Pr(S_1[j_2] + S_2[S_1[j_2] + S_1[2]] = 4) \\
&= \Pr(S_1[j_2] + S_2[S_1[j_2] + S_1[2]] = 4 \ \wedge \ j_2 = 4) \\
&+ \Pr(S_1[j_2] + S_2[S_1[j_2] + S_1[2]] = 4 \ \wedge \ j_2 \neq 4).
\end{aligned}$$

Computing the first term: The probability for the first event can be calculated as follows:

$$\begin{aligned}
&\Pr(S_1[j_2] + S_2[S_1[j_2] + S_1[2]] = 4 \ \wedge \ j_2 = 4) \\
&= \Pr(S_1[4] + S_2[S_1[4] + S_1[2]] = 4 \ \wedge \ j_2 = 4) \\
&= \sum_{y=0}^{N-1} \Pr\left(S_1[4] + S_2[y] = 4 \ \wedge \ S_1[4] + S_1[2] = y \ \wedge \ j_2 = 4\right) \\
&= \Pr(j_2 = 4) \cdot \sum_{y=0}^{N-1} \Pr\left(S_1[4] + S_2[y] = 4 \ \wedge \ S_1[4] + S_1[2] = y\right).
\end{aligned}$$

In the last expression, the values taken from S_1 are independent of the value

of j_2, and thus the events $(S_1[4] + S_2[y] = 4)$ and $(S_1[4] + S_1[2] = y)$ are both independent of the event $(j_2 = 4)$. Also, if $y = 4$, we obtain $S_1[4] + S_2[y] = S_1[4] + S_2[4] = S_1[4] + S_2[j_2] = S_1[4] + S_1[i_2] = S_1[4] + S_1[2]$, which results in the events $(S_1[4] + S_2[y] = 4)$ and $(S_1[4] + S_1[2] = y)$ being identical. In all other cases, we have $S_1[4] + S_2[y] \neq S_1[4] + S_1[2]$ and thus the values are chosen distinctly independent at random. Hence, we obtain:

$$\Pr(S_1[4] + S_2[y] = 4 \wedge S_1[4] + S_1[2] = y) = \begin{cases} \frac{1}{N}, & \text{if } y = 4; \\ \frac{1}{N(N-1)}, & \text{if } y \neq 4. \end{cases}$$

Thus, the probability $\Pr(S_1[j_2] + S_2[S_1[j_2] + S_1[2]] = 4 \ \wedge \ j_2 = 4)$ for the first event turns out to be:

$$\Pr(j_2 = 4) \cdot \left(\frac{1}{N} + \sum_{y \neq 4} \frac{1}{N(N-1)} \right) = \frac{7/3}{N} \cdot \left(\frac{1}{N} + \frac{N-1}{N(N-1)} \right) = \frac{7/3}{N} \cdot \frac{2}{N}.$$

Computing the second term: The probability calculation follows a similar path:

$$\begin{aligned} &\Pr(S_1[j_2] + S_2[S_1[j_2] + S_1[2]] = 4 \ \wedge \ j_2 \neq 4) \\ &= \sum_{x \neq 4} \Pr(S_1[x] + S_2[S_1[x] + S_1[2]] = 4 \ \wedge \ j_2 = x) \\ &= \sum_{x \neq 4} \sum_{y=0}^{N-1} \Pr(S_1[x] + S_2[y] = 4 \ \wedge \ S_1[x] + S_1[2] = y \ \wedge \ j_2 = x). \end{aligned}$$

The case $y = x$ poses an interesting situation. On the one hand, we obtain $S_1[x]+S_2[y] = S_1[x]+S_2[x] = S_1[x]+S_2[j_2] = S_1[x]+S_1[i_2] = S_1[x]+S_1[2] = 4$, while on the other hand, we get $S_1[x] + S_1[2] = x \neq 4$. We rule out this case to get $\Pr(S_1[j_2] + S_2[S_1[j_2] + S_1[2]] = 4 \wedge j_2 \neq 4)$:

$$\sum_{x \neq 4} \sum_{y \neq x} \Pr(S_1[x] + S_2[y] = 4 \ \wedge \ S_1[x] + S_1[2] = y) \cdot \Pr(j_2 = x).$$

As before, the values taken from S_1 are independent of the value of j_2, and thus the events $(S_1[x] + S_2[y] = 4)$ and $(S_1[x] + S_1[2] = y)$ are both independent of the event $(j_2 = x)$.

If $y = 4$, we have $S_1[x] + S_2[4] = 4$, while $S_1[x] + S_1[2] = 4$. One may note

that $S_1[4]$ does not get swapped to obtain S_2, as $i_2 = 2$ and $j_2 = x \neq 4$. Thus, $S_2[4] = S_1[4]$ and we get $S_1[x] + S_1[4] = 4$ and $S_1[x] + S_1[2] = 4$. This indicates $S_1[4] = S_1[2]$, which is impossible as S_1 is a permutation. All other cases ($y \neq 4$) deal with two distinct locations of the permutation S_1. Therefore,

$$\Pr(S_1[x] + S_2[y] = 4 \ \wedge \ S_1[x] + S_1[2] = y) = \begin{cases} 0, & \text{if } y = 4; \\ \frac{1}{N(N-1)}, & \text{otherwise.} \end{cases}$$

Thus, the probability $\Pr(S_1[j_2] + S_2[S_1[j_2] + S_1[2]] = 4 \wedge j_2 \neq 4)$ of the second event turns out to be:

$$\begin{aligned} &\sum_{x \neq 4} \Pr(j_2 = x) \cdot \left(0 + \sum_{y \neq x,4} \frac{1}{N(N-1)} \right) \\ &= \frac{N-2}{N(N-1)} \cdot \sum_{x \neq 4} \Pr(j_2 = x) = \frac{N-2}{N(N-1)} \cdot \left(1 - \frac{7/3}{N^2} \right). \end{aligned}$$

Calculation for $\Pr(S_2[2] = 4 - Z_2)$: Combining the probabilities for the first and second events, we get the following:

$$\Pr(S_2[2] = 4 - Z_2) = \frac{7/3}{N^2} \cdot \frac{2}{N} + \frac{N-2}{N(N-1)} \cdot \left(1 - \frac{7/3}{N^2} \right) \approx \frac{1}{N} + \frac{4/3}{N^2}. \quad (4.10)$$

Hence the result. □

This establishes a correlation between the state byte $S_2[2]$ and the keystream byte Z_2. For $N = 256$, the result supports our experimental data generated from 1 billion runs of RC4 with randomly selected 16-byte keys.

4.3 Long-term glimpse correlation in RC4

In 1996, Jenkins [75] pointed out that the RC4 keystream provides a glimpse of the RC4 state, now known as Glimpse theorem or Jenkins' correlation.

Theorem 4.8 (Glimpse theorem)**.** *After the r-th round of RC4 PRGA, for $r \geq 1$, we have* $\Pr(S_r[j_r] = i_r - Z_r) = \Pr(S_r[i_r] = j_r - Z_r) \approx 2/N$.

This glimpse correlation can be observed at any point of the RC4 keystream.

In Asiacrypt 2005, Mantin [102] has also explored a general set of similar events in this direction that leak state information with probability more than that of random association. There exist several related works that look only at the initial keystream bytes of RC4 to obtain information regarding the state and eventually the secret keys (few recent examples are in [137]). However, these observations never work in the long term scenario. We ask the question:

> *"Is there a correlation between the RC4 keystream and the state that offers a long-term glimpse with probability more than* $2/N$*?"*

In this section, we answer to this question affirmatively. We prove the following: given that two consecutive bytes Z_r, Z_{r+1} of RC4 are equal to the specific value $(r+2)$ during the consecutive two rounds r and $r+1$ (modulo N), the probability that the $(r+1)$-th location of the state array during round r (denoted as $S_r[r+1]$ as per our notation) will be equal to $(N-1)$ is $3/N$, significantly higher than the probability of random association $1/N$.

We shall assume the probabilistic model of uniformly random permutation S anytime during PRGA, where the pointer j is assumed to be uniformly random and independent of the permutation.

4.3.1 Proof of the long-term glimpse

We start with our most important observation which we made while trying to obtain the scenario where the S array comes back to the same permutation after two consecutive rounds. As one may note in Figure 4.4, if in the r-th round, $j_r = i_r + 1$ and $S_r[j_r] = N-1$, then the two places swapped in round $(r+1)$ will be restored in round $(r+2)$. That is, we shall have S_{r+2} identical to S_r in such a case. This motivated our first result, as in Theorem 4.9.

Theorem 4.9. *After the r-th round ($r \geq 1$) of RC4 PRGA, we have*

$$\Pr(S_r[r+1] = N-1 \mid Z_{r+1} = Z_r) \approx \frac{2}{N}.$$

Proof. We shall first prove $\Pr(Z_{r+1} = Z_r \mid S_r[r+1] = N-1) \approx 2/N$, and then apply Bayes' theorem to get the desired result. The condition $S_r[r+1] = N-1$, and the path $j_r = r+1$ results in $j_{r+1} = j_r + S_r[r+1] = r+1+N-1 = r$,

which eventually gives

$$\begin{aligned} t_{r+1} &= S_{r+1}[i_{r+1}] + S_{r+1}[j_{r+1}] = S_r[j_{r+1}] + S_r[i_{r+1}] \\ &= S_r[r] + S_r[r+1] = S_r[i_r] + S_r[j_r] = t_r. \end{aligned}$$

Thus, $Z_{r+1} = S_{r+1}[t_{r+1}] = S_{r+1}[t_r]$ is equal to $Z_r = S_r[t_r]$ in almost all cases, except when t_r equals either i_{r+1} or j_{r+1}, the only two locations that get swapped in transition from S_r to S_{r+1}. Thus,

$$\Pr(Z_{r+1} = Z_r \mid S_r[r+1] = N-1 \ \wedge \ j_r = r+1) \approx 1.$$

This scenario is as illustrated in Figure 4.4.

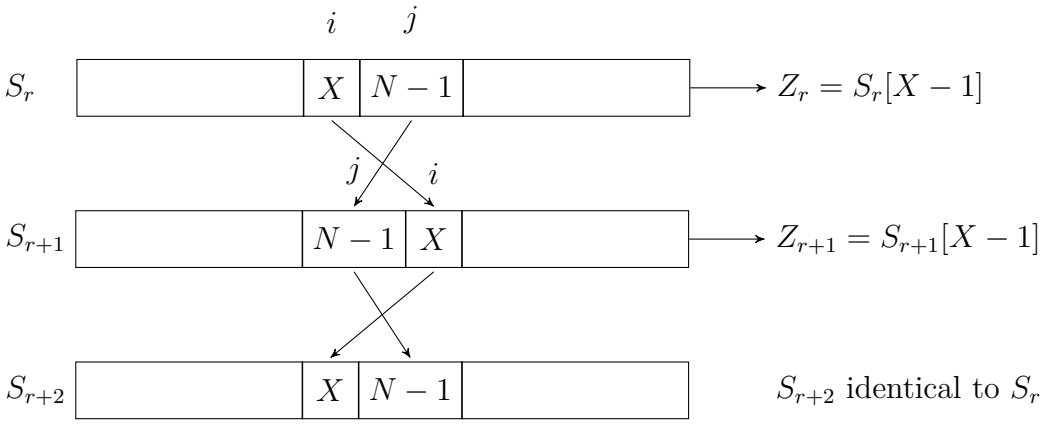

Figure 4.4: The scenario for $(Z_{r+1} = Z_r \mid S_r[r+1] = N-1 \ \wedge \ j_r = r+1)$.

We may now evaluate $\Pr(Z_{r+1} = Z_r \mid S_r[r+1] = N-1)$ as

$$\begin{aligned} &\Pr(Z_{r+1} = Z_r \mid S_r[r+1] = N-1 \ \wedge \ j_r = r+1) \cdot \Pr(j_r = r+1) \\ &+ \Pr(Z_{r+1} = Z_r \mid S_r[r+1] = N-1 \ \wedge \ j_r \neq r+1) \cdot \Pr(j_r \neq r+1) \\ &\approx 1 \cdot 1/N + 1/N \cdot (1 - 1/N) \approx 2/N. \end{aligned}$$

Applying Bayes' theorem to the above result, we obtain

$$\begin{aligned} &\Pr(S_r[r+1] = N-1 \mid Z_{r+1} = Z_r) \cdot \Pr(Z_{r+1} = Z_r) \\ &= \Pr(Z_{r+1} = Z_r \mid S_r[r+1] = N-1) \cdot \Pr(S_r[r+1] = N-1) \approx 2/N \cdot 1/N. \end{aligned}$$

Assuming pseudorandomness of RC4 keystream bytes, we may write that $\Pr(Z_{r+1} = Z_r) \approx 1/N$ (experimentally verified over a billion trials). This gives $\Pr(S_r[r+1] = N-1 \mid Z_{r+1} = Z_r) \approx 2/N$. □

Thus the event $(Z_{r+1} = Z_r)$ leaks the information of a single permutation location with probability twice that of random association. We would like to point out a simple corollary of the Glimpse theorem (Theorem 4.8) that leaks the information of a permutation location with the same probability.

Corollary 4.10 (Glimpse corollary)**.** *After the r-th round of RC4 PRGA, for $r \geq 1$, we have* $\Pr(S_r[r+1] = N-1 \mid Z_{r+1} = r+2) \approx \frac{2}{N}$.

Proof. In RC4 transition between rounds r and $r+1$, we have $i_{r+1} = r+1$, and $S_{r+1}[j_{r+1}] = S_r[i_{r+1}] = S_r[r+1]$, due to the swap in round r. Thus, by the Glimpse theorem (Theorem 4.8), we have

$$\Pr(S_r[r+1] = r+1-Z_{r+1}) \;\approx\; 2/N.$$

In case of $Z_{r+1} = r+2$, we get the desired conditional result. □

In the above two scenarios (Theorem 4.9 and Corollary 4.10), we find two different cases that leak the value of a specific location in the S array (namely, $S_r[r+1]$) with probability $2/N$ in each case. Moreover, the two events seem to be unrelated, or at least not completely dependent. Thus, it is quite natural to expect that considering the events together, one may have better confidence about the value in that specific location $S_r[r+1]$. In this direction, we present our result in Theorem 4.11.

Theorem 4.11. *After the r-th round ($r \geq 1$) of RC4 PRGA, we have*

$$\Pr(S_r[r+1] = N-1 \mid Z_{r+1} = Z_r \;\wedge\; Z_{r+1} = r+2) \;\approx\; \frac{3}{N}.$$

Proof. Let us define the main events as follows:

$$A := (S_r[r+1] = N-1),\; B := (Z_{r+1} = Z_r),\; C := (Z_{r+1} = r+2).$$

The result requires $\Pr(A|B \wedge C)$, and it seems that a naive composition of Theorem 4.9 (which gives $\Pr(A|B)$) and Theorem 4.10 (which gives $\Pr(A|C)$)

will produce the desired result. However, this is not the case. If we try to compute $\Pr(A \wedge B \wedge C)$ as $\Pr(B \wedge C|A)\Pr(A)$, then the first part is not easily computable as events B and C, conditional to event A, are not independent (verified experimentally over a billion trials). Hence we try the following route.

$$\Pr(A \wedge B \wedge C) = \Pr(C|B \wedge A) \cdot \Pr(B|A) \cdot \Pr(A).$$

Still there remains a problem with the first part, as event C occurs simultaneously with the occurrence of Z_{r+1} in event B. This is easy to observe experimentally, but not so easy to prove in theory.

To avoid the aforesaid problem in computing $\Pr(C|B \wedge A)$, we rewrite the problem definition slightly, and try to prove

$$\Pr(S_r[r+1] = N-1 \mid Z_r = r+2 \ \wedge \ Z_{r+1} = r+2) \ \approx \ 3/N.$$

We compute this as $\Pr(A \wedge B' \wedge C) = \Pr(C|B' \wedge A) \cdot \Pr(B'|A) \cdot \Pr(A)$, where $A := (S_r[r+1] = N-1)$, $C := (Z_{r+1} = r+2)$ as before, and $B' := (Z_r = r+2)$. Now we may compute $\Pr(C|B' \wedge A)$ easily, as event C occurs after completion of both the events A and B'.

Computing $\Pr(C|B' \wedge A)$*:* Note that event $A := (S_r[r+1] = N-1)$ implies $Z_{r+1} = S_{r+1}[S_{r+1}[r+1] + S_r[r+1]] = S_{r+1}[S_{r+1}[r+1] - 1]$. And of course, event $B' := (Z_r = r+2)$ implies $Z_r = S_r[t_r] = r+2$. We consider the following paths for the proof.

- *Case I:* $(S_{r+1}[r+1] = r+2)$. In case of this path, we shall have $Z_{r+1} = S_{r+1}[(r+2)-1] = S_{r+1}[r+1] = r+2$, with probability of occurrence 1.
- *Case II:* $(S_{r+1}[r+1] = t_r + 1)$. In case of this path, we shall have $Z_{r+1} = S_{r+1}[(t_r+1)-1] = S_{r+1}[t_r] = S_r[t_r] = r+2$, with probability of occurrence approximately 1, disregarding the two cases when t_r may be equal to either i_{r+1} or j_{r+1}.

In almost all other cases, we may assume that $C := (Z_{r+1} = r+2)$ happens with probability of random association $1/N$ (verified experimentally over a

billion trials). We compute $\Pr(C|B' \wedge A)$ as

$$\begin{aligned}
&\Pr(C|B' \wedge A \wedge (S_{r+1}[r+1] = r+2)) \cdot \Pr(S_{r+1}[r+1] = r+2) \\
&+ \Pr(C|B' \wedge A \wedge (S_{r+1}[r+1] = r+2)) \cdot \Pr(S_{r+1}[r+1] = t_r + 1) \\
&+ \sum_{\substack{X \neq r+2 \\ X \neq t_r+1}} \Pr(C|B' \wedge A \wedge (S_{r+1}[r+1] = X)) \cdot \Pr(S_{r+1}[r+1] = X) \\
&\approx 1 \cdot 1/N + 1 \cdot 1/N + (1 - 2/N) \cdot 1/N \approx 3/N.
\end{aligned}$$

Computing $\Pr(A|B' \wedge C)$*:* As no glimpse-like connection has been found between $S_r[r+1]$ and Z_r in the literature to date, we may assume $\Pr(B'|A) \approx 1/N$ (verified experimentally over a billion trials), and we may of course take $\Pr(A) \approx 1/N$ as per natural pseudorandomness assumptions of RC4. Thus,

$$\Pr(A \wedge B' \wedge C) = \Pr(C|B' \wedge A) \cdot \Pr(B'|A) \cdot \Pr(A) \approx 3/N \cdot 1/N \cdot 1/N.$$

We may assume $\Pr(B' \wedge C) = \Pr(B') \cdot \Pr(C) \approx 1/N \cdot 1/N$ (verified experimentally over a billion trials), and this produces the desired conditional result $\Pr(A|B \wedge C) = \Pr(A|B' \wedge C) \approx 3/N$. □

4.3.2 Experimental results and discussion

We have performed extensive experiments to obtain accurate practical estimates of each of the results presented in this chapter. Each correlation reported in this chapter is of order $1/N$ with respect to a base event of probability $1/N$. Thus, $O(N^3)$ trials are sufficient to identify the biases with considerable probability of success (refer to [100,119] for detailed explanation on the complexity).

The experimental results presented in this section are based on an average of N^4 trials of RC4, with keys chosen uniformly at random. The experiments were carried out using GCC-compiled C-code on a Unix machine with 3.34 GHz processor and 8 GB of memory. Table 4.3 lists the theoretical estimates against the experimental values for each of the results presented in Section 4.3.

The values presented in Table 4.3 testify that our theoretical estimates for our new glimpse correlation and associated results closely match their respective experimental values. Slight deviations, if any, are due to marginal gaps of

Table 4.3: Experimental values and theoretical estimates of our results, where $A := (S_r[r+1] = N-1)$, $B := (Z_{r+1} = Z_r)$ and $C := (Z_{r+1} = r+2)$.

Biased event	Probability of the biased event		Result
	Experiment	Theory	
$(A \mid B)$	0.0077881670	$2/N = 0.0078125$	Theorem 4.9
$(A \mid C)$	0.0078166422	$2/N = 0.0078125$	Corollary 4.10
$(A \mid B \wedge C)$	0.0117323766	$3/N = 0.01171875$	Theorem 4.11

order $1/N^2$ or less, which we have purposefully disregarded in case of theory.

Discussion on the new glimpse correlation

The glimpse correlations have been quite well studied in RC4 literature, as they provide practical leaks into the state permutation of the cipher from the knowledge of the output keystream. Glimpse correlations can be exploited towards state-recovery and key-recovery attacks on RC4. One may find some important results in state-recovery attacks on RC4 in [57, 107], and a few attacks along the lines of RC4 key-recovery from the permutation in [4,15,117].

Although glimpse biases provide practical cryptanalytic tools against RC4, not many have been identified over the last two decades of analysis. Jenkins [75] was the first to report a glimpse into RC4 state from the keystream with probability $2/N$, and it has since been the best one that persists in the long-term evolution of the PRGA. Later in 2001 and 2005, Mantin [100, 102] generalized the glimpse correlations into 'useful states' of RC4, which included Jenkins' correlations as a special case. These biases were again of magnitude $2/N$, and persisted in the long-term evolution of PRGA. In recent times, several correlations between the state permutation and keystream have been observed, mainly by Sepehrdad et al [136, 137], and later proved by us in [131, 132]. Although these correlations are larger in magnitude, none persist in the long-term, and only pertain to the initial bytes of the output.

Our result in this direction provides a long-term glimpse correlation of magnitude $3/N$, the best to date. It is interesting to note that no long-term glimpse bias of magnitude more than $2/N$ has been reported in the literature over the last 15 years, since the first one [75] in 1996. The result has been

published in our recent work [99].

The long-term glimpse correlations reported in the literature to date generally relate a keystream output byte to a single location of the state permutation, typically at a specific round of RC4. Thus, simultaneous knowledge of two or more keystream bytes may help in guessing two or more permutation locations, but does not always provide additional benefits in guessing a single location of the permutation over any one of them. Our result combines the knowledge of two consecutive output bytes Z_r, Z_{r+1} to obtain a significant advantage in guessing a single permutation location $S_r[r+1]$.

Chapter 5

Biases in Keystream Bytes of RC4

In the previous chapter, we discussed some biases involving the RC4 state variables S, i, j, during RC4 PRGA. Although a few of those biases involved the keystream bytes, we did not consider biases concerning *only* the keystream bytes. In this chapter, we concentrate only on biases exhibited by RC4 keystream bytes towards constant values in $\{0, \ldots, 255\}$.

This chapter deals with the following problems in RC4 analysis, as mentioned earlier in Section 1.4 of Chapter 1, the organization of this thesis.

Problem 1e. Theoretically justify the sinusoidal probability distribution of Z_1 and its negative bias towards zero, as observed by Mironov [112].

Problem 1f. Identify all significant biases towards zero in the initial bytes of RC4 keystream (Z_3 to Z_{255}), and prove all subsequent results.

Problem 1g. Prove all significant biases in the initial bytes of RC4 keystream identified by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5].

Problem 1h. Experimentally discover and subsequently prove biases in RC4 keystream which remain effective even after discarding the initial bytes.

Problem 1e has been solved in our work [132], Problem 1f was studied and solved in our papers [98, 132], Problem 1g was completed in our recent work [128], and Problem 1h was studied in our paper [132].

5.1 Probability distribution of first byte

Here we derive the complete probability distribution of the first RC4 keystream byte Z_1, as observed by Mironov [112, Figure 6] in CRYPTO 2002. Before proceeding to prove the general result, we start with a specific case, namely, the negative bias of Z_1 towards 0.

5.1.1 Negative bias in Z_1 towards zero

The special case of Z_1's negative bias towards 0 is contained in the complete probability distribution of Z_1 to be proved shortly. However, we present a separate proof for this special case because, unlike the proof for the complete case, this special case has a much simpler proof which reveals a different relationship of the RC4 state variables. This is elaborated further in Remark 5.3.

Theorem 5.1. *Suppose the initial permutation S_0 of RC4 PRGA is randomly chosen from the set of all permutations of $\{0, 1, \ldots, N-1\}$. Then the probability that the first keystream byte of RC4 is zero is* $\Pr(Z_1 = 0) \approx 1/N - 1/N^2$.

Proof. We explore the probability $\Pr(Z_1 = 0)$ using the following paths:

$$\begin{aligned}\Pr(Z_1 = 0) = &\Pr(Z_1 = 0 \mid S_0[j_1] = 0) \cdot \Pr(S_0[j_1] = 0) \\ &+ \Pr(Z_1 = 0 \mid S_0[j_1] \neq 0) \cdot \Pr(S_0[j_1] \neq 0).\end{aligned}$$

Case I: $S_0[j_1] = 0$: Suppose that $j_1 = S_0[1] = X \neq 1$ and $S_0[j_1] = S_0[S_0[1]] = 0$. Then we have

$$Z_1 = S_1[S_1[1] + S_1[X]] = S_1[S_0[X] + S_0[1]] = S_1[0 + X] = S_0[1] = X \neq 0,$$

as S_0 is a permutation, where X and 0 belong to two different indices 1 and X. Thus, in this case we have $\Pr(Z_1 = 0 \mid S_0[j_1] = 0) \approx 0$.

Case II: $S_0[j_1] \neq 0$: In this case, output byte Z_1 can be considered uniformly random, and thus $\Pr(Z_1 = 0 \mid S_0[j_1] \neq 0) \approx 1/N$.

Combining the two cases, the total probability that the first output byte is 0 is given by $\Pr(Z_1 = 0) \approx 0 \cdot 1/N + 1/N \cdot (1 - 1/N) = 1/N - 1/N^2$. □

From Theorem 5.1, we immediately get a distinguisher of RC4 that can effectively distinguish the output keystream of the cipher from a random sequence of bytes. For the event $E : (Z_1 = 0)$, the bias proved above can be written as $p(1+q)$, where $p = 1/N$ and $q = -1/N$. The number of samples required to distinguish RC4 from random sequence of bits with a constant probability of success in this case is approximately N^3.

5.1.2 Complete distribution of Z_1

In this section, we turn our attention to the complete probability distribution of the first byte Z_1. In [112, Figure 6], the empirical plot of Z_1 has a peculiar sinusoidal pattern which is not observed for any other variables or events related to RC4. In Theorem 5.2, we theoretically derive this interesting distribution.

We shall assume the probabilistic model of uniform random keys to prove the results. We assume actual RC4 next-state-function for the evolution of S and i, j, and no randomness assumptions are made on the initial state S_0 of PRGA.

Theorem 5.2. *The probability distribution of the first output byte of RC4 keystream is as follows, where* $v \in \{0, \ldots, N-1\}$, $\mathcal{L}_v = \{0, 1, \ldots, N-1\} \setminus \{1, v\}$ *and* $\mathcal{T}_{v,X} = \{0, 1, \ldots, N-1\} \setminus \{0, X, 1-X, v\}$.

$$\Pr(Z_1 = v)$$
$$= Q_v + \sum_{X \in \mathcal{L}_v} \sum_{Y \in \mathcal{T}_{v,X}} \Pr(S_0[1] = X \ \wedge \ S_0[X] = Y \ \wedge \ S_0[X+Y] = v),$$

$$\textit{with} \quad Q_v = \begin{cases} \Pr(S_0[1] = 1 \ \wedge \ S_0[2] = 0), & \textit{if } v = 0; \\ \Pr(S_0[1] = 0 \ \wedge \ S_0[0] = 1), & \textit{if } v = 1; \\ \Pr(S_0[1] = 1 \ \wedge \ S_0[2] = v) & \\ \quad + \Pr(S_0[1] = v \ \wedge \ S_0[v] = 0) & \\ \quad + \Pr(S_0[1] = 1 - v \ \wedge \ S_0[1-v] = v), & \textit{otherwise.} \end{cases}$$

Proof. The first output byte Z_1 can be explicitly written as

$$Z_1 = S_1[S_1[i_1] + S_1[j_1]] = S_1[S_0[j_1] + S_0[i_1]] = S_1[S_0[S_0[1]] + S_0[1]] = S_1[Y + X],$$

where we denote $j_1 = S_0[1]$ by X and $S_0[S_0[1]] = S_0[X]$ by Y. Thus, we have

$$\Pr(Z_1 = v) = \sum_{X=0}^{N-1} \sum_{Y=0}^{N-1} \Pr(S_0[1] = X \ \wedge \ S_0[X] = Y \ \wedge \ S_1[X+Y] = v).$$

Special cases depending on X, Y: Our goal is to write all probability expressions in terms of S_0. To express $S_1[X+Y]$ in terms of S_0, we observe that the state S_1 is different from S_0 in at most two places, $i_1 = 1$ and $j_1 = X$. Thus, we need to treat specially the case $X + Y = 1$, which holds if and only if $Y = 1 - X$, and $X + Y = X$, which holds if and only if $Y = 0$. Another special case to consider is $X = 1$, which holds if and only if $Y = X$, where no swap occurs from S_0 to S_1. These special cases are as follows:

- $X + Y = 1$ if and only if $Y = 1 - X$, which implies
  $Z_1 = S_1[1] = S_1[i_1] = S_0[j_1] = S_0[X] = Y = 1 - X$,
- $X + Y = X$ if and only if $Y = 0$, which implies
  $Z_1 = S_1[X] = S_1[j_1] = S_0[i_1] = S_0[1] = X$, and
- $X = 1$ if and only if $Y = X$, which implies
  $Z_1 = S_1[X+Y] = S_0[X+Y] = S_0[1+1] = S_0[2]$.

In all other circumstances, we would have $Z_1 = S_1[X+Y] = S_0[X+Y]$. Considering all the special cases as discussed above, we obtain $\Pr(Z_1 = v)$ in terms of S_0 as follows:

$$\begin{aligned}
\Pr(Z_1 = v) = &\sum_{X=0}^{N-1} \Pr(S_0[1] = X \ \wedge \ S_0[X] = 1 - X \ \wedge \ 1 - X = v) \\
&+ \sum_{X=0}^{N-1} \Pr(S_0[1] = X \ \wedge \ S_0[X] = 0 \ \wedge \ X = v) \\
&+ \Pr(S_0[1] = 1 \ \wedge \ S_0[2] = v) \\
&+ \sum_{X \neq 1} \sum_{Y \neq 0, X, 1-X} \Pr(S_0[1] = X \ \wedge \ S_0[X] = Y \ \wedge \ S_0[X+Y] = v).
\end{aligned}$$

The first sum refers to the special case $Y = 1 - X$ and the second one refers to $Y = 0$. The special case $X = 1$, which holds if and only if $Y = X$, merges to produce the third term, common point $(X = 1, Y = 1)$. All other points on $X = 1$ and $Y = X$ are discarded. The last double summation term denotes all other general cases.

One may refer to Figure 5.1 to obtain a clearer exposition of the ranges of summations required to be considered for this proof.

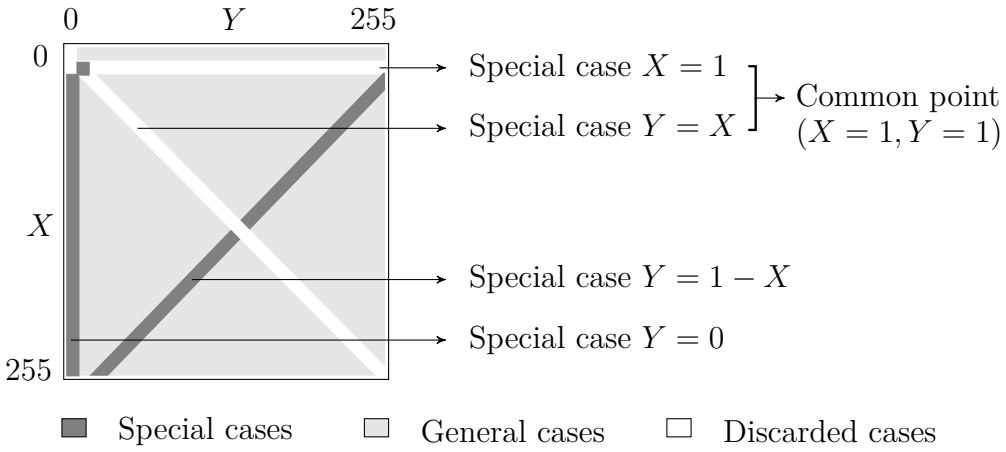

Figure 5.1: X, Y dependent special cases and range of sums for evaluation of $\Pr(Z_1 = v)$ in terms of S_0.

Special cases depending on v: The first summation term reduces to a single point $(X = 1 - v, Y = v)$, as we fix $1 - X = v$ and $Y = 1 - X$. The second summation, similarly, reduces to the point $(X = v, Y = 0)$. Furthermore, we have two impossible cases in the double summation:

- $(X = v, Y \neq 0)$ which implies
  $S_1[v] = v,\ X + Y \neq v \ \Rightarrow\ Z_1 = S_1[X + Y] \neq v$, and
- $(X \neq 1 - v, Y = v)$ which implies
  $S_1[1] = S_0[X] = v,\ X + Y \neq 1 \ \Rightarrow\ Z_1 = S_1[X + Y] \neq v$.

Hence, the most general form for the probability $\Pr(Z_1 = v)$ can be written as

$$Q_v + \sum_{X \in \mathcal{L}_v} \sum_{Y \in \mathcal{T}_{v,X}} \Pr(S_0[1] = X \ \wedge\ S_0[X] = Y \ \wedge\ S_0[X + Y] = v),$$

where $Q_v = \Pr(S_0[1] = 1 - v \ \wedge\ S_0[1 - v] = v) + \Pr(S_0[1] = v \ \wedge\ S_0[v] = 0) + \Pr(S_0[1] = 1 \ \wedge\ S_0[2] = v)$.

Value of Q_v: State S_0 being a permutation, some of the probability terms in Q_v are 0 when v takes particular values. We have the following three cases:

- Case $v = 0$: We have $Q_0 = \Pr(S_0[1] = 1 \ \wedge \ S_0[1] = 0) + \Pr(S_0[1] = 0 \ \wedge \ S_0[0] = 0) + \Pr(S_0[1] = 1 \ \wedge \ S_0[2] = 0) = \Pr(S_0[1] = 1 \ \wedge \ S_0[2] = 0)$, as S_0 is a permutation.

- Case $v = 1$: We have $Q_v = \Pr(S_0[1] = 0 \ \wedge \ S_0[0] = 1) + \Pr(S_0[1] = 1 \ \wedge \ S_0[1] = 0) + \Pr(S_0[1] = 1 \ \wedge \ S_0[2] = 1) = \Pr(S_0[1] = 0 \ \wedge \ S_0[0] = 1)$, as S_0 is a permutation.

- Case $v \neq 0, 1$: Here we have no conflicts or special conditions as in the previous cases, and hence the general form of Q_v holds.

Combining the general formula for $\Pr(Z_1 = v)$ and all three cases for Q_v, we obtain the desired theoretical probability distribution for the first byte Z_1. $\square$

5.1.3 Estimation of the joint probabilities

We consider two special cases while computing the numeric values of $\Pr(Z_1 = v)$. First, we investigate RC4 PRGA where S_0 is fed from the output of RC4 KSA, as in practice. Next, we probe into the scenario when S_0 is random.

Assume that the initial permutation S_0 of RC4 PRGA is constructed from the regular KSA, i.e., the probabilities $\Pr(S_0[u] = v)$ follow the distribution mentioned in Proposition 3.1. However, we require the joint probabilities like $\Pr(S_0[1] = X \ \wedge \ S_0[X] = Y \ \wedge \ S_0[X + Y] = v)$ in our formula derived in Theorem 5.2, and we devise the following estimates for these joint probabilities.

- Consider the joint probability $\Pr(S_0[u] = v \ \wedge \ S_0[u'] = v')$ where $u \neq u'$ and $v \neq v'$. We can represent this by $\Pr(S_0[u] = v \ \wedge \ S_0[u'] = v') = \Pr(S_0[u] = v) \cdot \Pr(S_0[u'] = v' \mid S_0[u] = v)$. The first term is estimated directly from Proposition 3.1. For the second term, $S_0[u] = v \ \Rightarrow \ S_0[u'] \neq v$. Thus we normalize $\Pr(S_0[u'] = v)$ and estimate the second term as

$$\Pr(S_0[u'] = v' \mid S_0[u] = v) \approx \Pr(S_0[u'] = v') + \frac{\Pr(S_0[u'] = v)}{N - 1}.$$

- For the joint probability $\Pr(S_0[u] = v \ \wedge \ S_0[u'] = v' \ \wedge \ S_0[u''] = v'')$, we can represent it by $\Pr(S_0[u] = v) \cdot \Pr(S_0[u'] = v' \mid S_0[u] = v) \cdot \Pr(S_0[u''] =$

$v'' \mid S_0[u'] = v' \ \wedge \ S_0[u] = v)$. The first term comes from Proposition 3.1 and the second term as above. The third term is estimated as

$$\begin{aligned}&\Pr(S_0[u''] = v'' \mid S_0[u'] = v' \ \wedge \ S_0[u] = v) \\ &\approx \Pr(S_0[u''] = v'') + \frac{\Pr(S_0[u''] = v')}{N-2} + \frac{\Pr(S_0[u''] = v)}{N-2}.\end{aligned}$$

We compute the theoretical values of $\Pr(Z_1 = v)$ using Theorem 5.2 and Proposition 3.1, along with the estimations for joint probabilities discussed above. Figure 5.2 shows the theoretical and experimental probability distributions of Z_1, where the experimental data is generated over 100 million runs of RC4 PRGA using 256-byte (full length) secret keys[1]. The figure clearly shows that our theoretical justification closely matches the experimental data, and justifies the observation by Mironov [112].

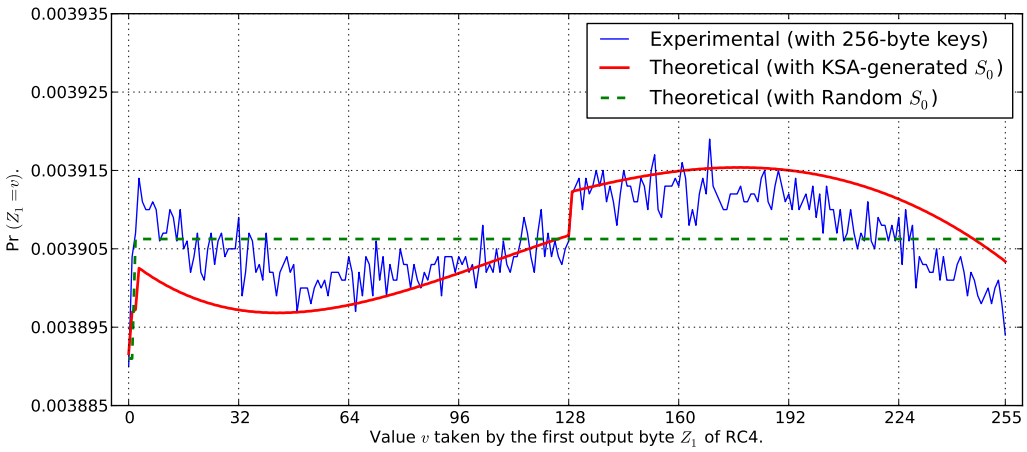

Figure 5.2: The probability distribution of the first output byte Z_1.

As an alternative to the *additive correction* described above for estimating the conditionals, one may consider *multiplicative correction* by normalizing the probabilities as follows:

- Estimate $\Pr(S_0[u'] = v' \mid S_0[u] = v)$ as $\dfrac{\Pr(S_0[u'] = v')}{1 - \Pr(S_0[u'] = v)}$.

[1]The typographic error 'with 16 byte keys' from [132, Fig.9] is corrected in this figure.

- Estimate $\Pr(S_0[u''] = v'' \mid S_0[u'] = v' \wedge S_0[u] = v)$ as

$$\frac{\Pr(S_0[u''] = v'')}{1 - \Pr(S_0[u''] = v') - \Pr(S_0[u''] = v)}.$$

We found that the numeric values of $\Pr(Z_1 = v)$ estimated using the two different models (additive and multiplicative) almost coincide and the graphs fall right on top of one another.

If the initial permutation S_0 of RC4 PRGA is considered to be random, then we would have $\Pr(S_0[u] = v) \approx 1/N$ for all u, v, and the joint probabilities can be computed directly (samples drawn without replacement). Substituting all the relevant probability values, we get

$$\begin{aligned} \Pr(Z_1 = 0) &\approx \Pr(Z_1 = 1) \approx \frac{1}{N} - \frac{1}{N(N-1)}, \quad \text{and} \\ \Pr(Z_1 = v) &\approx \frac{1}{N} + \frac{1}{N(N-1)(N-2)} \quad \text{for } 2 \leq v \leq 255, \end{aligned}$$

which is almost a uniform distribution for $2 \leq v \leq 255$. The dashed line in Figure 5.2 shows the graph for this theoretical distribution, and it closely matches our experimental data as well (we omit the experimental curve for random S_0 as it coincides with the theoretical one).

Remark 5.3. *Theorem 5.1 is the special case $v = 0$ of Theorem 5.2 and hence may seem redundant. However, we like to point out that the former has a simple and straightforward proof assuming S_0 to be random and the latter has a rigorous general proof without any assumption on S_0. The result of Theorem 5.1 signifies that this negative bias is not an artifact of non-random S_0 produced by RC4 KSA, rather it would be present, even if one starts PRGA with a uniform random permutation.*

5.2 Biases of initial bytes towards zero

In FSE 2001, Mantin and Shamir [103] proved the famous $2/N$ bias towards the value 0 for the second byte of RC4 keystream. They also claimed that

- MS-Claim-1: $\Pr(Z_r = 0) = \frac{1}{N}$ at PRGA rounds $3 \leq r \leq 255$.

- MS-Claim-2: $\Pr(Z_r = 0 \mid j_r = 0) > \frac{1}{N}$ and $\Pr(Z_r = 0 \mid j_r \neq 0) < \frac{1}{N}$ at PRGA rounds $3 \leq r \leq 255$.

It is reasoned in [103] that the two biases in MS-Claim-2 cancel each other to produce no bias in the event $(Z_r = 0)$ in rounds 3 to 255, thereby justifying MS-Claim-1. In this section, contrary to MS-Claim-1, we show (in Theorem 5.5) that $\Pr(Z_r = 0) > \frac{1}{N}$ for all rounds r from 3 to 255. To prove the main result, we will require Corollary 4.4. For ease of reference, we restate another version of this corollary below.

Corollary 5.4 (same as Corollary 4.4)**.** *For PRGA rounds* $3 \leq r \leq N-1$*,*

$$\Pr(S_{r-1}[r] = r) \approx \Pr(S_1[r] = r) \left(1 - \frac{1}{N}\right)^{r-2} + \sum_{t=2}^{r-1} \sum_{w=0}^{r-t} \frac{\Pr(S_1[t] = r)}{w! \cdot N} \left(\frac{r-t-1}{N}\right)^{w} \left(1 - \frac{1}{N}\right)^{r-3-w}.$$

5.2.1 Proof of biases in $(Z_r = 0)$ for $3 \leq r \leq N-1$

We shall assume the probabilistic model of uniform random keys to prove the results. We assume actual RC4 next-state-function for the evolution of S and i, j, and no randomness assumptions are made on the initial state S_0 of PRGA.

Theorem 5.5. *In PRGA rounds* $3 \leq r \leq N-1$*, probability* $\Pr(Z_r = 0)$ *is:*

$$\Pr(Z_r = 0) \approx \frac{1}{N} + \frac{c_r}{N^2},$$

$$\textit{where} \quad c_r = \begin{cases} \frac{N}{N-1}\left(N \cdot \Pr(S_{r-1}[r] = r) - 1\right) - \frac{N-2}{N-1}, & \textit{for } r = 3; \\ \frac{N}{N-1}\left(N \cdot \Pr(S_{r-1}[r] = r) - 1\right), & \textit{otherwise.} \end{cases}$$

Proof. The expression for c_r has an extra term $\left(-\frac{N-2}{N-1}\right)$ in the case $r = 3$, and everything else is the same as in the general formula for $4 \leq r \leq N-1$. We shall first prove the general formula for $4 \leq r \leq N-1$, and then justify the extra term for the special case $r = 3$. We may write:

$$\Pr(Z_r = 0) = \Pr\left(Z_r = 0 \ \wedge \ S_{r-1}[r] = r\right) + \Pr\left(Z_r = 0 \ \wedge \ S_{r-1}[r] \neq r\right). \quad (5.1)$$

We will also use the expression $Z_r = S_r[S_r[i_r] + S_r[j_r]] = S_r[S_r[r] + S_{r-1}[i_r]] = S_r[S_r[r] + S_{r-1}[i_r]]$, that is, $Z_r = S_r[S_r[r] + S_{r-1}[r]]$.

Calculation of $\Pr(Z_r = 0 \wedge S_{r-1}[r] = r)$: In this case, $Z_r = 0$, which implies $S_r[S_r[r] + r] = 0$, and thus:

$$\Pr(Z_r = 0 \wedge S_{r-1}[r] = r) = \sum_{x=0}^{N-1} \Pr(S_r[x + r] = 0 \wedge S_r[r] = x \wedge S_{r-1}[r] = r).$$

Now the events $(S_r[x + r] = 0)$ and $(S_r[r] = x)$ are both independent of $(S_{r-1}[r] = r)$, as a state update has occurred in the process, and $S_{r-1}[r] = r$ is one of the values that got swapped. Hence,

$$\begin{aligned}&\Pr(Z_r = 0 \ \wedge \ S_{r-1}[r] = r)\\ &= \sum_{x=0}^{N-1} \Pr(S_r[x + r] = 0) \cdot \Pr(S_r[r] = x \mid S_r[x + r] = 0) \cdot \Pr(S_{r-1}[r] = r).\end{aligned}$$

We note that if there exists any bias in the event $(S_r[x + r] = 0)$, then it must propagate from a similar bias in $(S_0[x+r] = 0)$, as was the case for $(S_{r-1}[r] = r)$ in Corollary 4.4. However, $\Pr(S_0[x + r] = 0) = 1/N$ by Proposition 3.1, and thus we assume $\Pr(S_r[x+r] = 0) \approx 1/N$ as well. For $\Pr(S_r[r] = x \mid S_r[x+r] = 0)$, we have the following two cases:

$$\begin{aligned}&x = 0 \ \Rightarrow \ x + r = r, \text{ which gives } (S_r[x + r] = 0) \Leftrightarrow (S_r[r] = x = 0), \text{ and}\\ &x \neq 0 \ \Rightarrow \ x + r \neq r, \text{ which gives } (S_r[x + r] = 0) \Leftrightarrow (S_r[r] = x \neq 0).\end{aligned}$$

Moreover, in the second case, the value of $S_r[r]$ is independent of $S_{r-1}[r]$ because $[r] = [i_r]$ position got swapped to generate S_r from S_{r-1}. Thus,

$$\Pr(S_r[x + r] = 0 \mid S_r[r] = x) = \begin{cases} 1, & \text{if } x = 0; \\ 1/(N - 1), & \text{if } x \neq 0. \end{cases} \tag{5.2}$$

Combining all the above probability values, we get the probability of the con-

ditional event $(Z_r = 0 \wedge S_{r-1}[r] = r)$ as

$$\begin{aligned}
&\Pr\left(Z_r = 0 \wedge S_{r-1}[r] = r\right) \\
&\approx \frac{1}{N} \cdot \Pr\left(S_{r-1}[r] = r\right) \cdot \sum_{x=0}^{N-1} \Pr\left(S_r[x+r] = 0 \mid S_r[r] = x\right) \\
&= \frac{1}{N} \cdot \Pr(S_{r-1}[r] = r) \cdot \left(1 + (N-1) \cdot \frac{1}{N-1}\right) \\
&= \frac{2}{N} \cdot \Pr(S_{r-1}[r] = r). \qquad (5.3)
\end{aligned}$$

Now we only need to compute $\Pr\left(Z_r = 0 \wedge S_{r-1}[r] \neq r\right)$ to complete this result.

Calculation of $\Pr\left(Z_r = 0 \wedge S_{r-1}[r] \neq r\right)$: Similarly to the previous case, we can derive $\Pr\left(Z_r = 0 \ \wedge \ S_{r-1}[r] \neq r\right)$ as

$$\sum_{y \neq r} \sum_{x=0}^{N-1} \Pr\left(S_r[x+y] = 0 \ \wedge \ S_r[r] = x \ \wedge \ S_{r-1}[r] = y\right).$$

In the above expression, we have $y \neq r$ and $x = r - y$, which implies $S_r[x+y] = S_r[r] = 0$ and $S_r[r] = x = r - y \neq 0$. This is a contradiction. Moreover, the events $(S_r[x+y] = 0)$ and $(S_r[r] = x)$ are both independent of $(S_{r-1}[r] = y)$, as $S_{r-1}[r]$ got swapped in the state update. Thus,

$$\begin{aligned}
&\Pr\left(Z_r = 0 \wedge S_{r-1}[r] \neq r\right) \\
&= \sum_{y \neq r} \sum_{x \neq r-y} \Pr\left(S_r[x+y] = 0 \wedge S_r[r] = x\right) \cdot \Pr\left(S_{r-1}[r] = y\right).
\end{aligned}$$

Similarly to the derivation of Equation (5.2), we obtain

$$\begin{aligned}
&\Pr\left(S_r[x+y] = 0 \ \wedge \ S_r[r] = x\right) \\
&= \begin{cases} 0 \cdot (1/N) = 0, & \text{if } x = 0; \\ (1/(N-1)) \cdot (1/N) = 1/(N(N-1)), & \text{if } x \neq 0. \end{cases} \qquad (5.4)
\end{aligned}$$

The only difference occurs in the case $x = 0$. Here we get $y \neq r$ and $x = 0$, which implies $S_r[x+y] = S_r[y] = 0$ and $S_r[r] = x = 0$. This is a contradiction as $y \neq r$ are distinct locations in the permutation S_r. In all other cases $(x \neq 0)$,

the argument is same as before. Combining the above probabilities, we get

$$\begin{aligned}
&\Pr\left(Z_r = 0 \wedge S_{r-1}[r] \neq r\right) \\
&\approx \sum_{y \neq r} \Pr\left(S_{r-1}[r] = y\right) \left(0 + \sum_{x \neq r-y,0} \frac{1}{N(N-1)}\right) \\
&= \sum_{y \neq r} \Pr\left(S_{r-1}[r] = y\right) \cdot (N-2) \cdot \frac{1}{N(N-1)} \\
&= \frac{N-2}{N(N-1)} \cdot \left(1 - \Pr\left(S_{r-1}[r] = r\right)\right). \qquad (5.5)
\end{aligned}$$

Combining Equations (5.1), (5.3) and (5.5) should now produce the result.

Calculation for $\Pr(Z_r = 0)$: Combining Equations (5.1), (5.3) and (5.5), we obtain $\Pr(Z_r = 0)$ as approximately equal to

$$\frac{2}{N} \cdot \Pr(S_{r-1}[r] = r) + \frac{N-2}{N(N-1)} \cdot (1 - \Pr(S_{r-1}[r] = r)) = \frac{1}{N} + \frac{c_r}{N^2}, \qquad (5.6)$$

where $c_r = \frac{N}{N-1}\left(N \cdot \Pr(S_{r-1}[r] = r) - 1\right)$, as required in the general case.

Special case for $r = 3$: The expression for $\Pr\left(Z_r = 0 \wedge S_{r-1}[r] = r\right)$ is identical to that in the general case, that is, the same as in Equation (5.3). However, for $\Pr\left(Z_r = 0 \wedge S_{r-1}[r] \neq r\right)$, we have a special case. For $r = 3$, if $S_{r-1}[r] = S_2[3] = 0$, we have $j_3 = j_2 + S_2[3] = j_2$, and thus $Z_3 = 0$ and $S_2[3] = 0$, which in turn implies

$$\begin{aligned}
&S_3[S_3[3]] = S_3[S_2[j_3]] = S_3[S_2[j_2]] = S_3[S_1[2]] = 0 \\
&S_2[3] = S_3[j_3] = S_3[j_2] = S_3[j_1 + S_1[2]] = 0.
\end{aligned}$$

This further implies $j_1 = S_0[1] = 0$, which poses a contradiction, as $S_0[1] = S_1[0] = 0$ can only produce $S_2[i_2] = S_2[2] = 0$ in the case $j_2 = 0$, and may never result in $S_2[3] = 0$. Thus, for $r = 3$, Equation (5.5) changes as follows:

$$\begin{aligned}
&\Pr\left(Z_r = 0 \wedge S_{r-1}[r] \neq r\right) \\
&\approx \frac{N-2}{N(N-1)} \cdot \left(1 - \Pr\left(S_{r-1}[r] = r\right) - \Pr\left(S_{r-1}[r] = 0\right)\right) \\
&= \frac{N-2}{N(N-1)} \cdot \left(1 - \Pr\left(S_{r-1}[r] = r\right)\right) - \frac{N-2}{N^2(N-1)}, \quad \text{by Proposition 3.1.}
\end{aligned}$$

This gives the special expression of $c_r = \frac{N}{N-1}\left(N \cdot \Pr(S_{r-1}[r] = r) - 1\right) - \frac{N-2}{N-1}$.

The extra term does not appear in the general case $4 \leq r \leq N-1$, because we have $Z_r = 0$ and $S_{r-1}[r] = 0$, which implies

$$S_r[S_r[r]] = S_r[S_{r-1}[j_r]] = S_r[S_{r-1}[j_{r-1}]] = S_r[S_{r-2}[r-1]] = 0$$
$$S_{r-1}[r] = S_r[j_r] = S_r[j_{r-1}] = S_r[j_{r-2} + S_{r-2}[r-1]] = 0$$

This further implies $j_{r-2} = 0$, which does not pose any contradiction for $r > 3$, as we can assume j_{r-2} to be random and independent to the condition $S_{r-1}[r] = y = 0$ in these cases. Hence the complete result. □

Corollary 5.6. *For $N = 256$ and $3 \leq r \leq 255$, the probability $\Pr(Z_r = 0)$ is bounded as follows:*

$$\frac{1}{N} + \frac{1.337057}{N^2} \;\geq\; \Pr(Z_r = 0) \;\geq\; \frac{1}{N} + \frac{0.242811}{N^2}.$$

Numerical calculation of c_r for $N = 256$ and $3 \leq r \leq 255$ gives that c_r decreases for $4 \leq r \leq 255$ (as in Figure 5.3). Thus, $c_4 = 1.337057 \geq c_r \geq 0.242811 = c_{255}$ for $4 \leq r \leq 255$, and the special case $c_3 = 0.351089$ for $r = 3$ also falls within the same bounds. Hence the bounds on $\Pr(Z_r = 0)$.

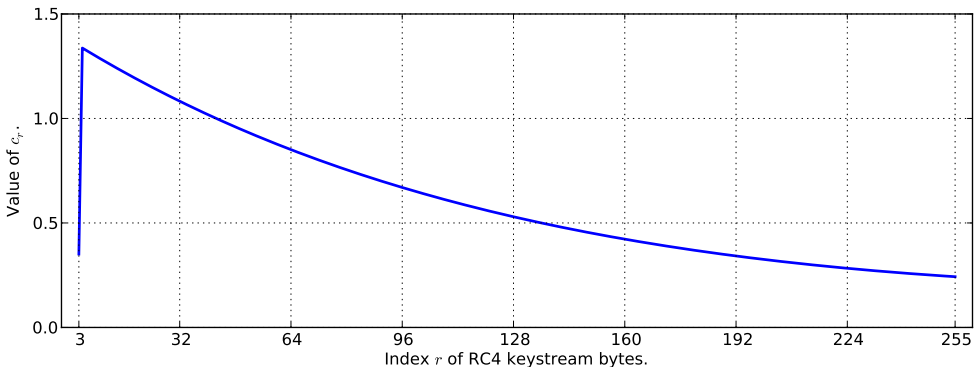

Figure 5.3: Value of c_r versus r during RC4 PRGA ($N = 256$ and $3 \leq r \leq 255$).

Figure 5.4 depicts a comparison between the theoretical and experimental values of $\Pr(Z_r = 0)$ plotted against r, where $N = 256$ and $3 \leq r \leq 255$, and the experimentation is performed over 1 billion runs of RC4, each with a randomly generated 16-byte key.

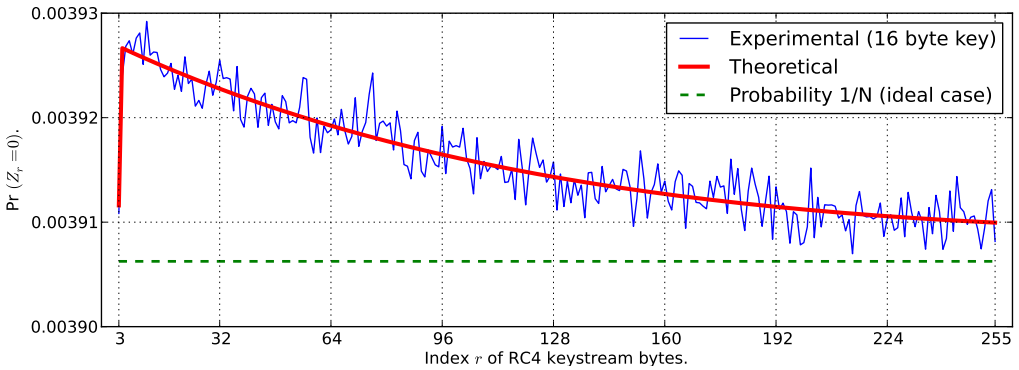

Figure 5.4: $\Pr(Z_r = 0)$ versus r during RC4 PRGA ($3 \leq r \leq 255$).

Let E_r denote the event $(Z_r = 0)$ for some $3 \leq r \leq 255$. If we write $p = 1/N$ and $q = c_r/N$, then to distinguish RC4 keystream from random sequence based on event E_r, one would need number of samples of the order of $(1/N)^{-1} \cdot (c_r/N)^{-2} \sim N^3$, which is quite high. However, a potential combination of these biases for $r = 3, \ldots, 255$ may be applicable towards an attack against broadcast RC4, as suggested in [98, 132]. In a recent work [5], the authors argued that the approach proposed in [98, 132] may be impractical. They proposed a new method [5] to combine all biases in $(Z_r = 0)$ for $r = 3, \ldots, 255$ towards a practical attack against broadcast RC4. The biases in $(Z_r = 0)$ have also been exploited recently in [70] to mount an independent plaintext-recovery attack on RC4 in broadcast mode.

In this section, we have contradicted MS-Claim-1 by proving the biases in $\Pr(Z_r = 0)$ for all $3 \leq r \leq 255$. If the supporting statement MS-Claim-2 was correct, then one would have a positive bias $\Pr(Z_r = 0 \mid j_r = 0) > \frac{1}{N}$. However, we have run extensive experiments to confirm that $\Pr(Z_r = 0 \mid j_r = 0) \approx \frac{1}{N}$, thereby contradicting MS-Claim-2 as well.

5.2.2 Guessing state information using the bias in Z_r

Mantin and Shamir [103] used the bias of the second byte of RC4 keystream to guess some information regarding $S_0[2]$, based on the following:

$$\Pr(S_0[2] = 0 \mid Z_2 = 0) = \frac{\Pr(S_0[2] = 0)}{\Pr(Z_2 = 0)} \cdot \Pr(Z_2 = 0 \mid S_0[2] = 0) \approx \frac{1/N}{2/N} \cdot 1 = \frac{1}{2}.$$

Note that in the above expression, no randomness assumption is required to obtain $\Pr(S_0[2] = 0) = 1/N$. This probability is exact and can be derived by substituting $u = 2, v = 0$ in Proposition 3.1. Hence, on every occasion we obtain $Z_2 = 0$ in the keystream, we can guess $S_0[2]$ with probability $1/2$, and this is significantly more than a random guess with probability $1/N$.

In this section, we use the biases in bytes 3 to 255 (observed in Theorem 5.5) to extract similar information about the state array S_{r-1} using the RC4 keystream byte Z_r. In particular, we try to explore the conditional probability $\Pr(S_{r-1}[r] = r \mid Z_r = 0)$ for $3 \leq r \leq 255$, as follows:

$$\Pr(S_{r-1}[r] = r \mid Z_r = 0) = \frac{\Pr(Z_r = 0 \wedge S_{r-1}[r] = r)}{\Pr(Z_r = 0)} \approx \frac{\Pr(S_{r-1}[r] = r) \cdot \frac{2}{N}}{\frac{1}{N} + \frac{c_r}{N^2}}.$$

In the above expression, c_r is as in Theorem 5.5, and one may write:

$$\Pr(S_{r-1}[r] = r)$$
$$= \begin{cases} 1/N + (1/N - 1/N^2) \cdot (c_r + (N-2)/(N-1)), & \text{for } r = 3; \\ 1/N + (1/N - 1/N^2) \cdot c_r, & \text{for } 3 < r \leq N-1. \end{cases}$$

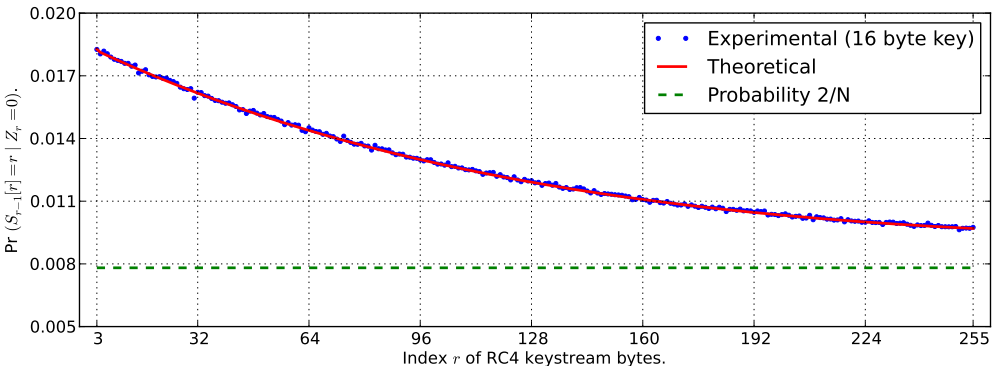

Figure 5.5: $\Pr(S_{r-1}[r] = r \mid Z_r = 0)$ versus r during RC4 PRGA ($3 \leq r \leq 255$).

In Figure 5.5, we plot the theoretical values of $\Pr(S_{r-1}[r] = r \mid Z_r = 0)$ for $3 \leq r \leq 255$ and $N = 256$, and the corresponding experimental values over 1 billion runs of RC4 with random 16-byte keys. It clearly shows that all values of $\Pr(S_{r-1}[r] = r \mid Z_r = 0)$ for $N = 256$ and $3 \leq r \leq 255$ (both theoretical and experimental) are greater than $2/N$. Thus, one can guess $S_{r-1}[r]$ with

probability more than twice of that of a random guess, every time we obtain $Z_r = 0$ in the keystream.

Remark 5.7. In proving Corollary 4.4, we use the initial condition $S_1[r] = r$ to branch out the probability paths, and not $S_0[r] = r$ as in [98, Lemma 1]. This is because the probability of $S[r] = r$ takes a leap from around $1/N$ in S_0 to about $2/N$ in S_1, and this turns out to be the actual cause behind the bias in $S_{r-1}[r] = r$. Consideration of this issue eventually corrects the mismatches observed in the graphs of [98, Figure 2 and Figure 3]. Note that Theorem 5.5, Figure 5.4 and Figure 5.5 are, respectively, the corrected versions of [98, Theorem 1, Figure 2 and Figure 3], and are also presented in [132].

5.3 Proof of some isolated short-term biases

In this section, we prove all open isolated short-term biases recently identified through extensive experimentation by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5] and Isobe, Ohigashi, Watanabe and Morii [70]. The case of $Z_1 = 129$ is related to the length of the secret key and the 'anomaly pairs', and hence we treat it separately in Chapter 3. In this section, we present our results from [128], as summarized in Table 5.1.

Table 5.1: Proved short-term single-byte keystream biases of RC4.

Biased event	Discovered	Theory [128]	Experiment [5,14,70]
$Z_2 = 129$	[5, 127]	$1/N - 2/N^2$	$1/N - 1.82/N^2$
$Z_2 = 172$	[5]	$1/N + 0.28/N^2$	$1/N + 0.2/N^2$
$Z_4 = 2$	[5]	$1/N + 1/N^2$	$1/N + 0.8/N^2$
$Z_{256} = 0$	[5, 70]	$1/N - 0.36/N^2$	$1/N - 0.38/N^2$
$Z_{257} = 0$	[70]	$1/N + 0.36/N^2$	$1/N + 0.35/N^2$

5.3.1 Proof of bias in $(Z_2 = 129)$

We notice that the bias in $(Z_2 = 129)$ for $N = 256$ is a special case of the general bias in $(Z_2 = N/2+1)$ for any even value of N. We present the general result as follows.

Theorem 5.8. *Suppose that the initial permutation S_0 of RC4 PRGA is randomly chosen from the set of all permutations of $\{0, 1, \ldots, N-1\}$, where N is even. Then* $\Pr(Z_2 = N/2 + 1) \approx 1/N - 2/N^2$.

Proof. We consider two mutually exclusive paths from the initial state S_0.

Path 1: Consider $S_0[2] = 0$ and $S_0[1] \neq 2$. From the analysis of Mantin and Shamir [103] for the bias in $(Z_2 = 0)$, we know that $Z_2 = 0$ in this situation. Thus, $Z_2 \neq N/2 + 1$.

Path 2: Consider $S_0[2] = N/2 + 1$ and $S_0[1] \neq 2$. After the first round, $j_1 = S_0[1] = X \neq 2$, and thus $S_1[2] = N/2 + 1$ and $S_1[X] = X$. In the second round, we get $j_2 = (N/2 + 1) + X$, and let us say $S_1[j_2] = S_1[(N/2 + 1) + X] = Z$. Since S_1 is a permutation, $X = S_1[X] \neq S_1[(N/2 + 1) + X] = Z$. After the swap in the second round, we get $Z_2 = S_2[(N/2 + 1) + Z] \neq S_2[(N/2 + 1) + X] = N/2 + 1$. Figure 5.6 illustrates the scenario.

We denote the above mutually exclusive events as $A = (S_0[2] = 0 \wedge S_0[1] \neq 2)$ and $B = (S_0[2] = N/2 + 1 \wedge S_0[1] \neq 2)$ to obtain $\Pr(Z_2 = N/2 + 1)$ as follows.

$$\begin{aligned}
&\Pr(Z_2 = N/2 + 1 \mid A) \cdot \Pr(A) + \Pr(Z_2 = N/2 + 1 \mid B) \cdot \Pr(B) \\
&\quad + \Pr(Z_2 = N/2 + 1 \mid \overline{A} \wedge \overline{B}) \cdot \Pr(\overline{A} \wedge \overline{B}) \\
&\approx 0 + 0 + \Pr(Z_2 = N/2 + 1 \mid \overline{A} \wedge \overline{B}) \cdot (1 - 2/N).
\end{aligned}$$

Assuming $\Pr(Z_2 = N/2 + 1 \mid \overline{A} \wedge \overline{B}) \approx 1/N$ due to random association, we get $\Pr(Z_2 = N/2 + 1) \approx (1/N) \cdot (1 - 2/N) = 1/N - 2/N^2$. □

5.3.2 Proof of bias in $(Z_2 = 172)$

We shall assume the probabilistic model of uniform random keys to prove this result. We assume actual RC4 next-state-function for the evolution of S and i, j, and no randomness assumptions are made on the initial state S_0 of PRGA.

Theorem 5.9. *In RC4 with $N = 256$,* $\Pr(Z_2 = 172) \approx 1/N + 0.28/N^2$.

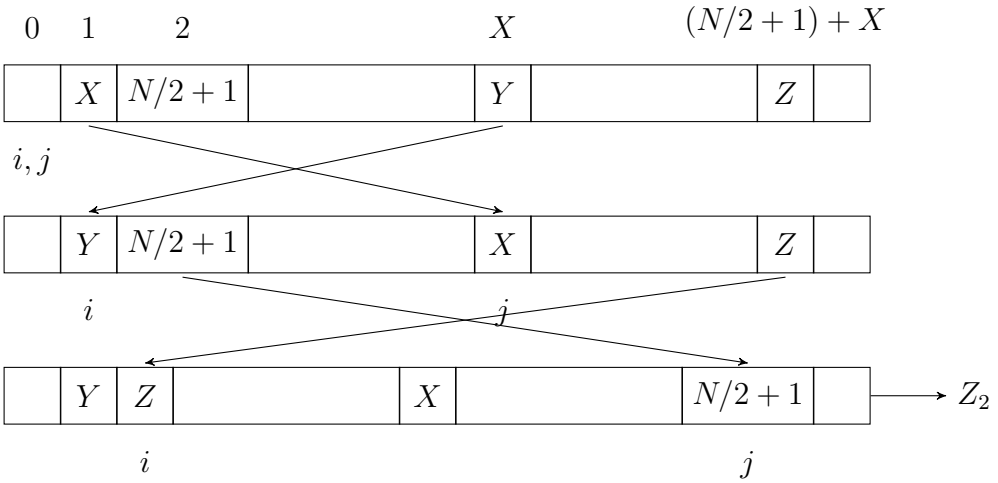

Figure 5.6: First two rounds of PRGA when $S_0[2] = N/2 + 1$ and $S_0[1] \neq 2$.

Proof. We consider following mutually exclusive paths from initial state S_0.

Path 1. Consider $S_0[2] = 0$. If $S_0[1] \neq 2$, from the second-byte bias of Mantin and Shamir [103], we know that $Z_2 = 0$ in this situation. Thus, $Z_2 \neq 172$. In case $S_0[1] = 2$, we may assume that $Z_2 = 172$ occurs with probability $1/N$. Thus, $\Pr(Z_2 = 172 \mid S_0[2] = 0) \approx 1/N^2$.

Path 2. Consider $S_0[2] = 86$. In this case, we have the following sub-paths.

1. Consider $S_0[1] = 172$. In this case, $j_1 = S_0[1] = 172$ results in a swap to produce $S_1[172] = 172$, while $S_1[2] = 86$ remains untouched. In the next round, $j_2 = j_1 + S_1[2] = 172 + 86 = 258 = 2 = i_2$ ensures that there is no swap in the S-array. Thus, $Z_2 = S_2[S_2[i_2] + S_2[j_2]] = S_1[86 + 86] = S_1[172] = 172$. Note that this path is possible for any X is $S_0[1] = X$ and $S_0[2] = X/2$, and if $X + X/2 = 2$. Thus, this path results in the modular equation $3X \equiv 4 \bmod N$, which has a unique solution $X = 172$ for $N = 256$.

2. Consider $S_0[1] \neq 172$ and $S_0[S_0[1] + 86] = 172$. In the first round, $S_1[2] = 86$ remains untouched, and $j_2 = j_1 + S_1[2] = S_0[1] + 86$ results in a swap to produce $S_2[2] = S_1[j_2] = S_1[S_0[1] + 86] = S_0[S_0[1] + 86] = 172$ and $S_2[S_0[1] + 86] = 86$. Thus, in the second round, we get $Z_2 = S_2[S_2[i_2] + S_2[j_2]] = S_2[172 + 86] = S_2[2] = 172$. Figure 5.7 illustrates the scenario.

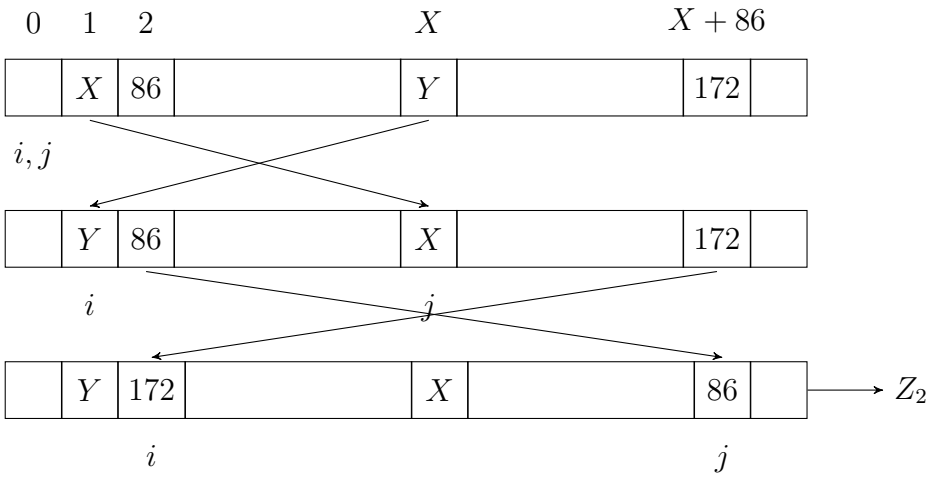

Figure 5.7: The first two rounds of RC4 main cycle when $S_0[2] = 86$, $S_0[1] \neq 2, 172$ and $S_0[S_0[1] + 86] = 172$.

Let us denote the aforesaid events as $B = (S_0[2] = 86)$, $C = (S_0[1] = 172)$, and $D = (S_0[S_0[1] + 86] = 172)$. This results in

$$\begin{aligned}
&\Pr(Z_2 = 172 \mid S_0[2] = 86) = \Pr(Z_2 = 172 \mid B) \\
&= \Pr(Z_2 = 172 \mid B \wedge C) \cdot \Pr(C) + \Pr(Z_2 = 172 \mid B \wedge \overline{C}) \cdot \Pr(\overline{C}) \\
&\approx 1 \cdot (1/N) + \Big(\Pr(Z_2 = 172 \mid B \wedge \overline{C} \wedge D) \cdot \Pr(D) \\
&\qquad + \Pr(Z_2 = 172 \mid B \wedge \overline{C} \wedge \overline{D}) \cdot \Pr(\overline{D})\Big) \cdot (1 - 1/N) \\
&\approx (1/N) + (1 \cdot (1/N) + (1/N) \cdot (1 - 1/N)) \cdot (1 - 1/N) \approx 3/N - 3/N^2.
\end{aligned}$$

Path 3. Considering $S_0[2] = 172$, $Z_2 = 172$ if $S_0[1] = 2$ and $S_0[4] = N-1$, and in all other cases, $Z_2 \neq 172$. Thus, $\Pr(Z_2 = 172 \mid S_0[2] = 172) \approx 1/N^2$.

Let us combine the aforesaid paths to obtain $\Pr(Z_2 = 172)$ as

$$\begin{aligned}
&\Pr(Z_2 = 172 \mid S_0[2] = 0) \cdot \Pr(S_0[2] = 0) \\
&\quad + \Pr(Z_2 = 172 \mid S_0[2] = 86) \cdot \Pr(S_0[2] = 86) \\
&\quad + \Pr(Z_2 = 172 \mid S_0[2] = 172) \cdot \Pr(S_0[2] = 172) \\
&\approx (1/N^2) \cdot \Pr(S_0[2] = 0) + (3/N - 3/N^2) \cdot \Pr(S_0[2] = 86) \\
&\quad + (1/N^2) \cdot \Pr(S_0[2] = 172).
\end{aligned}$$

In the above equation, computing the probability terms involving S_0 using the

formula of Mantin [100], we get $\Pr(Z_2 = 172) \approx 1/N + 0.28/N^2$. □

5.3.3 Proof of bias in $(Z_4 = 2)$

Theorem 5.10. *Suppose that the initial permutation S_0 of RC4 PRGA is randomly chosen from the set of all permutations of $\{0, 1, \ldots, N-1\}$, where $N = 256$. Then $\Pr(Z_4 = 2) \approx 1/N + 1/N^2$.*

Proof. We observe the main paths for this bias as follows.

Path 1. Consider $j_4 = 4$. Then, $Z_4 = S_4[S_4[4] + S_4[j_4]] = S_4[2 \cdot S_4[4]]$. We may further consider some sub-paths within this case.

Subpath 1: $S_4[4] = 2$ gives $Z_4 = S_4[4] = 2$ with probability 1. However, the event $(S_4[4] = 2 \mid j_4 = 4)$ occurs with probability approximately $2/N$, as follows.

- If $S_0[4] = 2$ and $j_1, j_2, j_3 \neq 4$, then $S_4[4] = 2$ is fixed during first three rounds. Thus, $\Pr(S_4[4] = 2 \mid j_4 = 4 \wedge S_0[4] = 2) \approx 1$.
- If $S_0[1] = 2$ and $j_3 \neq 4$, it can be shown that $S_4[4] = 2$ in first three rounds. Thus, $\Pr(S_4[4] = 2 \mid j_4 = 4 \wedge S_0[1] = 2) \approx 1$.
- Consider $S_0[4] \neq 2$ and $S_0[1] \neq 2$. In this case, we show that $S_4[4] = 2$ and $j_4 = 4$ can not occur simultaneously. Suppose the event $(j_4 = 4 \wedge S_4[4] = 2)$ does occur. Then we have $j_3 = 0$, and hence $S_2[4] = S_3[4] = S_4[4] = 2$. As we know $S_0[4] \neq 2$, this implies $j_1 = 4$ and/or $j_2 = 4$. If $j_1 = 4$ and $j_2 = 4$, we get $S_2[4] = 0$, contradiction, as $S_2[4] = 2$. If $j_1 = 4$ and $j_2 \neq 4$, we get $S_1[4] = S_2[4] = 4$, contradiction, as $S_2[4] = 2$. Consider $j_1 \neq 4$ and $j_2 = 4$. Since $j_2 = S_0[1] + S_1[2]$ and $S_0[1] \neq 2$, we must have $S_2[4] = S_1[2] \neq 2$, a contradiction.

In summary, $\Pr(S_4[4] = 2 \mid j_4 = 4) \approx 2/N$, and $\Pr(Z_4 = 2 \wedge S_4[4] = 2 \mid j_4 = 4) \approx 2/N$.

Subpath 2: $S_4[4] = N/2 + 2$ gives $Z_4 = S_4[N + 4] = S_4[4] = N/2 + 2 \neq 2$. So, $\Pr(Z_4 = 2 \wedge S_4[4] = N/2 + 2 \mid j_4 = 4) = 0$.

Subpath 3: $S_4[4] = 0$ gives $Z_4 = S_4[0] = 2$ with probability $1/N$. However, the event $(S_4[4] = 0 \mid j_4 = 4)$ occurs with probability $2/N$:

- If $S_0[1] = 2$ and $S_0[3] = 0$, then $j_1 = 2$, $S_1[2] = S_0[1] = 2$, which implies $j_2 = 4$. This produces $S_2[4] = S_1[2] = 2$, and we get $j_3 = j_2 = 4$ because of $S_2[3] = S_1[3] = S_0[3] = 0$. After the third round, $S_3[4] = S_2[3] = 0$, and in the next round, $j_4 = j_3 = 4$ ensures no swap. Thus, we get both $j_4 = 4$ and $S_4[4] = S_3[4] = 0$.
- In other situations, $S_4[4] = 0$ and $j_4 = 4$ occur randomly.

In summary,

$$\Pr(S_4[4] = 0 \mid j_4 = 4) \approx 2/N, \quad \text{and}$$
$$\Pr(Z_4 = 2 \wedge S_4[4] = 0 \mid j_4 = 4) \approx 2/N^2.$$

Subpath 4: $S_4[4] \neq 0, 2, N/2 + 2$ gives $Z_4 = 2$ with probability approximately $1/N$ due to random association. Due to the previous subpaths, we know that the event $(S_4[4] \neq 0, 2, N/2 + 2 \mid j_4 = 4)$ occurs with probability $(1 - 5/N)$. Thus,

$$\Pr(Z_4 = 2 \wedge S_4[4] \neq 0, 2, N/2 + 2 \mid j_4 = 4) \approx (1/N) \cdot (1 - 5/N).$$

Combining all subpaths mentioned above, we get $\Pr(Z_4 = 2 \wedge j_4 = 4)$ as

$$\begin{aligned}
&\Pr(Z_4 = 2 \wedge S_4[4] = 2 \mid j_4 = 4) \cdot \Pr(j_4 = 4) \\
&\quad + \Pr(Z_4 = 2 \wedge S_4[4] = N/2 + 2 \mid j_4 = 4) \cdot \Pr(j_4 = 4) \\
&\quad + \Pr(Z_4 = 2 \wedge S_4[4] = 0 \mid j_4 = 4) \cdot \Pr(j_4 = 4) \\
&\quad + \Pr(Z_4 = 2 \wedge S_4[4] \neq 0, 2, N/2 + 2 \mid j_4 = 4) \cdot \Pr(j_4 = 4) \\
&= (2/N) \cdot (1/N) + 0 + (2/N^2) \cdot (1/N) + (1/N - 5/N^2) \cdot (1/N) \\
&= 3/N^2 - 3/N^3.
\end{aligned}$$

Path 2. Consider $j_4 \neq 4$. Then, $Z_4 = S_4[S_4[4] + S_4[j_4]] = S_4[S_4[4] + X]$, where $X = S_4[j_4] \neq S_4[4]$, say. Here we may consider two subpaths, as follows.

Subpath 1: $S_4[4] = 2$ gives $Z_4 = S_4[2 + X] \neq S_4[4] = 2$, as $X = S_4[j_4] \neq S_4[4] = 2$ for $j_4 = 4$. Thus we get $\Pr(Z_4 = 2 \wedge S_4[4] = 2 \mid j_4 \neq 4) = 0$.

Subpath 2: $S_4[4] \neq 2$ gives $Z_4 = 2$ with due to random association. Thus we get $\Pr(Z_4 = 2 \wedge S_4[4] \neq 2 \mid j_4 \neq 4) \approx 1/N \cdot (1 - 1/N) = (1/N - 1/N^2)$.

Combining the aforesaid subpaths, we have $\Pr(Z_4 = 2 \wedge j_4 \neq 4)$ as

$$\begin{aligned}
&\Pr(Z_4 = 2 \wedge S_4[4] = 2 \mid j_4 \neq 4) \cdot \Pr(j_4 \neq 4) \\
&\quad + \Pr(Z_4 = 2 \wedge S_4[4] \neq 2 \mid j_4 \neq 4) \cdot \Pr(j_4 \neq 4) \\
&= 0 + (1/N - 1/N^2) \cdot (1 - 1/N) = 1/N - 2/N^2 + 1/N^3.
\end{aligned}$$

Adding the contributions from the two mutually exclusive paths above, we get the desired probability $\Pr(Z_4 = 2)$ computed as

$$\begin{aligned}
&\Pr(Z_4 = 2) \\
&= \Pr(Z_4 = 2 \wedge j_4 = 4) + \Pr(Z_4 = 2 \wedge j_4 \neq 4) \\
&= (3/N^2 - 3/N^3) + (1/N - 2/N^2 + 1/N^3) = 1/N + 1/N^2 - 2/N^3.
\end{aligned}$$

Thus we obtain $\Pr(Z_4 = 2) \approx 1/N + 1/N^2$. □

5.3.4 Proof of bias in $(Z_{256} = 0)$

We shall assume the probabilistic model of uniform random keys to prove this result. We assume actual RC4 next-state-function for the evolution of S and i, j, and no randomness assumptions are made on the initial state S_0 of PRGA.

Theorem 5.11. *In RC4 with* $N = 256$, $\Pr(Z_N = 0) \approx 1/N - 0.36/N^2$.

Proof. Let us consider the following two paths.

Path 1. Consider $S_1[0] = 0$. In this case, if $j_2, \ldots, j_{N-1}$ are all non zero, then one can check that $Z_N \neq 0$. In all other cases, one may consider $\Pr(Z_N = 0 \mid S_1[0] = 0) \approx 1/N$ due to random association. Thus, $\Pr(Z_N = 0 \mid S_1[0] = 0) \approx \left(1 - (1 - 1/N)^{N-2}\right) \cdot (1/N)$.

Path 2. Consider $S_1[0] \neq 0$. We may consider following sub-paths, depending

on state S_{N-3}. Denote $A := Z_N = 0$, $B := S_1[0] \neq 0$ to get:

$$\begin{aligned}
&\Pr(Z_N = 0 \mid S_1[0] \neq 0) = \Pr(A \mid B) \\
&= \Pr(A \mid B \wedge S_{N-3}[0] = 0) \cdot \Pr(S_{N-3}[0] = 0 \mid S_1[0] \neq 0) \\
&\quad + \Pr(A \mid B \wedge S_{N-3}[N-2] = 0) \cdot \Pr(S_{N-3}[N-2] = 0 \mid S_1[0] \neq 0) \\
&\quad + \Pr(A \mid B \wedge S_{N-3}[N-1] = 0) \cdot \Pr(S_{N-3}[N-1] = 0 \mid S_1[0] \neq 0) \\
&\quad + \sum_{x=1}^{N-3} \Pr(A \mid B \wedge S_{N-3}[x] = 0) \cdot \Pr(S_{N-3}[x] = 0 \mid S_1[0] \neq 0).
\end{aligned}$$

Case 1: If $S_{N-3}[0] = 0$ and $j_{N-2}, j_{N-1} \neq 0$, we have $S_{N-1}[0] = 0$, which implies $j_N = j_{N-1}$ and $S_{N-1}[j_{N-1}] \neq j_{N-1}$. Thus, $Z_N = S_N[S_{N-1}[j_N] + S_{N-1}[0]] = S_N[S_{N-1}[j_{N-1}]] \neq S_N[j_{N-1}] = S_N[j_N] = S_{N-1}[0] = 0$. Thus for $Z_N = 0$, we must have either $j_{N-2} = 0$ or $j_{N-1} = 0$ in this case, and in each case, $Z_N = 0$ will occur with probability $1/N$ of random association. Hence $\Pr(Z_N = 0 \mid S_1[0] \neq 0 \wedge S_{N-3}[0] = 0) \approx 2/N^2$.

Case 2: If $S_{N-3}[N-2] = 0$ and $j_{N-2} = 0$, we have $S_{N-2}[0] = 0$ and $j_{N-1} = S_{N-2}[N-1] \neq 0$. Thus, $S_{N-1}[0] = 0$ and $j_N = j_{N-1}$, which gives $Z_N = S_N[S_{N-1}[0] + S_{N-1}[j_N]] = S_N[S_{N-1}[j_{N-1}]] = S_N[S_{N-2}[N-1]] = S_N[j_{N-1}] = S_N[j_N] = S_{N-1}[0] = 0$. So, $\Pr(Z_N = 0 \mid S_1[0] \neq 0 \wedge S_{N-3}[N-2] = 0 \wedge j_{N-2} = 0) = 1$. On the other hand, if $S_{N-3}[N-2] = 0$ and $j_{N-2} \neq 0$, then $Z_N \neq 0$ where $j_{N-1} \neq 0$ and $S_{N-1}[j_N] = 0$, and $Z_N = 0$ due to random association in all other cases. So, $\Pr(Z_N = 0 \mid S_1[0] \neq 0 \wedge S_{N-3}[N-2] = 0 \wedge j_{N-2} \neq 0) \approx 1/N - 1/N^2$. Combining the two items as above, we get

$$\Pr(Z_N = 0 \mid S_1[0] \neq 0 \wedge S_{N-3}[N-2] = 0) \approx 2/N - 2/N^2.$$

Case 3: Similarly for $S_{N-3}[N-1] = 0$, it can be proved that $\Pr(Z_N = 0 \mid S_1[0] \neq 0 \wedge S_{N-3}[N-1] = 0) \approx 2/N - 2/N^2$.

Case 4: Now consider the case $S_{N-3}[x] = 0$ for $1 \leq x \leq N-3$. If $j_{N-2} \neq x$, $j_{N-1} \neq x$ and $j_N = x$, one can verify that $Z_N \neq 0$. In all other cases, $Z_N = 0$ occurs with probability $1/N$. Thus for $1 \leq x \leq N-3$, $\Pr(Z_N = 0 \mid S_1[0] \neq 0 \wedge S_{N-3}[x] = 0) \approx 1/N - 1/N^2$.

Now, let us consider the conditional events $(S_{N-3}[x] = 0 \mid S_1[0] \neq 0)$,

for $0 \leq x \leq N-1$, to complete the picture. Starting with $S_1[0] \neq 0$, if $j_2, \ldots, j_{N-3}$ are all non zero, we have $S_{N-3}[0] \neq 0$ as well. So, $\Pr(S_{N-3}[0] = 0 \mid S_1[0] \neq 0) = \left(1 - (1-1/N)^{N-4}\right) \cdot (1/N) = P_A$, say. For all $x \neq 0$, we may now assume $\Pr(S_{N-3}[x] = 0 \mid S_1[0] \neq 0) \approx (1 - P_A)/(N-1) = P_B$, say. Taking into account the contributions from all four sub-cases within this path, we get

$$\begin{aligned}
&\Pr(Z_N = 0 \mid S_1[0] \neq 0) \\
&= (2/N^2) \cdot P_A + (2/N - 2/N^2) \cdot P_B \\
&\quad + (2/N - 2/N^2) \cdot P_B + (1/N - 1/N^2) \cdot (1 - P_A - 2P_B) \\
&= (1/N - 1/N^2) - (1/N - 3/N^2) \cdot P_A + (2/N - 2/N^2) \cdot P_B.
\end{aligned}$$

Combining the above two paths, we get $\Pr(Z_N = 0)$ as

$$\begin{aligned}
&\Pr(Z_N = 0 \mid S_1[0] = 0) \cdot P(S_1[0] = 0) + \Pr(Z_N = 0 \mid S_1[0] \neq 0) \cdot P(S_1[0] \neq 0) \\
&\approx \left(1 - (1-1/N)^{N-2}\right) \cdot (1/N) \cdot (2/N) \\
&\quad + ((1/N - 1/N^2) - (1/N - 3/N^2) \cdot P_A \\
&\quad + (2/N - 2/N^2) \cdot P_B) \cdot (1 - 2/N).
\end{aligned}$$

For $N = 256$, as in the case with practical RC4, this result produces the value $\Pr(Z_N = 0) \approx 1/N - 0.36/N^2$. □

5.3.5 Proof of bias in $(Z_{257} = 0)$

We shall assume the probabilistic model of uniform random keys to prove this result. We assume actual RC4 next-state-function for the evolution of S and i, j, and no randomness assumptions are made on the initial state S_0 of PRGA.

Theorem 5.12. *In RC4 with $N = 256$, $\Pr(Z_{N+1} = 0) \approx 1/N + 0.36/N^2$.*

Proof. We write $Z_{N+1} = S_{N+1}[S_N[1] + S_N[j_{N+1}]]$, and consider following paths.

Path 1. Consider the case $S_N[1] = 1$, where we may write $Z_{N+1} = S_{N+1}[1 + S_N[j_{N+1}]]$. If $S_N[j_{N+1}] = 0$, we have $Z_{N+1} = S_{N+1}[1] = S_N[j_{N+1}] = 0$. Otherwise if $S_N[j_{N+1}] = X \neq 0$, we have $Z_{N+1} = S_{N+1}[1 + X] = 0$ only

due to random association. Let us denote events $A = (S_N[1] = 1)$ and $B = (S_N[j_{N+1}] = 0)$ to get

$$\begin{aligned}
&\Pr(Z_{N+1} = 0 \mid A) \\
&= \Pr(Z_{N+1} = 0 \mid A \wedge B) \cdot \Pr(B) + \Pr(Z_{N+1} = 0 \mid A \wedge \overline{B}) \cdot \Pr(\overline{B}) \\
&\approx 1 \cdot (1/N) + (1/N) \cdot (1 - 1/N) = 2/N - 1/N^2.
\end{aligned}$$

Path 2. Consider the case $S_N[1] = X \neq 1$. Here we have $Z_{N+1} = S_{N+1}[X + S_N[j_{N+1}]]$. If $S_N[j_{N+1}] = 0$, we will get $Z_{N+1} = S_{N+1}[X] \neq S_{N+1}[1] = S_N[j_{N+1}] = 0$. Otherwise, for $S_N[j_{N+1}] = Y \neq 0$, we may have $Z_{N+1} = S_{N+1}[X + Y] = 0$ due to random association. Let us denote events $A = (S_N[1] = 1)$ and $B = (S_N[j_{N+1}] = 0)$ to get

$$\begin{aligned}
&\Pr(Z_{N+1} = 0 \mid \overline{A}) \\
&= \Pr(Z_{N+1} = 0 \mid \overline{A} \wedge B) \cdot \Pr(B) + \Pr(Z_{N+1} = 0 \mid \overline{A} \wedge \overline{B}) \cdot \Pr(\overline{B}) \\
&\approx 0 + (1/N) \cdot (1 - 1/N) = 1/N - 1/N^2.
\end{aligned}$$

From [132, Theorem 1], we get $\Pr(S_N[1] = 1) \approx 0.00532$ for $N = 256$. Thus,

$$\begin{aligned}
&\Pr(Z_{N+1} = 0) \\
&= \Pr(Z_{N+1} = 0 \mid A) \cdot \Pr(A) + \Pr(Z_{N+1} = 0 \mid \overline{A}) \cdot \Pr(\overline{A}) \\
&\approx (2/N - 1/N^2) \cdot (0.00532) + (1/N - 1/N^2) \cdot (1 - 0.00532).
\end{aligned}$$

For $N = 256$, as in the case with practical RC4, this produces the value $\Pr(Z_{N+1} = 0) \approx 1/N + 0.36/N^2$. □

5.4 Periodic long-term bias in RC4

The biases discussed so far are prevalent in the initial bytes of the RC4 keystream, and are generally referred to as the short-term biases. It is a common practice to discard a few hundred initial bytes of the keystream to avoid these biases, and this motivates the search for long-term biases in RC4 that are present even after discarding an arbitrary number of initial bytes.

The first result in this direction was observed in 1997 by Golic [52], where certain correlation was found between the least significant bits of the two non-consecutive output bytes Z_r and Z_{r+2}, for all rounds r of RC4. In 2000, a set of results was proposed by Fluhrer and McGrew [45], where the biases depend upon the frequency of occurrence of certain digraphs in the RC4 keystream. Later in 2005, Mantin [102] improved these to obtain the $AB\mathcal{S}AB$ distinguisher, which depends on the repetition of digraph AB in the keystream after a gap of string $\mathcal{S}$ having G bytes. This is the best long-term distinguisher of RC4 to date. In 2008, Basu et al. [12] identified another conditional long-term bias, depending on the relationship between two consecutive bytes in the keystream.

In this section, we prove that the event $(Z_{wN+2} = 0 \ \wedge \ Z_{wN} = 0)$ is positively biased for all $w \geq 1$. After the first long-term bias observed by Golic [52] in 1997, this is the only one that involves non-consecutive bytes of RC4 keystream. Golic [52] proved a strong bitwise (only least significant bit) correlation between Z_{wN} and Z_{wN+2}, while we prove a byte-wise correlation.

Theorem 5.13. *For any integer $w \geq 1$, assume that the permutation S_{wN} is randomly chosen from the set of all possible permutations of $\{0, \ldots, N-1\}$. Then $\Pr(Z_{wN+2} = 0 \ \wedge \ Z_{wN} = 0) \approx 1/N^2 + 1/N^3$.*

Proof. The positive bias in Z_2, proved in [103], propagates to round $(wN+2)$ if $j_{wN} = 0$. Mantin and Shamir's observation [103, Theorem 1] implies

$$\Pr(Z_{wN+2} = 0 \mid j_{wN} = 0) \approx 2/N - 1/N^2. \tag{5.7}$$

If $j_{wN} \neq 0$, we observe that Z_{wN+2} does not take the value 0 by uniform random association. In particular, we get the following:

$$\Pr(Z_{wN+2} = 0 \mid j_{wN} \neq 0) \approx 1/N - 1/N^2. \tag{5.8}$$

For Z_{wN}, we have $i_{wN} = 0$, and when $j_{wN} = 0$ (this happens with probability $1/N$), no swap takes place and the output is $Z_{wN} = S_{wN}[2 \cdot S_{wN}[0]]$. Two cases may arise from here. If $S_{wN}[0] = 0$, then $Z_{wN} = S_{wN}[0] = 0$ for sure. Otherwise if $S_{wN}[0] \neq 0$, the output Z_{wN} takes the value 0 only due to random

association. Combining the cases,

$$\Pr(Z_{wN} = 0 \mid j_{wN} = 0) \approx 1/N \cdot 1 + (1 - 1/N) \cdot 1/N = 2/N - 1/N^2. \quad (5.9)$$

Similarly to $\Pr(Z_{wN+2} = 0 \mid j_{wN} \neq 0)$, it is easy to show that

$$\Pr(Z_{wN} = 0 \mid j_{wN} \neq 0) \approx 1/N - 1/N^2. \quad (5.10)$$

Now, we may compute the joint probability $\Pr(Z_{wN+2} = 0 \wedge Z_{wN} = 0)$ as

$$\Pr(Z_{wN+2} = 0 \wedge Z_{wN} = 0 \wedge j_{wN} = 0) + \Pr(Z_{wN+2} = 0 \wedge Z_{wN} = 0 \wedge j_{wN} \neq 0).$$

Given $j_{wN} = 0$, bytes Z_{wN+2} and Z_{wN} can be considered independent. Using Equations (5.7) and (5.9), we get $\Pr(Z_{wN+2} = 0 \wedge Z_{wN} = 0 \wedge j_{wN} = 0)$ as

$$\begin{aligned}&\Pr(Z_{wN+2} = 0 \mid j_{wN} = 0) \cdot \Pr(Z_{wN} = 0 \mid j_{wN} = 0) \cdot \Pr(j_{wN} = 0)\\ &\approx (2/N - 1/N^2) \cdot (2/N - 1/N^2) \cdot (1/N) \approx 4/N^3 - 4/N^4.\end{aligned}$$

From Equations (5.8), (5.10), we get $\Pr(Z_{wN+2} = 0 \wedge Z_{wN} = 0 \wedge j_{wN} \neq 0)$ as

$$\begin{aligned}&\Pr(Z_{wN+2} = 0 \mid j_{wN} \neq 0) \cdot \Pr(Z_{wN} = 0 \mid j_{wN} \neq 0) \cdot \Pr(j_{wN} \neq 0)\\ &\approx (1/N - 1/N^2)^2 \cdot (1 - 1/N) \approx 1/N^2 - 3/N^3 + 3/N^4.\end{aligned}$$

Adding the two expressions, $\Pr(Z_{wN+2} = 0 \wedge Z_{wN} = 0) \approx 1/N^2 + 1/N^3$. □

This is the *first long-term byte-wise correlation* to be observed between *two non-consecutive bytes* (Z_{wN}, Z_{wN+2}). The gap between the related bytes in this case is 1, and we could not find any other significant long-term bias with this gap. An interesting direction for experimentation and analysis would be to look for similar long-term biases with larger gaps between the related bytes.

THIS PAGE INTENTIONALLY LEFT BLANK

Part II

Implementation of RC4 Stream Cipher

Chapter 6

Overview of RC4 Implementation

In this part of the thesis, we study several aspects of the hardware implementation of RC4 stream cipher, with respect to its high-throughput implementation. We present two new hardware designs in Chapters 7 and 8 that allow fast generation of RC4 keystream. The better of the two is the *fastest* known hardware implementation of the cipher till date. To motivate our contribution, we first discuss the current literature on RC4 hardware implementations.

6.1 Existing hardware implementations

We perform a survey of RC4 hardware implementations with a focus on its throughput efficiency, where the throughput is defined as the average number of cycles required to generate each byte of output keystream of the cipher. Thus, throughout this thesis, we shall refer to the throughput in terms of 'cycles-per-byte' or 'bytes-per-cycle' keystream generation rate of RC4.

In 2003, a 3 cycles-per-byte implementation of RC4 on a custom pipelined hardware was proposed by Kitsos, Kostopoulos, Sklavos and Koufopavlou [79]. In the same year, a patent by Matthews Jr. [106] was disclosed, which provided a similar 3 cycles-per-byte architecture using multi-port memory units. Another patent by Matthews Jr. [105] was disclosed in 2008, which proposed a new design for RC4 hardware using pipeline architecture. This increased the efficiency of the cipher to obtain a 1 byte-per-cycle throughput.

6.1.1 Kitsos et al custom pipeline design [79]

In 2003, Kitsos, Kostopoulos, Sklavos and Koufopavlou [79] proposed a custom pipelined architecture for RC4, as in Figure 6.1 (originally [79, Figure 3]).

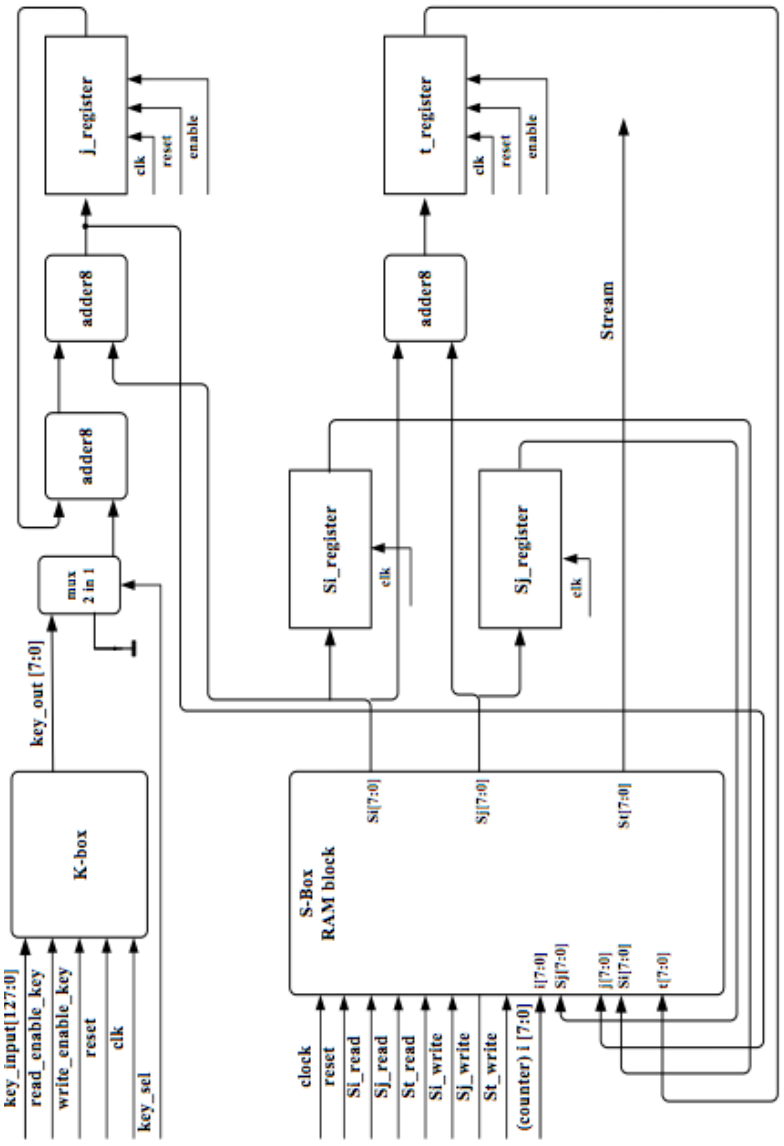

Figure 6.1: RC4 architecture proposed by Kitsos et al [79, Figure 3].

The storage unit for the design proposed in [79] is based on a three-part block RAM for the S-box, and a control unit is designed to synchronize the operations of the storage architecture. The design provides a throughput of N keystream bytes in $3N + 768$ cycles, that is, 3 cycles-per-byte for large N.

6.1.2 Matthews Jr. multi-port memory units [106]

In 2003, a US patent published by Matthews Jr. [106] proposed a hardware pipelined method for RC4 implementation using a circuit that includes at least one dual port memory, as shown in Figure 6.2 (originally [106, Fig. 6]).

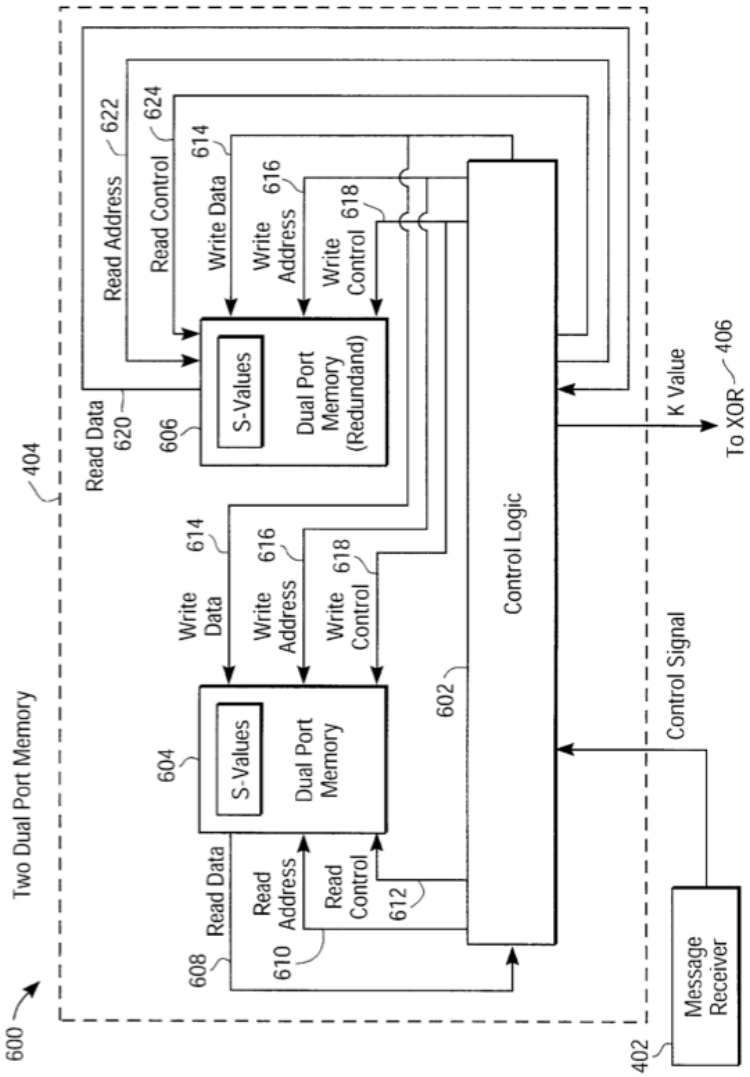

Figure 6.2: RC4 architecture proposed by Matthews Jr. [106, Fig. 6].

The design of RC4 hardware in [106] based on multi-port memory modules provides a throughput of 3 cycles-per-byte if a single memory unit is used. The throughput may be further improved to 2 cycles-per-byte if more multi-port memory units are used simultaneously. However, the generic throughput of the design, 3 cycles-per-byte, is same as that of Kitsos et al [79].

6.1.3 Matthews Jr. four-stage hardware pipeline [105]

In 2008, another US patent published by Matthews Jr. [105] proposed a better hardware architecture for RC4, where a multiple ported memory is used to allow pipelined read and write access to values in memory.

To improve the earlier design [106], this new design applies coherency checking to maintain the consistency of read-after-write and write-after-write operations in RC4. Further, the initialization of the memory is improved in [105] so that it occurs in a single cycle. The architecture of the RC4 hardware in [105] is similar to that in Figure 6.2, but improves the pipeline structure to a four-stage model, as in Table 6.1 (originally [105, Table 1]). Combination of these improvements resulted in a throughput of 1 cycle-per-byte for the cipher.

Table 6.1: Pipeline stages for design proposed by Matthews Jr. [105, Table 1].

$i_1 = i_0 + 1$	$i_2 = i_1 + 1$	$i_3 = i_2 + 1$		
	Read $S[i_1]$	Read $S[i_2]$		
	Store $S[i_1]$ into SI	Store $S[i_2]$ into SI		
	$j_1 = j_0 + S[i_1]$	$j_2 = j_1 + S[i_2]$		
		Read $S[j_1]$	Read $S[j_2]$	
		Store $S[j_1]$ in SJ	Store $S[j_2]$ in SJ	
		$t_1 = SI + S[j_1]$	$t_2 = SI + S[j_2]$	
		Write SI into $S[j_1]$	Write SI into $S[j_2]$	
			Read $S[t_1]$	Read $S[t_2]$
			Store $S[t_1]$ into K	Store $S[t_2]$ into K
			Write SJ into $S[i_1]$	Write SJ into $S[i_2]$

6.2 New implementations of RC4 hardware

The RC4 hardware architecture proposed by Matthews Jr. [105] was the best in terms of throughput, since 2008. Our recent works [129, 133] present new designs for RC4 hardware with an improved throughput.

6.2.1 Sen Gupta et al loop unrolling approach [133]

In 2010, we proposed [133] an RC4 architecture that produces 1 byte per cycle (or 1 cycle-per-byte), that is the same throughput as in the design by

Matthews [105]. However, the model does not use hardware pipeline approach to obtain this result.

Our main contribution in [133] was to take a new look at RC4 hardware design and exploit the idea of *loop unrolling* in this context. We combined consecutive pairs of cycles in a pipelined fashion, and read off the values of one state of the S-box from previous or later rounds of the cipher. To the best of our knowledge, the idea of *loop unrolling* in RC4 has never been exploited in designing an efficient hardware. The comprehensive design strategy and analysis of the design is presented in Chapter 7.

6.2.2 Sen Gupta et al hybrid approach [129]

One may note that the above design of [133], based on loop unrolling, is completely independent of the design idea of hardware pipelining in case of RC4. Recently in 2013, we proposed [129] a completely new design of RC4 hardware using *efficient hardware pipeline* and *loop unrolling* simultaneously. This model provided a throughput of 2 bytes-per-cycle (or $\frac{1}{2}$ cycle-per-byte) in RC4 PRGA, without losing the clock performance compared to [133]. Detailed account of the design strategy and circuit analysis is presented in Chapter 8.

6.2.3 Comparison with earlier designs

The implementation of both the designs of [129, 133] have been done using VHDL description, synthesized with 90 nm and 65 nm technologies using Synopsys Design Compiler in topographical mode. The improved design of [129] has been synthesized with 130 nm technology as well for comparison. With strict clock period constraints, we could device [129] a hybrid model based on our design of simultaneous loop unrolling and hardware pipelining, which offered the best throughput in hardware implementation:

10 Gbps (i.e., 1.25 GBps) on 130 nm technology
21.92 Gbps (i.e., 2.74 GBps) on 90 nm technology
30.72 Gbps (i.e., 3.84 GBps) on 65 nm technology

Section 8.3 presents the final implementation results of both the aforesaid designs [129, 133], including some intermediate design points. The optimization includes experimentations with strict clock period constraints, and some restructuring of the original model. The final architecture offers the best throughput. Section 8.4 discusses the scope for efficiency improvement using further loop unrolling, and illustrates the limitations regarding this approach. Issues with storage access in terms of further hardware pipeline are also discussed in the same part of the thesis.

The throughput, in terms of cycles-per-byte of keystream generation, of the loop-unrolled design proposed by us in [133] is the same as that of the designs proposed by Kitsos, Kostopoulos, Sklavos and Koufopavlou [79] and Matthews Jr. [106]. The throughput of the hybrid design proposed by us in [129] is approximately *six times* that of the designs proposed by Kitsos, Kostopoulos, Sklavos and Koufopavlou [79] and Matthews Jr. [106], and approximately *twice* that of the design proposed by Matthews Jr. [105]. Table 6.2 summarizes the comparative results in this regard.

Table 6.2: Throughput comparison of RC4 hardware implementations.

Year	Technique	Throughput	Reference
Existing hardware implementations			
2003	Custom pipeline	3 cycles-per-byte	Kitsos et al [79]
2003	Multi-port memory	3 cycles-per-byte	Matthews Jr. [106]
2008	Pipelining	1 cycle-per-byte	Matthews Jr. [105]
New implementations of RC4 hardware			
2010	Loop unrolling	1 byte per cycle	Sen Gupta et al [133]
2013	Hybrid model	2 bytes per cycle	Sen Gupta et al [129]

We discuss the new implementations of RC4 hardware in Chapters 7 and 8.

Chapter 7

Design 1 – One Byte per Clock

In retrospect of the existing hardware designs of RC4 by Kitsos et al [79] and Matthews Jr. [105, 106], we consider the following research problem, as discussed earlier in Section 1.4 of Chapter 1.

Problem 2a. Is it possible to provide a simpler alternative to the best existing designs for RC4 hardware that would yield the same throughput?

This problem has been attempted and solved by us [133] in 2010, and the details of the proposed solution is presented in this chapter.

To solve this problem, we exploit the idea of *loop unrolling* for RC4 hardware implementation. We consider the generation of two consecutive values of Z together, for the two consecutive plaintext bytes to be encrypted. To the best of our knowledge, the idea of loop unrolling has never been exploited in the literature to design RC4 hardware for high throughput.

Loop unrolling – the basic idea

Assume that the initial values of the variables i, j and S are i_0, j_0 and S_0, respectively. After the first execution of the PRGA loop, these values will be i_1, j_1 and S_1, respectively and the output byte is Z_1, say. Similarly, after the second execution of the PRGA loop, these will be i_2, j_2, S_2 and Z_2, respectively. Thus, for the first two loops of execution to complete, we have to perform the operations shown in Table 7.1.

Table 7.1: Two consecutive loops of RC4 Stream Generation

First Loop	Second Loop
$i_1 = i_0 + 1$	$i_2 = i_1 + 1 = i_0 + 2$
$j_1 = j_0 + S_0[i_1]$	$j_2 = j_1 + S_1[i_2] = j_0 + S_0[i_1] + S_1[i_2]$
Swap $S_0[i_1] \leftrightarrow S_0[j_1]$	Swap $S_1[i_2] \leftrightarrow S_1[j_2]$
$Z_1 = S_1[S_0[i_1] + S_0[j_1]]$	$Z_2 = S_2[S_1[i_2] + S_1[j_2]]$

7.1 Individual components of Design 1

To store S-array in hardware, we use a bank of 8-bit registers, 256 in total. The output lines of any one of these 256 registers can be accessed through a 256 to 1 Multiplexer (MUX), with its control lines set to the required address i_1, j_1, i_2 or j_2. Thus, we need 4 such 256 to 1 MUX units to simultaneously read $S[i_1]$, $S[i_2]$, $S[j_1]$ and $S[j_2]$. Before that, let us study how to compute the increments of i and j at each level.

7.1.1 Step 1: Calculation of i_1 and i_2

Incrementing i_0 by 1 and 2 can be done by the same clock pulse applied to two synchronous 8-bit counters. The counter for i_1 is initially loaded with 00000001 and the counter for i_2 is loaded with 00000010, the initial states of these two indices. This serves the purpose for the first two rounds of RC4, in both KSA and PRGA.

Thereafter, in every other cycle, the clock pulse is applied to all the flip-flops except the ones at the LSB position for both the counters, as shown in Figure 7.1. This will result in proper increments of i_1 that assumes only the odd values $1, 3, 5, \ldots$, and that of i_2 assuming only the even values $2, 4, 6, \ldots$, as required in RC4. This is assured as the LSB of i_1 will always be 1 and that of i_2 will always be 0, as shown in Figure 7.1.

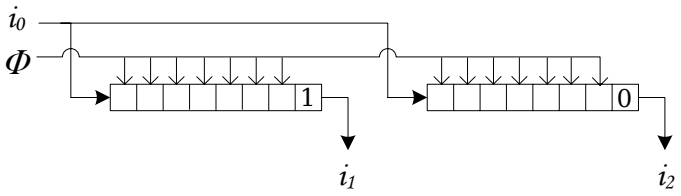

Figure 7.1: [Circuit 1] Circuit to compute i_1 and i_2.

7.1.2 Step 2: Calculation of j_1 and j_2

The values of j_1 and j_2 will be computed and stored in two 8-bit registers. To compute j_1, we need a 2-input parallel adder unit. It may be one using a carry lookahead adder, or one using *scan* operation as proposed by Sinha and Srimani [140], or one using *carry-lookahead-tree* as proposed by Lynch and Swarzlander, Jr. [94]. For computing j_2, there are two special cases:

$$j_2 = j_0 + S_0[i_1] + S_1[i_2] = \begin{cases} j_0 + S_0[i_1] + S_0[i_2] \text{ if } i_2 \neq j_1 \\ j_0 + S_0[i_1] + S_0[i_1] \text{ if } i_2 = j_1 \end{cases}$$

The only change from S_0 to S_1 is the swap $S_0[i_1] \leftrightarrow S_0[j_1]$, and hence we need to check if i_2 is equal to either of i_1 or j_1. Now, i_2 can not be equal to i_1 as they differ only by 1 modulo 256. Therefore, $S_1[i_2] = S_1[j_1] = S_0[i_1]$ if $i_2 = j_1$, and $S_1[i_2] = S_0[i_2]$ otherwise. In both the cases, three binary numbers are to be added.

Let us denote the k^{th} bit of $j_0, S_0[i_1]$ and $S_1[i_2]$ (either $S_0[i_2]$ or $S_0[i_1]$) by a_k, b_k and c_k, respectively, where $0 \leq k \leq 7$. We first construct two 9-bit vectors R and C, where the k^{th} bits ($0 \leq k \leq 8$) of R and C are given by

$$R_k = \text{XOR}(a_k, b_k, c_k) \text{ for } 0 \leq k \leq 7,\ R_8 = 0,\ C_0 = 0,$$
$$C_k = a_{k-1}b_{k-1} + b_{k-1}c_{k-1} + c_{k-1}a_{k-1} \text{ for } 1 \leq k \leq 8.$$

In RC4, all additions are done modulo 256. Hence, we can discard the 9^{th} bit ($k = 8$) of the vectors R, C while adding them together, and carry out normal 8-bit parallel addition considering $0 \leq k \leq 7$. Therefore, one may add R and C by a parallel full adder as used for j_1. The circuit to compute j_1 and j_2 is as shown in Figure 7.2.

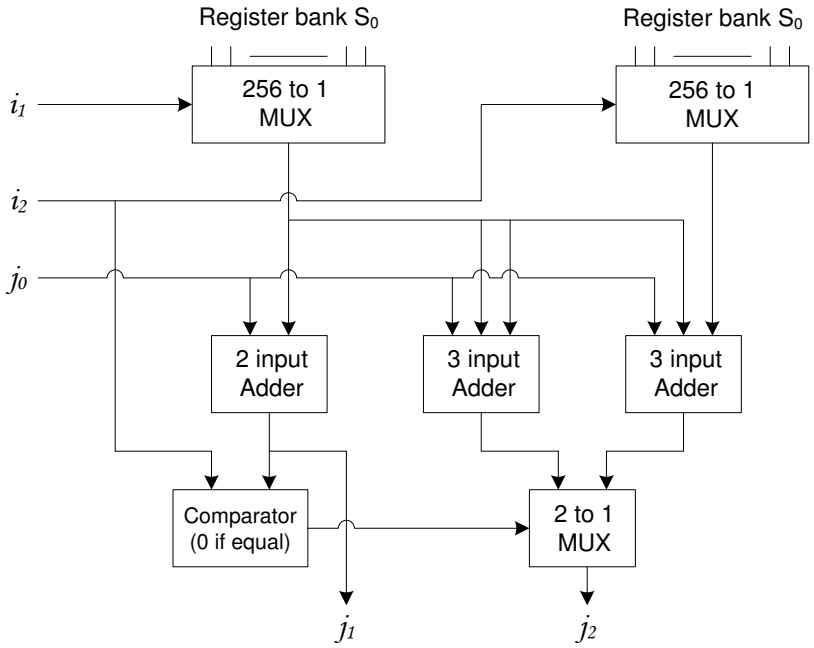

Figure 7.2: [Circuit 2] Circuit to compute j_1 and j_2.

7.1.3 Step 3: Swapping the S values

In Table 7.1, the two swap operations in the third row results in one of the following 8 possible data transfer requirements among the registers of the S-register bank, depending on the different possible values of i_1, j_1, i_2 and j_2. We have to check if i_2 and j_2 can be equal to i_1 or j_1 (we only know that $i_2 \neq i_1$). All the cases in this direction can be listed as in Table 7.2. A more detailed explanation for each case is presented as follows.

Case 1: $i_2 \neq j_1$ and $j_2 \neq i_1$ and $j_2 \neq j_1$
These data transfers are symbolically represented by the following permutation on data in S_0.

$$\begin{pmatrix} i_2 & j_2 \\ j_2 & i_2 \end{pmatrix} \circ \begin{pmatrix} i_1 & j_1 \\ j_1 & i_1 \end{pmatrix}$$

This involves four simultaneous register to register data transfers, as follows: $S_0[i_1] \rightarrow S_0[j_1]$, $S_0[j_1] \rightarrow S_0[i_1]$, $S_0[i_2] \rightarrow S_0[j_2]$ and $S_0[j_2] \rightarrow S_0[i_2]$.

Table 7.2: Cases for the Register-to-Register transfers in the swap operation.

#	Condition	Register-to-Register Transfers
1	$i_2 \neq j_1$ & $j_2 \neq i_1$ & $j_2 \neq j_1$	$S_0[i_1] \to S_0[j_1]$, $S_0[j_1] \to S_0[i_1]$, $S_0[i_2] \to S_0[j_2]$, $S_0[j_2] \to S_0[i_2]$
2	$i_2 \neq j_1$ & $j_2 \neq i_1$ & $j_2 = j_1$	$S_0[i_1] \to S_0[i_2]$, $S_0[i_2] \to S_0[j_1] = S_0[j_2]$, $S_0[j_1] \to S_0[i_1]$
3	$i_2 \neq j_1$ & $j_2 = i_1$ & $j_2 \neq j_1$	$S_0[i_1] \to S_0[j_1]$, $S_0[i_2] \to S_0[i_1] = S_0[j_2]$, $S_0[j_1] \to S_0[i_2]$
4	$i_2 \neq j_1$ & $j_2 = i_1$ & $j_2 = j_1$	$S_0[i_1] \to S_0[i_2]$, $S_0[i_2] \to S_0[i_1] = S_0[j_1] = S_0[j_2]$
5	$i_2 = j_1$ & $j_2 \neq i_1$ & $j_2 \neq j_1$	$S_0[i_1] \to S_0[j_2]$, $S_0[j_2] \to S_0[j_1] = S_0[i_2]$, $S_0[j_1] \to S_0[i_1]$
6	$i_2 = j_1$ & $j_2 \neq i_1$ & $j_2 = j_1$	$S_0[i_1] \to S_0[j_1] = S_0[i_2] = S_0[j_2]$, $S_0[j_1] \to S_0[i_1]$
7	$i_2 = j_1$ & $j_2 = i_1$ & $j_2 \neq j_1$	This indicates an identity permutation, and hence no data transfer occurs.
8	$i_2 = j_1$ & $j_2 = i_1$ & $j_2 = j_1$	This situation is impossible, as this implies $i_1 = i_2 = i_1 + 1$.

Case 2: $i_2 \neq j_1$ and $j_2 \neq i_1$ and $j_2 = j_1$
In this case the data transfers are represented by

$$\begin{pmatrix} i_2 & j_1 \\ j_1 & i_2 \end{pmatrix} \circ \begin{pmatrix} i_1 & j_1 \\ j_1 & i_1 \end{pmatrix}$$

This involves three data transfers, as follows: $S_0[i_1] \to S_0[i_2]$, $S_0[i_2] \to S_0[j_1] = S_0[j_2]$ and $S_0[j_1] \to S_0[i_1]$.

Case 3: $i_2 \neq j_1$ and $j_2 = i_1$ and $j_2 \neq j_1$
In this case the data transfers are represented by

$$\begin{pmatrix} i_2 & i_1 \\ i_1 & i_2 \end{pmatrix} \circ \begin{pmatrix} i_1 & j_1 \\ j_1 & i_1 \end{pmatrix}$$

This involves three data transfers, as follows: $S_0[i_1] \to S_0[j_1]$, $S_0[i_2] \to S_0[i_1] = S_0[j_2]$ and $S_0[j_1] \to S_0[i_2]$.

Case 4: $i_2 \neq j_1$ and $j_2 = i_1$ and $j_2 = j_1$
In this case the data transfers are represented by

$$\begin{pmatrix} i_2 & i_1 \\ i_1 & i_2 \end{pmatrix} \circ \begin{pmatrix} i_1 & i_1 \\ i_1 & i_1 \end{pmatrix}$$

This involves two data transfers, as follows: $S_0[i_1] \to S_0[i_2]$ and $S_0[i_2] \to S_0[i_1] = S_0[j_1] = S_0[j_2]$.

Case 5: $i_2 = j_1$ and $j_2 \neq i_1$ and $j_2 \neq j_1$
In this case the data transfers are represented by

$$\begin{pmatrix} j_1 & j_2 \\ j_2 & j_1 \end{pmatrix} \circ \begin{pmatrix} i_1 & j_1 \\ j_1 & i_1 \end{pmatrix}$$

This involves three data transfers, as follows: $S_0[i_1] \to S_0[j_2]$, $S_0[j_2] \to S_0[j_1] = S_0[i_2]$ and $S_0[j_1] \to S_0[i_1]$.

Case 6: $i_2 = j_1$ and $j_2 \neq i_1$ and $j_2 = j_1$
In this case the data transfers are represented by

$$\begin{pmatrix} j_1 & j_1 \\ j_1 & j_1 \end{pmatrix} \circ \begin{pmatrix} i_1 & j_1 \\ j_1 & i_1 \end{pmatrix}$$

This involves two data transfers, as follows: $S_0[i_1] \to S_0[j_1] = S_0[i_2] = S_0[j_2]$ and $S_0[j_1] \to S_0[i_1]$.

Case 7: $i_2 = j_1$ and $j_2 = i_1$ and $j_2 \neq j_1$
In this case the data transfers are represented by

$$\begin{pmatrix} j_1 & i_1 \\ i_1 & j_1 \end{pmatrix} \circ \begin{pmatrix} i_1 & j_1 \\ j_1 & i_1 \end{pmatrix}$$

This is the identity permutation, and hence does not involve any data transfer.

Case 8: $i_2 = j_1$ and $j_2 = i_1$ and $j_2 = j_1$
This case cannot occur, as it implies $i_1 = i_2$, which is impossible because $i_2 = i_0 + 2 = i_1 + 1$.

After the swap operation is completed successfully, one obtains S_2 from S_0. From the point of view of the receiving registers (in the S-register bank) in case of the above mentioned register-to-register transfers, we can summarize the cases as follows.

$S_2[i_1]$ receives data from $S_0[i_1], S_0[j_1]$ or $S_0[i_2]$
$S_2[j_1]$ receives from $S_0[i_1], S_0[j_1], S_0[i_2]$ or $S_0[j_2]$
$S_2[i_2]$ receives from $S_0[i_1], S_0[j_1], S_0[i_2]$ or $S_0[j_2]$
$S_2[j_2]$ receives from $S_0[i_1], S_0[i_2]$ or $S_0[j_2]$.

In view of the above, the input data (1 byte) for each of the 256 registers in the S-register bank will be taken from the output of an 8 to 1 MUX unit, whose data inputs are taken from $S_0[i_1], S_0[j_1], S_0[i_2], S_0[j_2]$, and the control inputs are taken from the outputs of three comparators comparing (i) i_2 and j_1, (ii) j_2 and i_1, (iii) j_2 and j_1. The circuit in Figure 7.3 realizes the swap.

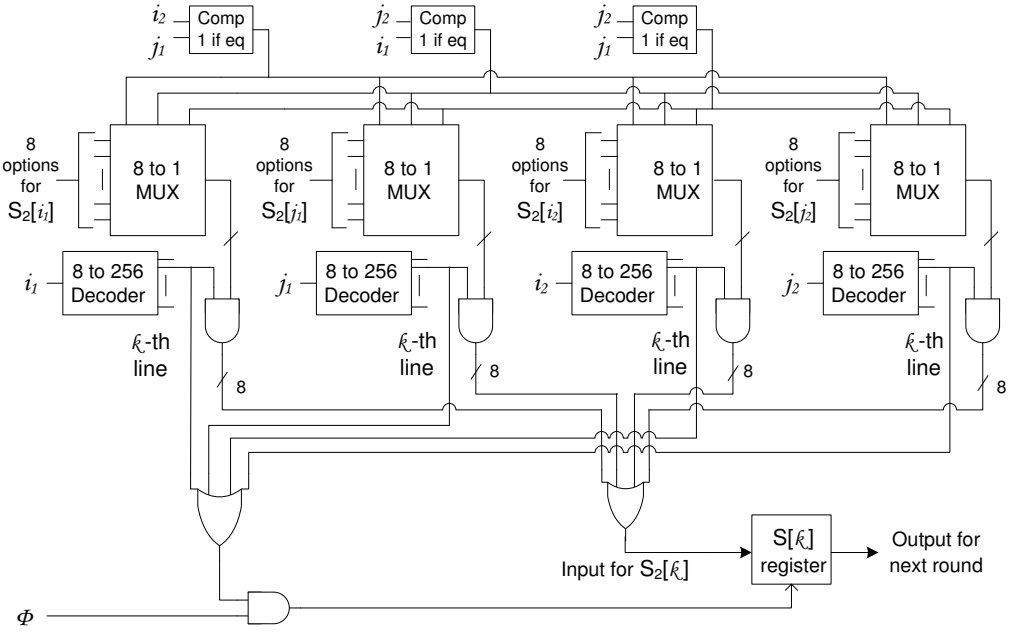

Figure 7.3: [Circuit 3] Circuit to swap S values (data lines for a fixed k).

For the simultaneous register-to-register data transfer during the swap operation, we propose the use of Master-Slave JK flip-flops to construct the registers in the S-register bank. This way, the read and write operations will respect the required order of functioning, and the synchronization can be performed at the end of each clock cycle to update the S-state.

7.1.4 Step 4: Calculation of Z_1 and Z_2

To get the most out of loop unrolling, we need to completely bypass the generation of S_1, and move directly from S_0 to S_2, as discussed right before. However, note that we require the state S_1 for computing the output byte $Z_1 = S_1\left[S_0[j_1] + S_0[i_1]\right]$. We apply the trick of cross-loop look-back to resolve this issue. We can rewrite $Z_1 = S_1\left[S_1[i_1] + S_1[j_1]\right] = S_1\left[S_0[j_1] + S_0[i_1]\right]$ as

$$Z_1 = \begin{cases} S_2[i_2], & \text{if } S_0[j_1] + S_0[i_1] = j_2; \\ S_2[j_2] & \text{if } S_0[j_1] + S_0[i_1] = i_2; \\ S_2[S_0[j_1] + S_0[i_1]] & \text{otherwise.} \end{cases}$$

Computing Z_1 involves adding $S_0[i_1]$ and $S_0[j_1]$ first, which can be done using a 2-input parallel adder. The 256 to 1 MUX, which is used to extract appropriate data from S_2, will be controlled by another 4 to 1 MUX. This 4 to 1 MUX is in turn controlled by the outputs of two comparators comparing (i) $S_0[j_1] + S_0[i_1]$ and i_2, and (ii) $S_0[j_1] + S_0[i_1]$ and j_2, as illustrated in the circuit of Figure 7.4.

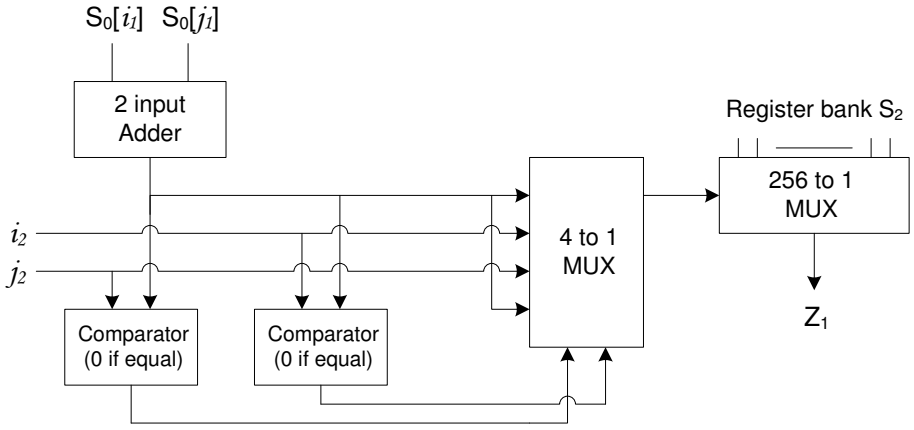

Figure 7.4: [Circuit 4] Circuit to compute Z_1.

Computation of the second keystream output byte Z_2, however, involves adding $S_1[i_2], S_1[j_2]$, as in the following formula:

$$Z_2 = S_2\left[S_2[i_2] + S_2[j_2]\right] = S_2\left[S_1[j_2] + S_1[i_2]\right].$$

In this case, we unwrap one cycle of RC4 and gather the values of $S_1[i_2]$

and $S_1[j_2]$ from the S_0 state. $S_1[i_2]$ and $S_1[j_2]$ receive the values from the appropriate registers of S_0 as given below, depending on the conditions:

$$
\begin{aligned}
&i_2 \neq j_1,\ j_2 \neq i_1,\ j_2 \neq j_1\colon\ S_1[i_2] = S_0[i_2],\ S_1[j_2] = S_0[j_2]\\
&i_2 \neq j_1,\ j_2 \neq i_1,\ j_2 = j_1\colon\ S_1[i_2] = S_0[i_2],\ S_1[j_2] = S_0[i_1]\\
&i_2 \neq j_1,\ j_2 = i_1,\ j_2 \neq j_1\colon\ S_1[i_2] = S_0[i_2],\ S_1[j_2] = S_0[j_1]\\
&i_2 \neq j_1,\ j_2 = i_1,\ j_2 = j_1\colon\ S_1[i_2] = S_0[i_2],\ S_1[j_2] = S_0[j_1]\\
&i_2 = j_1,\ j_2 \neq i_1,\ j_2 \neq j_1\colon\ S_1[i_2] = S_0[i_1],\ S_1[j_2] = S_0[j_2]\\
&i_2 = j_1,\ j_2 \neq i_1,\ j_2 = j_1\colon\ S_1[i_2] = S_0[i_1],\ S_1[j_2] = S_0[i_1]\\
&i_2 = j_1,\ j_2 = i_1,\ j_2 \neq j_1\colon\ S_1[i_2] = S_0[i_1],\ S_1[j_2] = S_0[j_1]
\end{aligned}
$$

These conditions can be realized using an 8 to 1 MUX unit controlled by the outputs of three comparators comparing (i) i_2 and j_1, (ii) j_2 and i_1, (iii) j_2 and j_1. We can use the same control lines as in case of the swapping operation. The circuit is as shown in Figure 7.5.

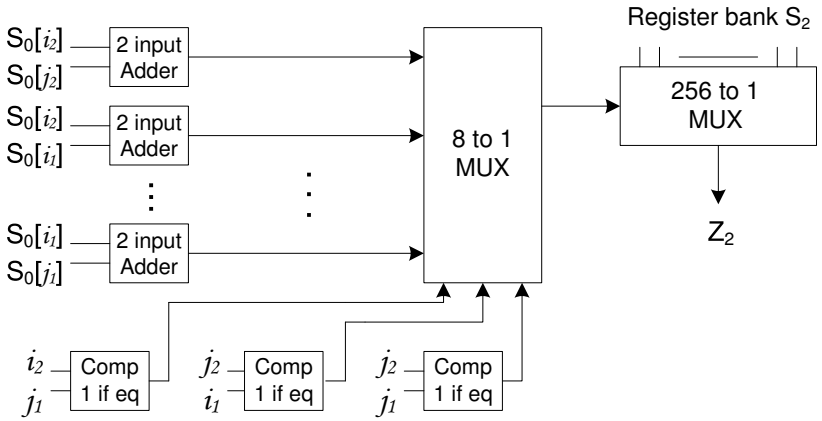

Figure 7.5: [Circuit 5] Circuit to compute Z_2.

7.2 Complete architecture of Design 1

The complete circuit for PRGA algorithm of Design 1 is shown in Figure 7.6.

We shall henceforth denote the clock by ϕ and its cycles numbered as ϕ_1, ϕ_2, etc., where ϕ_0 refers to the clock pulse that initiates PRGA. In Figure 7.6, L_i denote the latches operated by the trailing edge of ϕ_{2n+i}, i.e., the $(2n+i)^{th}$ cycle of the master clock ϕ where $n \geq 0$. For example, the latches labeled L_1

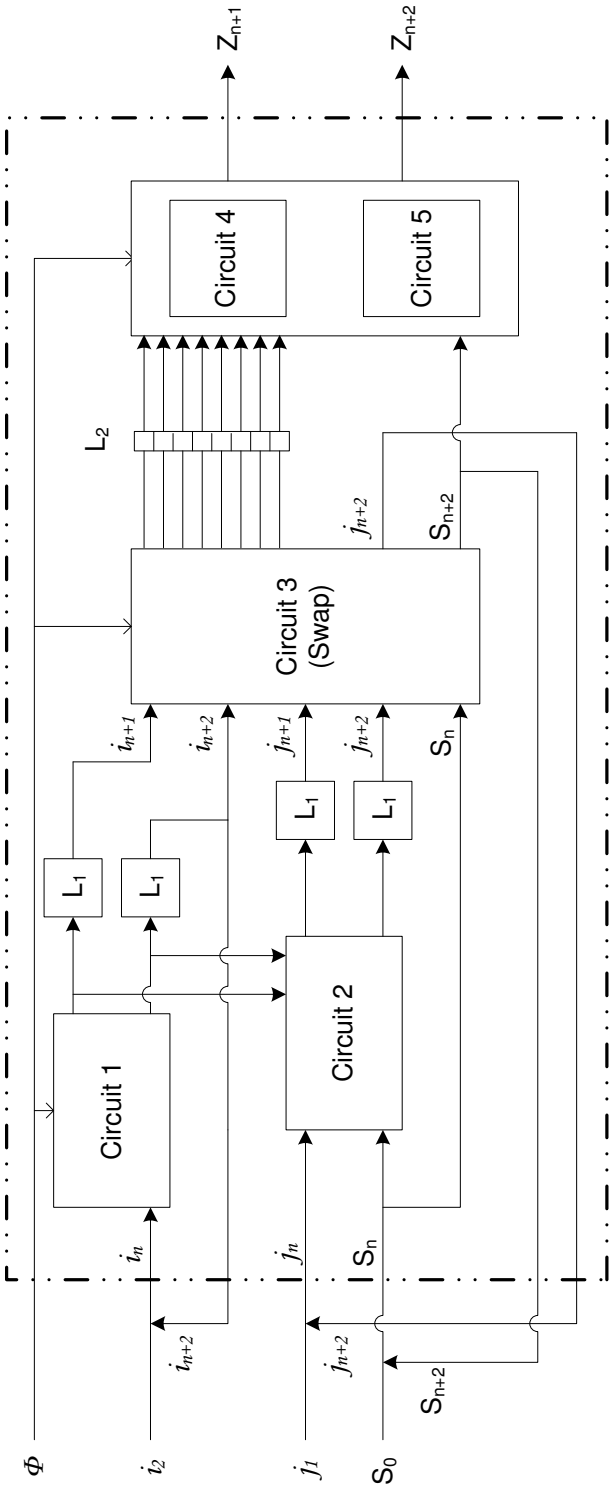

Figure 7.6: Circuit for RC4 PRGA in the proposed architecture (Design 1).

(four of them) are released at the trailing edge of $\phi_1, \phi_3, \phi_5, \ldots$ and the latches labeled L_2 (eight of them) are released at the trailing edge of $\phi_2, \phi_4, \phi_6, \ldots$ etc. In the final implementation, these latches have been replaced by edge-triggered flip-flops which operate at the trailing edge of the clock.

7.3 Timing analysis of Design 1

The timing analysis for the complete PRGA circuit (shown in Figure 7.6) is as shown in the three-stage pipeline diagram of Figure 7.7. We illustrate the first two iterations, and the rest falls along similar lines.

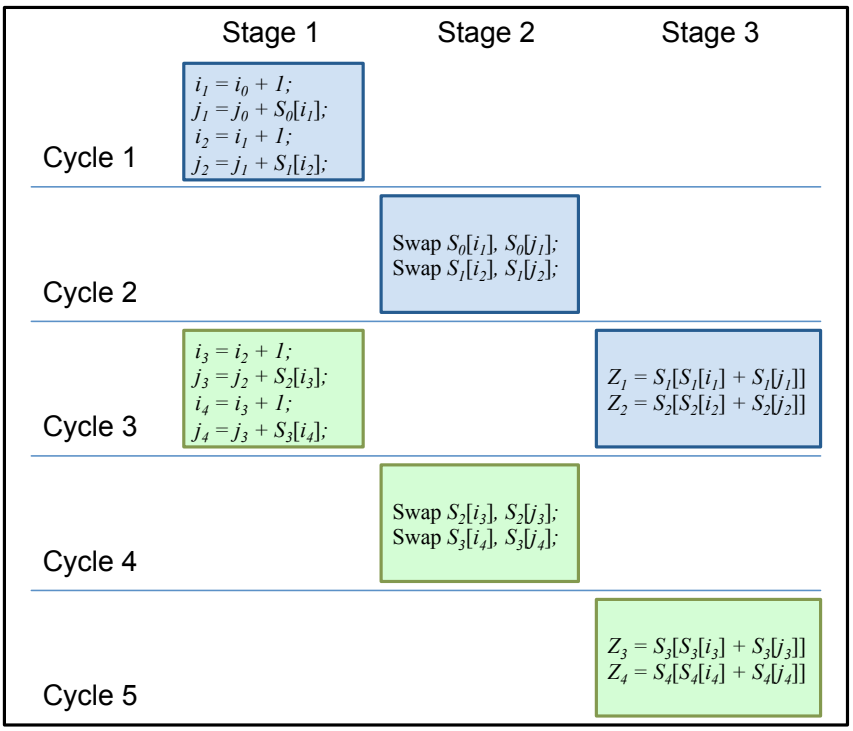

Figure 7.7: Pipeline structure for the proposed Design 1.

The combinational logics operate between the clock pulses and all read, swap and increment operations are done at the trailing edges of the clock pulses. The first two bytes Z_1, Z_2 are obtained at the end of the third clock cycle and the next two bytes Z_3, Z_4 are obtained at the fifth clock cycle. A detailed explanation follows.

7.3.1 Throughput of PRGA in Design 1

We can generalize the previous observations to claim that

> *The hardware proposed for the PRGA stage of RC4 in Design 1, as shown in Figure 7.6, produces "one byte per clock" after an initial delay of two clock cycles.*

Let us call the stage of the PRGA circuit shown in Figure 7.6 the n^{th} stage. This actually denotes the n^{th} iteration of our model, which produces the output bytes Z_{n+1} and Z_{n+2}. The first block (Circuit 1) operates at the trailing edge of ϕ_n, and increments i_n to i_{n+1}, i_{n+2}. During cycle ϕ_{n+1}, the combinational part of Circuit 2 operates to produce j_{n+1}, j_{n+2}. The trailing edge of ϕ_{n+1} releases the latches of type L_1, and activates the swap circuit (Circuit 3).

The combinational logic of the swap circuit functions during cycle ϕ_{n+2} and the actual swap operation takes place at the trailing edge of ϕ_{n+2} to produce S_{n+2} from S_n. Simultaneously, the latch of type L_2 is released to activate the Circuits 4 and 5. The combinational logic of these two circuits operate during ϕ_{n+3}, and we get the outputs Z_{n+1} and Z_{n+2} at the trailing edge of ϕ_{n+3}.

This complete block of architecture performs in a cascaded pipeline fashion, as the indices i_2, j_2 and the state S_{n+2} are fed back into the system at the end of ϕ_{n+2} (actually, i_{n+2} is fed back at the end of ϕ_{n+1} to allow for the increments at the trailing edge of ϕ_{n+2}). The operational gap between two iterations (e.g., n^{th} and $(n+2)^{th}$) of the system is thus two clock cycles (e.g., ϕ_n to ϕ_{n+2}), and we obtain two output bytes per iteration.

Hence, the PRGA architecture of Design 1, as shown in Figure 7.6, produces $2N$ bytes of output stream in N iterations, over $2N$ clock cycles. Note that the initial clock pulse ϕ_0 is an extra one, and the production of the output bytes lag the feedback cycle by one clock pulse in every iteration (e.g., ϕ_{n+3} in case of n^{th} iteration). Therefore, our model practically produces $2N$ output bytes in $2N$ clock cycles, that is "one byte per clock", after an initial lag of two clock cycles.

7.3.2 Throughput of KSA in Design 1

Note that the general KSA routine runs for 256 iterations to produce the initial permutation of the S-box. Moreover, the steps of KSA are quite similar to the steps of PRGA, apart from the following:

- Calculation of j involves key K along with S and i.
- Computing Z_1, Z_2 is neither required nor advised.

We propose the use of our *loop-unrolled* PRGA architecture (Figure 7.6) for the KSA as well, with some minor modifications, as follows:

1. *K-register bank:* Introduce a new register bank for key K. It will contain l number of 8-bit registers, where $8 \leq l \leq 15$ in practice.
2. *K-register MUX:* To read key values $K[i_1 \bmod l]$ and $K[i_2 \bmod l]$ from the K-registers, we introduce two 16 to 1 multiplexer unit. The first l input lines of this MUX will be fed data from registers $K[0]$ to $K[l-1]$, and the rest $(16-l)$ inputs can be left floating (recall that $8 \leq l \leq 15$). The control lines of these MUX units will be $i_1 \bmod l$ and $i_2 \bmod l$ respectively, and hence the floating inputs will never be selected.
3. *Modular Counters:* To obtain modular indices $i_1 \bmod l$ and $i_2 \bmod l$, we incorporate two modular counters (modulo l) for the indices. These are synchronous counters and the one for i_2 will have no clock input for the LSB position, similar to Figure 7.1.
4. *Extra 2-input Parallel Adders:* Two 2-input parallel adders are appended to Figure 7.2 for adding $K[i_1 \bmod l]$ and $K[i_2 \bmod l]$ to j_1 and j_2 respectively.
5. *No Outputs:* Circuits of Figure 7.4 and Figure 7.5 are removed from the overall structure, so that no output byte is generated during KSA. If any such byte is generated, the key K may be compromised.

Using this modified hardware configuration, one can implement two rounds of KSA in 2 clock cycles, that is "one round per clock", after an initial lag of 1 cycle. Total time required for KSA is $256 + 1 = 257$ clock cycles.

7.4 Implementation of Design 1

We have implemented Design 1, the proposed structure for RC4 stream cipher, using synthesizable VHDL description. The S-register box and K-register box are implemented as array of master-slave flip-flops, and are synthesized as standard-cell memory architecture (register-based implementation). The entire VHDL code consists of approximately 1500 lines.

A major area impact of the circuit originates from the large number of accesses to the S-box and the K-box from the KSA and PRGA circuit. Since the PRGA and KSA will not run in parallel, we shared the read and write ports of S-box and K-box between PRGA and KSA. From KSA, 1 read access to K-box, 2 read accesses to S-box and 2 write accesses to S-box are needed. From PRGA, 6 read accesses to S-box and 4 write accesses to S-box are needed. The 2 read accesses correspond to simultaneous generation of two Z values at the last step of PRGA. The 4 read and write accesses correspond to the *double swap* operation. While sharing the mutually exclusive accesses, all the accesses from KSA can be merged amongst the PRGA accesses. Therefore, the total number of read ports to K-box is 1, the total number of read ports to S-box is 6 and the total number of write ports to S-box is 4. This sharing of storage access is as shown in Figure 7.8.

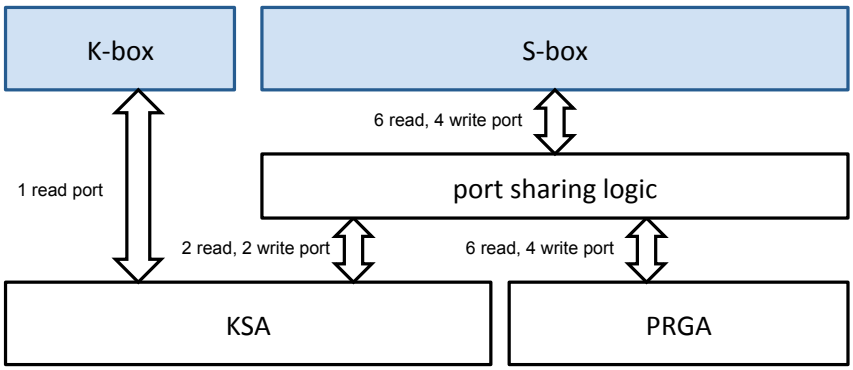

Figure 7.8: Access sharing of KSA and PRGA in Design 1.

The VHDL code is synthesized with 90 nm and 65 nm fabrication technologies using Synopsys Design Compiler in topographical mode. The detailed implementation results are presented in Table 8.1 of Section 8.3, along with proper comparisons with existing designs of RC4 hardware in Table 8.2.

7.5 Comparison with existing designs

Let us compare the proposed design with the ones that existed for RC4 hardware till date. We only consider existing designs that are focused towards improved throughput, and not any other hardware considerations.

7.5.1 Kitsos et al [79] and Matthews Jr. [106]

Combining our KSA and PRGA architectures, we can obtain $2N$ output-stream bytes in $2N + 259$ clock cycles, counting the initial delay of 1 cycle for KSA and 2 cycles for PRGA. The hardware implementation of RC4 described in [79] or [106] provides an output of N bytes in $3N + 768$ clock cycles. A formal comparison of the timings is shown in Table 8.2. One can easily observe that for large N, the throughput of our RC4 architecture is 3 times compared to that of the designs proposed in [79] and [106].

Exact area comparison with [79] is not possible since, we do not have access to the FPGA board for which the area figures of [79] is reported. Considering the design idea, both [79] and [106] modeled their storage using block RAMs. This implementation restricts the number of port accesses per cycle. To overcome that, three 256-byte dual-port RAM blocks are used in [79]. Even then, the design requires 3 cycles to produce 1 byte of data. An improved design is reported in [106] where only two 256-byte dual-port RAM blocks are used. Note that we utilize register-based storage for the S and K arrays instead of RAM, as a RAM based storage would incorporate port-access restrictions and latency issues, resulting in a compromise of throughput. An alternative technique to maintain the high throughput with RAM based implementation may be partitioning the arrays according to the accesses, and optimize accordingly.

7.5.2 Comparison with the design of Matthews Jr. [105]

This design proposes a 1 byte-per-cycle design of RC4 hardware using an approach different from ours, and achieves the claimed throughput by means of hardware pipeline. Instead of two, only one iteration per cycle in PRGA is performed. The pipeline design is shown in Table 7.3 (same as [105, Table 1]).

Table 7.3: Pipeline stages for the design proposed in [105].

1st stage	$i_1 = i_0 + 1$	$i_2 = i_1 + 1$	$i_3 = i_2 + 1$		
2nd stage		Read $S[i_1]$	Read $S[i_2]$	Read $S[i_3]$	
		Store $S[i_1]$ into SI	Store $S[i_2]$ into SI	Store $S[i_3]$ into SI	
		$j_1 = j_0 + S[i_1]$	$j_2 = j_1 + S[i_2]$	$j_3 = j_2 + S[i_3]$	
3rd stage			Read $S[j_1]$	Read $S[j_2]$	Read $S[j_3]$
			Store $S[j_1]$ in SJ	Store $S[j_2]$ in SJ	Store $S[j_3]$ in SJ
			$t_1 = SI + S[j_1]$	$t_2 = SI + S[j_2]$	$t_3 = SI + S[j_3]$
			Write SI into $S[j_1]$	Write SI into $S[j_2]$	Write SI into $S[j_3]$
4th stage				Read $S[t_1]$	Read $S[t_2]$	Read $S[t_3]$
				Store $S[t_1]$ into K	Store $S[t_2]$ into K	Store $S[t_3]$ into K
				Write SJ into $S[i_1]$	Write SJ into $S[i_2]$	Write SJ into $S[i_3]$

In terms of throughput, this design provides 1 cycle-per-byte output in PRGA and completes KSA in 256 cycles, with an initial lag of 3 cycles due to the four-stage pipeline (as in Table 7.3). Thus, N bytes of output is produced in approximately $N+259$ clock cycles, which is comparable to the performance of Design 1. Detailed comparative results are presented in Table 8.2.

7.5.3 Study of the design by Matthews Jr. [105]

Main advantage of the design proposed by Matthews Jr. [105] is compactness. It provides a 1 byte-per-cycle throughput without resorting to loop unrolling. We have studied this design closely and have implemented similar idea on our own to understand the time and area constraints better. This is required since the documentation in [105] does not report area or timing figures corresponding to any technology node. Instead a figurative summary of the logic structure is reported. We implemented a similar design, named Pipelined-A, to closely study the design of Matthews Jr. [105], which helps us further to fuse the ideas of loop unrolling and hardware pipelining to obtain a better design.

Pipelined-A: Design based on hardware pipelining

Our first design motivated by hardware pipelining is an architecture pipelined in two stages, as in Figure 7.9. While [105] proposed a deep pipelining with bypass and data-forwarding, our schemes allow a simpler two-stage pipelining with the same throughput of 1 byte-per-cycle and similar memory port restrictions.

The first pipeline stage is devoted for calculation of j and performing the swap, the second pipeline stage computes the value of Z. To minimize the read and write accesses to S-box, the index to be used for second pipeline is computed at the first stage itself. Note that, the index computation at first stage does not alter the result as $S[i]$ and $S[j]$ are swapped, thus the addition result $S[i] + S[j]$ remains intact.

With the aforementioned structure, the pipeline in PRGA circuit is considerably simplified with respect to Design 1. We further study the circuit in order to improve its area and timing. To that effect, we first re-organized the

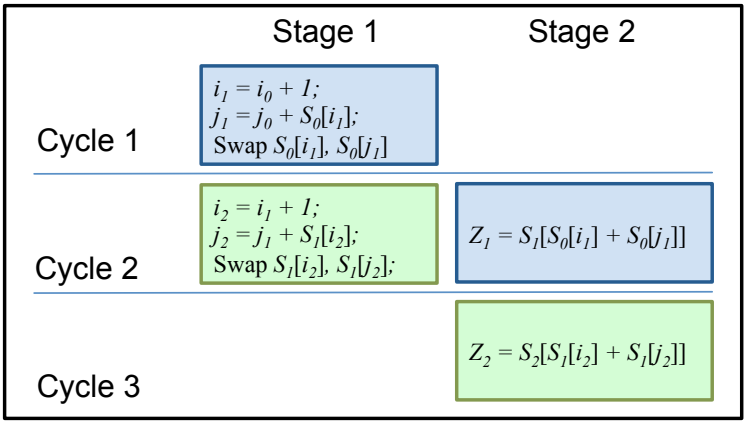

Figure 7.9: 1 byte-per-cycle by Hardware Pipeline (Pipelined-A).

KSA circuit to merge 2 iterations of key generation in 1 cycle. The benefit of this will be to have the S-box created in 128 cycles instead of 256 cycles.

This is done in similar fashion to the unrolling of KSA iterations as per the design discussed earlier. The logic for the two consecutive KSA loops in shown in Table 7.4, and the design idea follows that of Design 1. For this design, the K-box is optimized away as it had constant reset inputs. In the improvements discussed later, the K-box values are controlled from external input. We synthesized the circuit without port-sharing, using 90 nm technology at a strict clock frequency, and the synthesis results are as presented in Section 8.3. The throughput is the same as that of Design 1.

Table 7.4: Two consecutive loops of RC4 Key Scheduling.

First Loop	Second Loop
$i_1 = i_0 + 1$	$i_2 = i_1 + 1 = i_0 + 2$
$j_1 = j_0 + S_0[i_1] + K[i_1]$	$j_2 = j_1 + S_1[i_2] + K[i_2]$
	$= j_0 + S_0[i_1] + S_1[i_2] + K[i_2]$
Swap $S_0[i_1] \leftrightarrow S_0[j_1]$	Swap $S_1[i_2] \leftrightarrow S_1[j_2]$

One may prefer Pipelined-A over Design 1 because of its obvious simplicity. However, in Chapter 8, we look into the possibility of improving the architecture even further by incorporating loop unrolling into the design.

Chapter 8

Design 2 – Two Bytes per Clock

In retrospect of the existing hardware designs of RC4 by Kitsos et al [79] and Matthews Jr. [105, 106], we consider the following research problem, as discussed earlier in Section 1.4 of Chapter 1.

Problem 2b. Is it possible to design a new RC4 hardware that would yield a better throughput compared to the best existing architecture?

This problem has been recently studied and solved by us in [129], and the details of the proposed solution are presented in this chapter.

In this chapter, we present a novel design for RC4 hardware which provides the best throughput till date. We have already proposed a design in Chapter 7 to obtain a throughput of 1 byte-per-cycle. We have also designed and studied a hardware pipeline architecture, namely Pipelined-A (based on [105]), that provides the same. Now we will analyze the two models from a more detailed implementation point of view for potential improvement in the design.

8.1 Optimization of previous designs

Note that in the hardware pipeline based Pipelined-A, as discussed in the Chapter 7 (Figure 7.9), we had fused the idea of loop unrolling to merge two consecutive rounds of KSA. As a result, the number of read accesses to K-box from KSA grew to 2. The number of read and write accesses from KSA to S-

box are both 4 due to the double swap in one cycle. We propose the following optimized designs based on Pipelined-A to probe further into the design for potential improvements.

8.1.1 Pipelined-B: Optimized version of Pipelined-A

We modified Pipelined-A to implement access sharing (read and write) for S-box between KSA and PRGA. In case of Pipelined-A, KSA contains 4 read, 4 write accesses and PRGA contains 3 read and 2 write accesses to the S-box. Naturally, all the accesses from PRGA can be shared with accesses from KSA, resulting in total 4 read and 4 write accesses. The synthesis indicated a compact circuit with the same throughput.

8.1.2 Pipelined-C: Optimized version of Pipelined-A

Another idea, exploited to reduce the circuit size further, is to perform only one iteration of KSA per cycle. In this approach, KSA will require 256 cycles to initialize the S-box. However, the number of both read and write accesses to S-box will become 2 per cycle. By applying access sharing on top of that, total number of read and write accesses to S-box is reduced to 3 and 2 respectively. Furthermore, the number of read accesses to K-box also dropped to 1. The synthesis of this circuit also indicated a more compact design.

We observed a sharp reduction in S-box and KSA areas for Pipelined-C, in comparison with the previous designs. The reduction in the area for S-box is most prominent as Pipelined-C directly reduces the area requirements for the address decoders and multiplexers, due to less number of S-box access ports. However, the reduction in K-box access ports costs us 256 cycles for KSA, instead of 128 as in the previous two designs. Later, we shall present final synthesis results for all the designs to compare the mutual pros and cons.

In the next section, we extend our analysis on the area and timing improvements of hardware pipelining to propose a completely new and considerably improved design for the RC4 implementation.

8.2 Architecture for Design 2

Recall the design based on the hardware pipeline approach (Pipelined-A) as shown in Figure 7.9, and also the main idea of Chapter 7 (Design 1) where a completely new approach to RC4 hardware design gave rise to a one-byte-per-clock architecture based on the technique of loop unrolling. In case of hardware pipeline, we used the idea of pipeline registers to control the read-after-write sequence during S-box swaps, and that resulted in a natural two stage pipeline model for RC4 PRGA. In case of loop unrolling, this idea of pipeline registers was not used at all, but the same throughput was obtained by merging two consecutive rounds of RC4 PRGA. Two obvious questions in this direction are:

1. Can these two techniques be combined?
2. Will that provide any better result at all?

This is the main motivation driving the implementation of the next design, which answers both the aforesaid questions in affirmative.

Fusing two design paradigms

We fused the idea of 2-stage hardware pipeline with that of loop unrolling to generate an RC4 circuit with maximum throughput to date. This is obtained for the case with 2-stage PRGA pipeline and KSA circuit with double iterations per cycle in each case. In this case, 128 cycles for S-box preparation is needed at the KSA stage, and then onwards after a gap of one cycle, 2 bytes per cycle are generated for encryption purposes. This shows significant improvements over the previously published RC4 implementations in literature.

8.2.1 Designing the pipeline structure

For an intuitive pipeline architecture and timing analysis of this new design, one needs to recall the pipeline structures of the individual designs based on loop unrolling and hardware pipelining. Notice that the loop unrolling approach of Design 1 used a 3-stage pipeline, as in Figure 7.7:

1. Increment of indices i and j

2. Swap operation in the S-register
3. Read output byte Z from S-register

Alternatively, the hardware pipeline idea of Pipelined-A, as in Figure 7.9, achieved the same using a 2-stage pipeline:

1. Increment of i, j, and Swap in the S-register
2. Read output byte Z from S-register

In the design for 2 bytes per clock cycle throughput, we propose a fusion of the two ideas, to generate a 2-stage pipeline architecture, as shown in Figure 8.1.

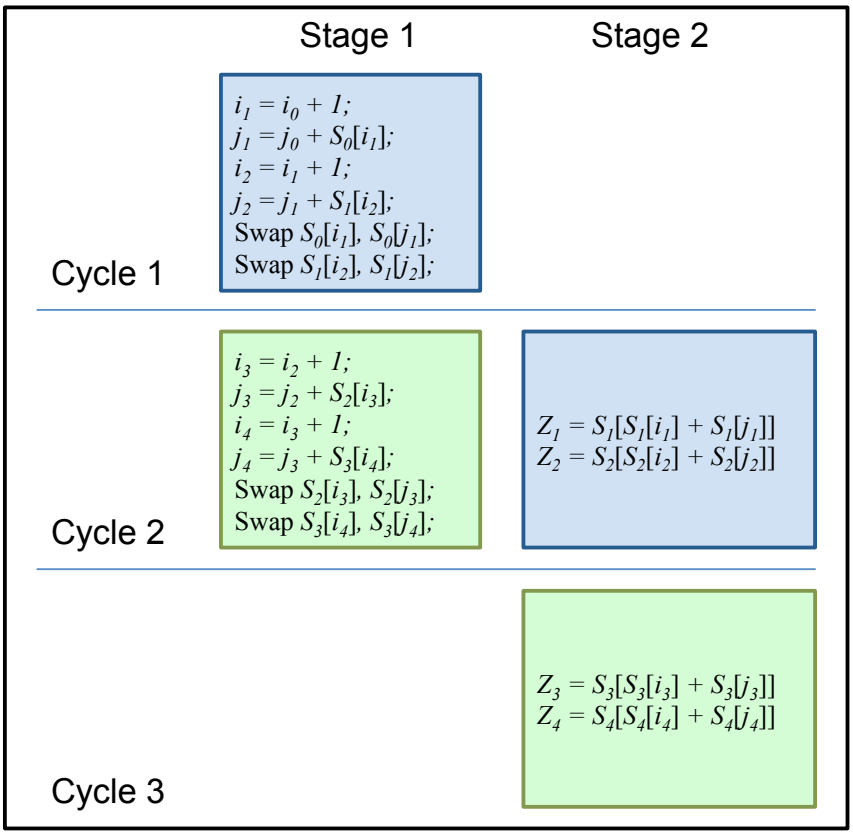

Figure 8.1: Pipeline structure for Design 2.

The double swap operation starts at Stage 1 in this case, and takes the help of pipeline registers to maintain the read-after-write ordering during the swap operations. This part of the operation is same as in the hardware pipelined

approach for one-byte-per-clock design (Pipelined-A/B/C). The Z values are read from the S-registers after the completion of the double-swap, and using the loop unrolling logic from the first one-byte-per-clock design. That is, two consecutive values of the output byte Z are read from the same state S by using some suitable combinational logic. Similarly, the increment of two consecutive i and j values are done simultaneously using the combinational logic of the original one-byte-per-clock design.

This design obviously provides 2 output bytes per clock cycle, after an initial lag of 1 cycle, as is evident from Figure 8.1. Thus for the generation of $2N$ keystream bytes in RC4 PRGA, the circuit has to operate for just $N + 1$ clock cycles, thereby producing an asymptotic throughput of *2 bytes-per-clock*. In KSA, we simply omit Stage 2 of the pipeline structure, and obtain a speed of 2 KSA rounds per clock cycle. Thus, KSA is completed within 128 cycles in this design. In Section 8.4, we will discuss about the issues with further pipelining to obtain better throughput using a similar architecture.

8.2.2 Designing the storage access

Since the PRGA and KSA will not run in parallel, we shared the read and write ports of S-box and K-box between PRGA and KSA. From KSA, 2 read accesses to K-box are required as two loops are merged per cycle. Further, 4 read and 4 write accesses to S-box are needed for the double swap operation. From PRGA, 6 read accesses to S-box and 4 write accesses to S-box are required. The 2 read accesses correspond to simultaneous generation of two Z values at the last stage of PRGA, while the 4 read and 4 write accesses correspond to the double swap operation. While sharing the mutually exclusive accesses, all the accesses from KSA can be merged amongst the PRGA accesses. Therefore, the total number of read ports to K-box is 2, the total number of read ports to S-box is 6 and the total number of write ports to S-box is 4. This port-sharing logic is as in Figure 8.2.

The port sharing logic reduces the multiplexer area significantly. It should be noted that the multiplexer logic to the register banks claims the major share of area. The port sharing logic, as shown in Figure 8.2, reduces a major share of this combinational area in our design. In Figure 8.3, we illustrate the circuit

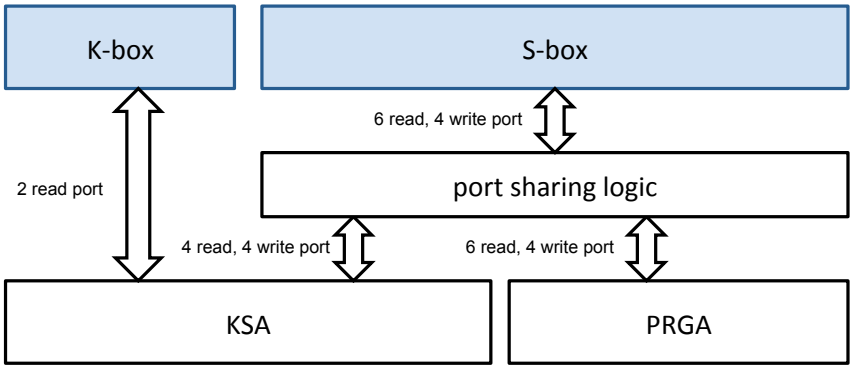

Figure 8.2: Read-write access sharing of KSA and PRGA.

structure for the port sharing logic that operates with the S-box during KSA and PRGA. Note that K-box accesses are only made by KSA, and there is no question of port sharing in that context. In Figure 8.3, we illustrate the port sharing logic, using just 1 read and 1 write port for simplicity.

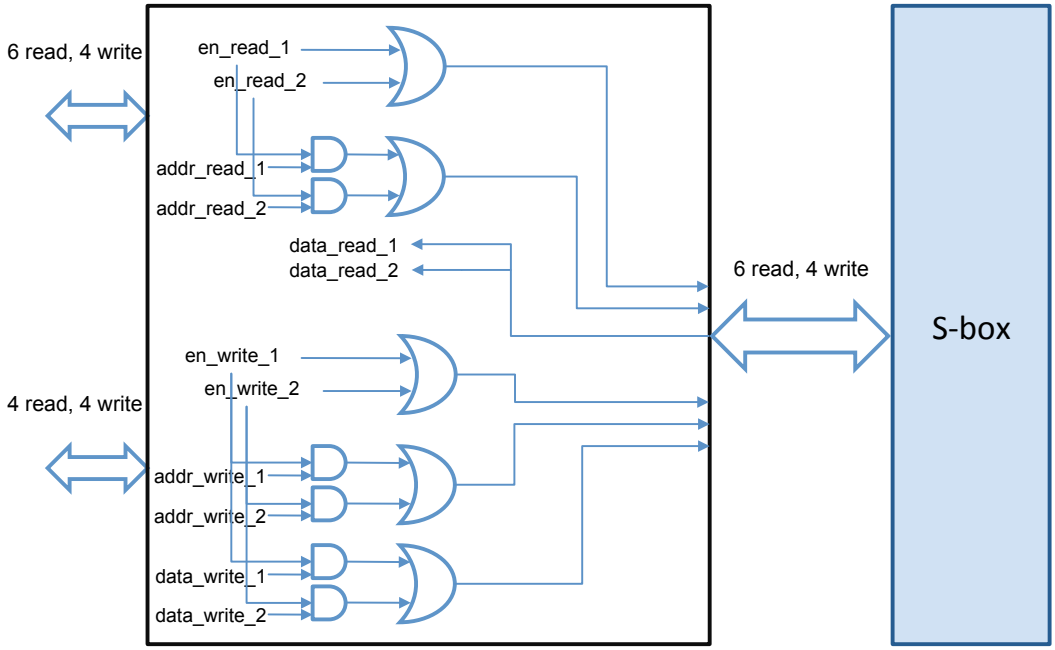

Figure 8.3: Port-sharing of KSA and PRGA for S-box access.

The main storage for the RC4 hardware, as before, is centered around the S-register array and the K-register array. The S-register box comprises of 8 bit registers made of edge-triggered master-slave flip-flops, with a total of 256 such registers to maintain the RC4 states.

To accommodate the read and write accesses to the S-box, we use write-

access decoders and read-access decoders which in turn control 256-to-1 multiplexer units associated to each location of the state array. The K-register box, that holds the RC4 key, is also designed in a similar fashion, but with the exception that no write accesses are required for the K-registers.

8.2.3 Structure of PRGA and KSA circuits

The schematic diagrams for PRGA and KSA circuits in the proposed design are shown in Figure 8.4 and Figure 8.5 respectively.

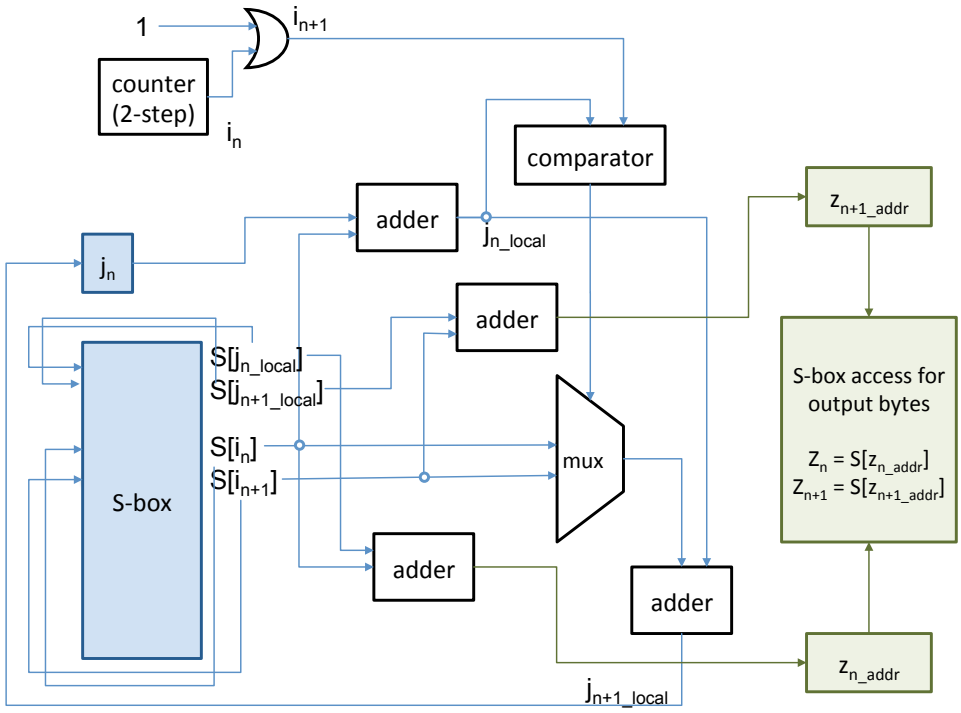

Figure 8.4: PRGA circuit structure for Design 2.

The PRGA circuit operates as per the 2-stage pipeline structure, where the increment of indices take place in the first stage, and so does the double-swap operation for the S-box. In the same stage, the addresses for the two consecutive output bytes Z_n and Z_{n+1} are calculated as the swap does not change the outcomes of the additions $S[i_n] + S[j_n]$ or $S[i_{n+1}] + S[j_{n+1}]$. In the second stage of the pipeline, the output addresses z_{n_addr} and z_{n+1_addr} are used to read the appropriate keystream bytes from the updated S-box.

The circuit for KSA operates similarly, but has no pipeline feature as the

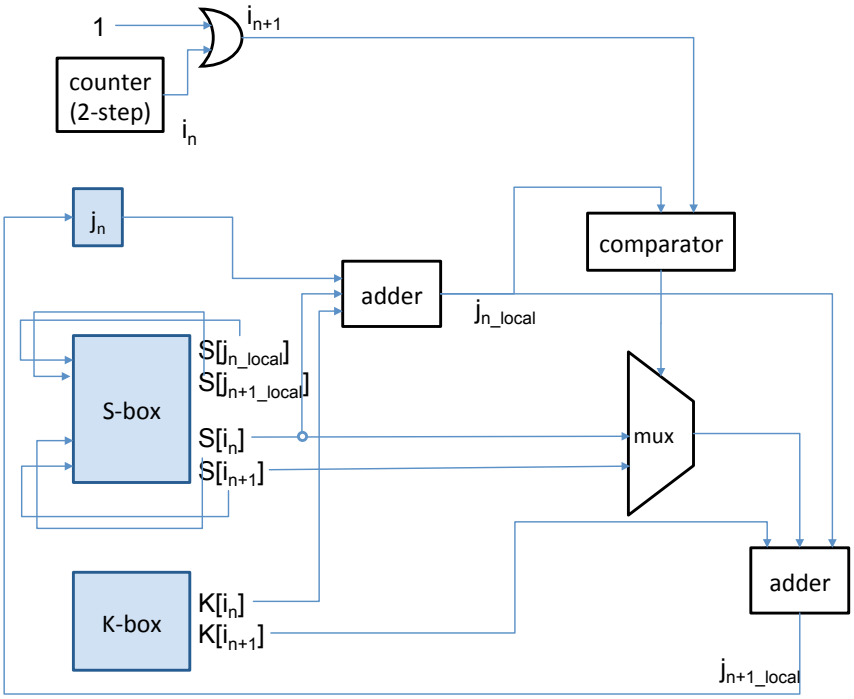

Figure 8.5: KSA circuit structure for Design 2.

operation happens in a single stage. Here, the increment of indices and swap are done for two consecutive rounds of KSA in a single clock cycle, thereby producing a speed of 2 rounds-per-cycle.

Based on this schematic diagram for the PRGA and KSA circuits, and the port sharing logic described earlier, we now attempt the hardware implementation of our new design – Design 2.

8.2.4 Implementation of Design 2

We have implemented the proposed structure for RC4 stream cipher using synthesizable VHDL description. The S-register box and K-register box are implemented as array of master-slave flip-flops, as discussed earlier, and are synthesized as standard-cell memory architecture.

The VHDL code is synthesized with 130 nm, 90 nm and 65 nm fabrication technologies using Synopsys Design Compiler in topographical mode. The synthesis results for Design 2 are presented in Section 8.3.

8.3 Implementation Results

In this section, we describe our attempts to optimize our designs and obtain the best throughput in implementation. The gate-level synthesis was carried out using Synopsys Design Compiler Version D-2010.03-SP4, using topographical mode for 130 nm, 90 nm and 65 nm target technology libraries. The area results are reported using equivalent two-input NAND gates.

8.3.1 Hardware performance of our designs

Implementation in 90 nm technology

We experimented with the synthesis of designs in the following order:

1 byte-per-clock design using loop unrolling (Design 1)
1 byte-per-clock by hardware pipelining (Pipelined-A)
1 byte-per-clock by hardware pipelining (Pipelined-B)
1 byte-per-clock by hardware pipelining (Pipelined-C)
2 bytes-per-clock design combining the two (Design 2)

In order to get the best throughput out of our proposed designs, we performed a few experiments with varying clock speed. This included running of all synthesis at 90 nm with strict clock period constraints until no further improvement was possible. The synthesis results are shown in Table 8.1. The clock period in Pipelined-B is higher than that for Pipelined-A, due to the port sharing logic that we introduced in Pipelined-B.

Optimizing the critical path for Design 2

After the initial implementations of the designs, we found that the critical path for Design 2 is through the KSA read access of the S-array, followed by the additions for updating j values in the first stage of Figure 8.1.

We tried two mechanisms to reduce this critical path. First, we attempted reduced port sharing, as port sharing puts longer delay in the multiplexers. Second, we attempted a modified design with 3 pipeline stages; the first stage

to load the S and K values and put those in pipeline registers, the second stage to perform the additions for j update, and the last one for the swap in KSA. This not only made the design 3-stage pipelined instead of 2-stages, but also required additional bypass logic, which did not help reduce the critical path. So, we finally kept the 2-stage pipeline as in Figure 8.1, and avoided some port sharing along the critical path. This shifted the critical path to the S-box write access from KSA. By removing the port sharing, the clock frequency could be improved even further.

Currently, the critical path is in the S-box read access from PRGA. That could also be improved by removing some port sharing, but only by causing a heavy increase in area. Therefore, we chose to avoid it.

Implementation in 65 nm technology

We used the same designs which yielded best clock frequencies in 90 nm, and mainly focused at the following three designs:

1 byte-per-clock design using loop unrolling (Design 1)
1 byte-per-clock design by hardware pipelining (Pipelined-C)
2 bytes-per-clock design combining the two (Design 2)

The synthesis results provide us the best throughput for these three designs, obtained by using strict clock period constraints during the implementation. We pushed the clock period until no further improvement could be made, that is until the point that it could generate a valid gate-level netlist. The synthesis results are presented in Table 8.1.

Implementation in 130 nm technology

In order to benchmark against comparable technology libraries, we have synthesized using 130 nm fabrication technology since, several stream ciphers from eSTREAM project [40] have reported their performance in 130 nm and 250 nm technologies [59, 61]. We used the same designs which yielded best clock frequencies in 65 nm and 90 nm design, and have just implemented our best proposal – Design 2. The synthesis results are presented in Table 8.1.

Table 8.1: Synthesis results for optimized designs with different target technology libraries.

Tech.	Design	Area (NAND gate equivalent)							Max. Clock	KSA	PRGA	Speed
(nm)		KSA	PRGA	S-box	K-box	Sequential	Combinational	Total	Freq. (GHz)	(cycles)	(bytes/cycle)	(Gbps)
130	Design 2	310	1000	57435	1180	13819	46106	59925	0.625	256	2	10.00
90	Design 1	310	1199	48669	1084	10942	40320	51262	1.22	256	1	9.76
	Pipelined-A	560	428	52133	1084	10644	43561	54205	1.37	128	1	10.96
	Pipelined-B	527	428	43454	1084	10611	34882	45493	1.25	128	1	10.00
	Pipelined-C	484	612	39231	1084	10801	30610	41411	1.49	256	1	11.92
	Design 2	310	985	52557	1084	10955	43981	54936	1.37	256	2	21.92
65	Design 1	310	1240	53638	1180	14109	42259	56368	1.85	256	1	14.80
	Pipelined-C	550	690	48159	1180	13622	36957	50579	2.22	256	1	17.76
	Design 2	310	927	52998	1180	13484	41931	55415	1.92	256	2	30.72

Table 8.2: Timing comparison of Design 1 and Design 2 with existing designs.

Operations	Number of clock cycles required for each operation			
	[79] and [106]	[105]	Design 1	Design 2
Per KSA round	3	1	1	1
Complete KSA	$256 \times 3 = 768$	$256 + 3 = 259$	$256 + 1 = 257$	$256 + 1 = 257$
N bytes of PRGA	$3N$	$N + 3$	$N + 2$	$N/2 + 2$
N bytes of RC4	$3N + 768$	$259 + (N + 3) = N + 262$	$257 + (N + 2) = N + 259$	$257 + (N/2 + 2) = N/2 + 259$
Cycles per byte	$3 + \frac{768}{N}$	$1 + \frac{262}{N}$	$1 + \frac{259}{N}$	$1/2 + \frac{259}{N}$

Best throughput for Design 1 and Design 2

To summarize, the optimized synthesis offers us the best throughput (in gigabits per second) for hardware implementation of RC4 cipher to date.

Design 1 (one-byte-per-clock):

9.76 Gbps in 90 nm technology
14.8 Gbps in 65 nm technology

Design 2 (two-bytes-per-clock):

10 Gbps in 130 nm technology
21.92 Gbps in 90 nm technology
30.72 Gbps in 65 nm technology

8.3.2 Comparison with existing designs

We compare our proposed designs Design 1 (from Chapter 7) and Design 2 (from Chapter 8) with the ones that existed for RC4 hardware to date. We only consider the existing designs [79, 105, 106] that are focused towards improved throughput, and not any other hardware considerations.

Comparison of throughput

In Chapter 7, we have already seen the main RC4 designs proposed in the literature: 3 cycles-per-byte designs of [79] and [106] and 1 cycle-per-byte design of [105]. Chapter 7 presents our 1 byte-per-cycle architecture Design 1, and in Chapter 8 we propose the first 2 bytes-per-cycle RC4 architecture Design 2.

Design 2 requires 257 cycles to complete KSA and generates 2 bytes-per-cycle in PRGA, with an initial lag of 2 cycles. Thus, Design 2 produces N keystream bytes in approximately $257+(N/2+2) = N/2+259$ clock cycles. For large N, this is twice in comparison with Design 1 and the hardware proposed in [105]. Detailed comparison of throughput is presented in Table 8.2.

Comparison of area

As discussed earlier in Chapter 7, a precise comparison of area requirements with [79] could not be made due to mismatch in implementation platforms, and [106] does not specify any area figures at all. However, [106] uses two 256-byte dual-port RAM blocks for the storage, and for the purpose of comparison we synthesized the RAM using 65 nm technology. This resulted in approximately 11.3 KGates of storage area (without considering the associated circuitry), which is comparable to the total sequential area (approximately 13.5-14.0 KGates for Design 1 and Design 2 in 65 nm technology, as shown in Table 8.1) of our designs with register-based storage. Fair comparison with [105] could not be done due to lack of relevant area figures.

8.3.3 Comparison with other stream ciphers

To put our results in perspective, we surveyed the throughputs of a few popular hardware stream ciphers. The current eSTREAM portfolio of hardware stream ciphers contain three ciphers: Grain_v1, MICKEY_v2 and Trivium. According to a hardware performance evaluation of the ciphers (as in [59] and [61]), these ciphers achieve the following throughputs, with maximum possible optimization and parallelization.

- Grain128: 14.48 Gbps (130 nm), 4.475 Gbps (250 nm)
- MICKEY: 0.413 Gbps (130 nm), 0.287 Gbps (250 nm)
- Trivium: 22.3 Gbps (130 nm), 18.568 Gbps (250 nm)

In the data above, the current version Grain128_v1 is tested on 130 nm technology, whereas the result for 250 nm technology is with Grain128_v0, as in [61].

One may observe that in context of the eSTREAM hardware stream ciphers, the optimized implementation of RC4 that we provide fares quite well in terms of throughput (10 Gbps), although RC4 is never claimed to be a hardware cipher. It should also be noted that the area requirements for the proposed RC4 designs (50-60 KGates for Designs 1 and 2) are fairly high compared to those in case of the aforementioned eSTREAM ciphers, as evaluated

in [59] (3.2 KGates for Grain128, 5.0 KGates for MICKEY and 4.9 KGates for Trivium). However, compared to processors or co-processors in embedded systems, this area is quite reasonable, and is small enough to be integrated in modern embedded processors.

The optimization is between the efficiency-requirement and the area-constraint on the user end. If high throughput is required for the time-tested and widely accepted stream cipher RC4, the user may go for a slightly high-area implementation as we have proposed in this chapter, and if the area constraints are stricter, the user may go for the RAM-based implementation proposed in [46], or prefer the new lightweight stream ciphers over RC4.

The techniques of loop unrolling and hardware pipelining discussed in this thesis may be adopted in general for any stream cipher based on the paradigm of extracting pseudorandom words from pseudorandom permutations; namely VMPC [154], RC4A [122], RC4$^+$ [95,116], RC4(n, m) [114], GGHN [58], Py [19] and the HC-family of stream ciphers from the eSTREAM project [40].

8.4 Further improvements in throughput

Based on Design 2, the fastest known hardware implementation of RC4 till date, one may be tempted to push the architecture even further so as to increase its throughput. We tried to venture in this direction as well, and noticed that a better throughput can be obtained via one of the two following avenues:

1. Unroll more loops of the algorithm
2. Increase the pipeline depth

First, we shall take a look at issues with further loop unrolling, starting with the idea of Design 1, as detailed in Chapter 7.

8.4.1 Unrolling three or more loops of RC4

In hardware design, the idea of loop unrolling proves to be most effective when the parameters involved in each of the loops are independent. In case of KSA or

PRGA in RC4 stream cipher, we are not quite so lucky. The interdependencies between consecutive loops of RC4 originate from the following ordering of steps:

$$\text{Update indices} \rightarrow \text{Swap } S\text{-values} \rightarrow \text{Output } Z$$

This order has to be obeyed at all circumstances to maintain the correctness of the PRGA routine in the RC4 stream cipher.

Recall Design 1 (Chapter 7) where we first introduced the concept of loop unrolling in case of RC4. To implement this idea, we had to take into account all dependencies between two consecutive loops. As the Swap and Output stages depend directly upon the indices i and j, we only needed to consider the interplay between the indices of the two rounds, i.e., i_1, i_2 and j_1, j_2. We had a total of $\binom{4}{2} = 6$ pairs to deal with, but the equality or inequality of 3 of these $(i_1, i_2;\ i_1, j_1;\ i_2, j_2)$ did not matter, as they were either impossible to occur, or were anyway expected in the RC4 algorithm. So, we had to worry about the comparison between 3 pairs of indices

$$(i_1, j_2) \quad \text{and} \quad (i_2, j_1) \quad \text{and} \quad (j_1, j_2)$$

in case of S-box swaps as well as the computations for Z_1 and Z_2. These 3 pairs gave rise to $2^3 = 8$ choices in each case, and that in turn contributed to the large area for the combinational logic in our architecture.

Now, if we try to unroll another loop of RC4, i.e., three consecutive rounds of the cipher at a time, we will have to deal with 6 indices i_1, i_2, i_3 and j_1, j_2, j_3. There are $\binom{6}{2} = 15$ pairs, out of which we will have to consider 9 pairs for comparison:

$$(i_1, j_2), (i_1, j_3), (i_2, j_1), (i_2, j_3), (i_3, j_1), (i_3, j_2)$$

$$\text{and} \quad (j_1, j_2), (j_2, j_3), (j_1, j_3)$$

These pairs will give rise to $2^9 = 512$ choices of S-box swaps and output computation, and will require combinational logic to take care of the choices.

In the hardware implementation of Design 1 and Design 2, one may observe that the combinational area already figures quite high. With three loops unrolled, it will be impossible to manage as the logic requirement grows exponentially (8 choices for 2 loops to 512 choices for 3 loops). Hence, the idea of

further loop unrolling in RC4 does not seem feasible in practice.

8.4.2 Increasing depth of the pipeline

In this case, the motivation was to check if deeper hardware pipelining can further improve the throughput for our RC4 architecture. During our experiments with the design, we observed that the critical path in the architecture is due to S-box access, which cannot be improved by whatever deep pipelining one may design. The only way to reduce the critical path in S-box access logic is to explore either of the following choices.

Hand-optimizing the multiplexer logic, or
Partitioned S-box to reduce multiplexing logic

The first option is hard to perform from the RTL and the multiplexer logic is anyway highly optimized by the synthesis tools. One may however investigate the second option further, but most likely such a structure will require predictable access pattern for different partitions of the S-box. This 'predictable access pattern' might lead to undesirable compromise of cryptographic security.

In summary, we have considered all intuitive avenues to further increase the throughput of the RC4 hardware, but have realized that accomplishing the task with loop unrolling and increased pipeline depth may be highly improbable. Hence, we end the quest for solving Problem 2b with our best (2 bytes-per-cycle) architecture Design 2, proposed in [129] and discussed in Chapter 8.

Part III

Conclusion and Bibliography

Chapter 9

Conclusion and Open Problems

In this chapter, we conclude the thesis. We studied the analysis of RC4 stream cipher in Chapters 3, 4 and 5, and implementation of RC4 in Chapters 7 and 8.

Section 9.1 revisits the individual chapters to summarize the thesis. We mention the existing results, specific research problems, and our results presented in each chapter. In Section 9.2, we present a consolidated account of our contributions in this thesis. The conclusion ends in Section 9.3 with an account of the future scope for research and open problems.

9.1 Summary of the thesis

This thesis constitutes of two parts, discussing the analysis and implementation of RC4 stream cipher. Chapter 1 presented the basic structure of stream ciphers, details of our target cipher RC4, and our motivation to consider the research problems for this thesis. Chapter 2 acted as an introduction to Part I of this thesis, and presented a comprehensive overview of RC4 cryptanalysis to date. Chapter 6 played the same role for Part II, and presented the details of current state-of-the-art in hardware implementation of RC4.

The original technical content of this thesis spanned over Chapters 3, 4, 5, 7 and 8, and dealt with ten problems in total: Problems 1a–1h in Part I and Problems 2a–2b in Part II. The individual problems, their respective motivation, and our results on each of them are summarized as follows.

9.1.1 Chapter 3 – RC4 biases based on keylength

This chapter dealt with two problems: Problem 1a and Problem 1b.

Problem 1a – Keylength and extended keylength dependent biases

> *Some of the empirical biases observed by Sepehrdad, Vaudenay and Vuagnoux [136] seem to be related to the length of the secret key used in RC4. If this is the case, is it possible to identify and prove such general relations between the keylength to the keystream biases?*

Our motivation arose from the bias $\Pr(S_{16}[j_{16}] = 0 \mid Z_{16} = -16) \approx 0.038488$ observed by Sepehrdad, Vaudenay and Vuagnoux [136]. A related KSA version of this bias (of the same order) was reported in [134, Section 6.1] for the event $(S_{17}^K[16] = 0 \mid Z_{16} = -16)$, but neither was proved in the literature.

Keylength dependent conditional biases: While exploring these conditional biases in RC4 PRGA, we ran extensive experiments with $N = 256$ and keylength $5 \leq l \leq 32$. We observed that the biases correspond to the keylength l:

$$\begin{aligned} \Pr\left(S_l[j_l] = 0 \mid Z_l = -l\right) &\approx \eta_l^{(1A)}/256, \\ \Pr\left(S_{l+1}^K[l] = 0 \mid Z_l = -l\right) &\approx \eta_l^{(1B)}/256, \end{aligned}$$

where each of $\eta_l^{(1A)}$ and $\eta_l^{(1B)}$ decreases from 12 to 7 (approx.) as l increases from 5 to 32. We proved both the results in Chapter 3. We also observed and proved a family of new conditional biases in this direction:

$$\begin{aligned} \Pr\left(Z_l = -l \mid S_l[j_l] = 0\right) &\approx \eta_l^{(2)}/256, \\ \Pr\left(S_l[l] = -l \mid S_l[j_l] = 0\right) &\approx \eta_l^{(3)}/256, \\ \Pr\left(t_l = -l \mid S_l[j_l] = 0\right) &\approx \eta_l^{(4)}/256, \\ \Pr\left(S_l[j_l] = 0 \mid t_l = -l\right) &\approx \eta_l^{(5)}/256, \end{aligned}$$

where $\eta_l^{(2)}$ decreases from 12 to 7 (approx.), each of $\eta_l^{(3)}$ and $\eta_l^{(4)}$ decreases from 34 to 22 (approx.), and $\eta_l^{(5)}$ decreases from 30 to 20 (approx.) as l increases from 5 to 32. Proofs of all these new biases are also presented in Chapter 3.

Keylength dependent bias in keystream: One of the main results in Chapter 3 was to find a keylength dependent bias in the following event in RC4 keystream:

$$(Z_l = -l) \quad \text{for } 5 \leq l \leq 32.$$

We proved that the estimate of $\Pr(Z_l = -l)$ is always greater than $1/N + 1/N^2 \approx 0.003922$ for $N = 256$ and $5 \leq l \leq 32$. In Figure 9.1 (same as Figure 3.2), we plot the theoretical as well as the experimental values of $\Pr(Z_l = -l)$ against l for $5 \leq l \leq 32$ (experiments with 1 billion trials).

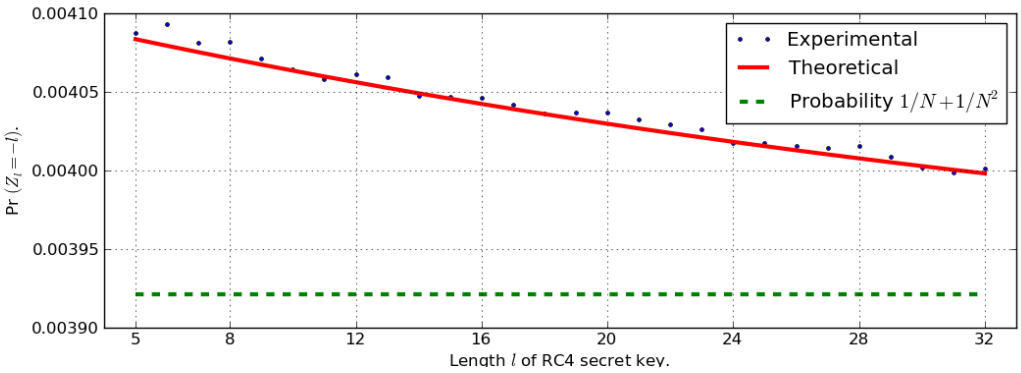

Figure 9.1: Keylength dependent bias in $(Z_l = -l)$ for $5 \leq l \leq 32$.

Extended keylength dependent bias in keystream: Isobe, Ohigashi, Watanabe and Morii [70] observed that similar biases also exist in the class of events $(Z_{xl} = -xl)$ for positive integer $x = 1, \ldots, \lfloor N/l \rfloor$. In an attempt to prove these biases, they explored certain paths, but could not prove all of them. Hence in [70], the authors substituted experimental values to compute what they referred to as semi-theoretical values.

We observed that instead of following the approach of [70], if one follows the approach proposed by us in [132], then the theoretical derivations of the extended keylength dependent biases become much simpler. In Chapter 3, we generalized all the keylength dependent biases of [132] for any keylength $3 \leq l \leq N-1$ and any integer $x = 1, 2, \ldots, \lfloor \frac{N}{l} \rfloor$, and thereby completed the proof of extended keylength biases that was left open in [70]. In fact, the keylength dependent bias of [132] becomes a special case (for $x = 1$) of our general proof of extended keylength dependent biases.

In Figure 9.2 (same as Figure 3.3), we compare the experimental values of $(Z_{xl} = -xl)$, obtained from the data of [5, 14], with our theoretical values derived in Chapter 3, for keylength $l = 16$ and $x = 1, 2, \ldots, 15$. We have obtained similar results for other practical keylengths $(5 \leq l \leq 32)$ as well.

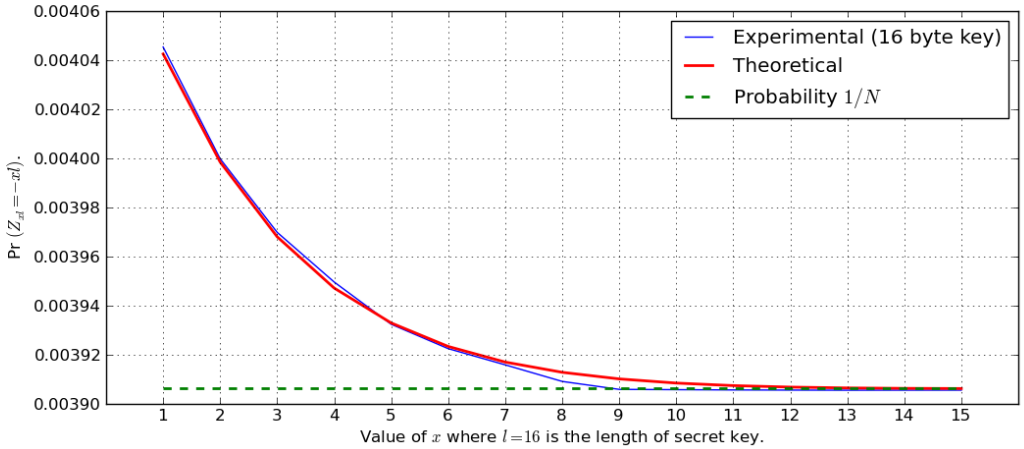

Figure 9.2: Bias in $(Z_{xl} = -xl)$ for $l = 16$ and $x = 1, 2, \ldots, 15$.

The data-point $x = 2$ produced certain mismatch between the theoretical estimates and the experimental observations for $l = 8, 12$, as in Figures 3.4 and 3.5. We leave these as minor open problems for potential future work.

Problem 1b – Keylength dependent bias in Z_1 and anomalies

> *Investigate the negative bias of Z_1 towards 129 ($N = 256$), as observed by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5], and identify its keylength dependence characteristics.*

We attempted to solve the negative bias in the event $(Z_1 = 129)$, which was observed by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5, 14], but neither by Mironov [112] nor in our earlier work [132][1]. Our experiments showed that the negative bias of $(Z_1 = 129)$ is prominent only for keylength l

[1]The graphical representation of the complete probability distribution of Z_1 presented in [132, Fig.9] had a typographic error stating that the experimental values are 'with 16 byte keys', whereas the experimental values were actually recorded for full-length 256 byte secret keys. This is why the curve for Z_1 in [132] missed the prominent, but keylength dependent, bias in $(Z_1 = 129)$. We correct the typographic error in Figure 5.2 of Chapter 5.

equal to non-trivial factors of 256, that is, for $l = 2, 4, 8, 16, 32, 64, 128$. This behavior is depicted in Figure 9.3 (same as Figure 3.10).

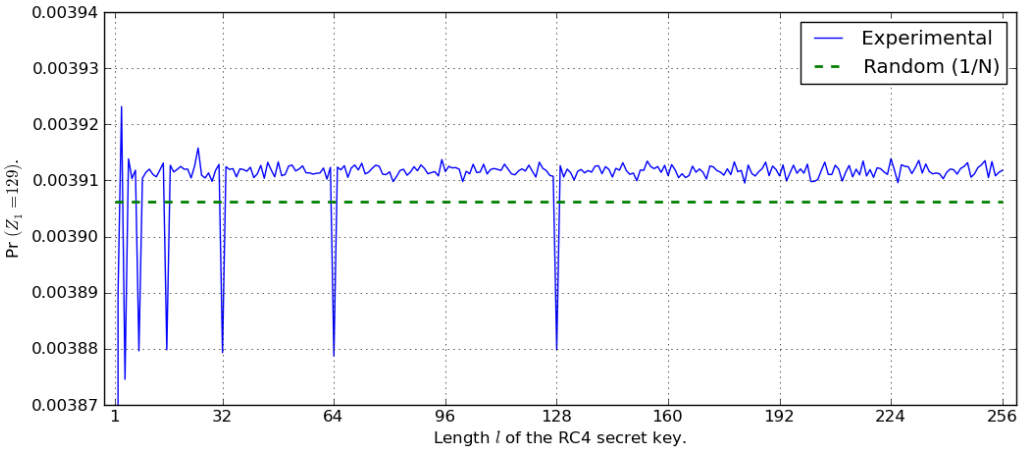

Figure 9.3: Bias in the event $(Z_1 = 129)$ for keylength $1 \leq l \leq 256$.

We observed that the keylength dependence pattern of $(Z_1 = 129)$ is amazingly similar to the keylength dependence of $\Pr(S_0[128] = 127)$, an anomaly observed by Mantin [100] and Paul [116], but not proved in the literature. We tried to relate the bias in $(Z_1 = 129)$ with the long-standing open 'anomaly' in $(S_0[128] = 127)$, and could finally prove the anomaly in Chapter 3. We could not settle the proof of the bias in $(Z_1 = 129)$, and leave it as an open problem.

9.1.2 Chapter 4 – RC4 biases involving state variables

This chapter dealt with two problems: Problem 1c and Problem 1d.

Problem 1c – Identification and proof of significant biases

> *Prove all known significant biases of RC4 involving the state variables, as empirically observed in [136]. In addition, is it possible to identify and prove other interesting biases of similar nature?*

We investigated the significant empirical biases discovered and reported by Sepehrdad, Vaudenay and Vuagnoux [136], and provided theoretical justification for all biases involving the state variables, of the approximate order of $2/N$ or more. The proofs are presented in Chapter 4 under the following categories.

Biases at specific initial rounds: We proved the following results.

$$\Pr(j_1 + S_1[i_1] = 2) = \Pr(S_0[1] = 1) + \sum_{X \neq 1} \Pr(S_0[X] = 2 - X \ \wedge \ S_0[1] = X),$$
$$\Pr(j_2 + S_2[j_2] = 6) \approx \Pr(S_0[1] = 2) + \sum_{X \text{ even}, X \neq 2} (2/N) \cdot \Pr(S_0[1] = X),$$
$$\Pr(j_2 + S_2[j_2] = S_2[i_2]) \approx 2/N - 1/N^2,$$
$$\Pr(j_2 + S_2[j_2] = S_2[i_2] + Z_2) \approx 2/N - 1/N^2.$$

Round-independent biases at initial rounds: We proved the following results for any round $r \geq 1$ of RC4 PRGA.

$$\Pr(j_r + S_r[j_r] = i_r + S_r[i_r]) \approx 2/N,$$
$$\Pr(j_r + S_r[i_r] = i_r + S_r[j_r]) \approx 2/N.$$

Round-dependent biases at all initial rounds: We proved the r-dependent biases in the following events for all initial rounds $3 \leq r \leq N - 1$.

$$\Pr(S_r[j_r] = i_r) \approx \Pr(S_1[r] = r) \left(1 - \frac{1}{N}\right)^{r-2} + \sum_{t=2}^{r-1} \sum_{w=0}^{r-t} \frac{\Pr(S_1[t] = r)}{w! \cdot N} \left(\frac{r-t-1}{N}\right)^w \left(1 - \frac{1}{N}\right)^{r-3-w},$$

$$\Pr(S_r[i_r] = j_r) \approx \Pr(S_r[t_r] = t_r) \approx \frac{r}{N^2} + \sum_{X=r}^{N-1} \frac{1}{N} \left(\Pr(S_1[X] = X) \left(1 - \frac{1}{N}\right)^{r-2} + \sum_{u=2}^{r-1} \sum_{w=0}^{r-u} \frac{\Pr(S_1[u] = r)}{w! \cdot N} \left(\frac{r-u-1}{N}\right)^w \left(1 - \frac{1}{N}\right)^{r-3-w} \right).$$

The above biases were tagged by specific labels in [136], as summarized in Table 9.1 (similar to Table 4.1). In Chapter 4, we proved all these biases.

Problem 1d – Characterization of non-randomness in index j

> *It seems that the index j exhibits certain non-random behavior in the initial rounds of RC4 PRGA. Is it possible to completely characterize the (non-)randomness of index j throughout RC4 PRGA?*

Table 9.1: Significant biases observed in [136] and proved in Chapter 4.

Type of Bias	Label as in [136]	Biases proved in Chapter 4
Specific Initial Rounds	"New_004"	$j_2 + S_2[j_2] = S_2[i_2] + Z_2$
	"New_noz_007"	$j_2 + S_2[j_2] = 6$
	"New_noz_009"	$j_2 + S_2[j_2] = S_2[i_2]$
	"New_noz_014"	$j_1 + S_1[i_1] = 2$
All Rounds (r-independent)	"New_noz_001"	$j_r + S_r[i_r] = i_r + S_r[j_r]$
	"New_noz_002"	$j_r + S_r[j_r] = i_r + S_r[i_r]$
All Initial Rounds (r-dependent)	"New_000"	$S_r[t_r] = t_r$
	"New_noz_004"	$S_r[i_r] = j_r$
	"New_noz_006"	$S_r[j_r] = i_r$

Index j depends on the values of i and $S[i]$ simultaneously, and the pseudorandomness of the permutation S causes the pseudorandomness in j. However, we know that the initial PRGA states S_0 and S_1 of RC4 show certain non-random behavior, as discussed earlier by Mantin [100], as well as by us [132]. This motivated us to analyze the non-randomness of j.

In Chapter 4, we identified and proved the following non-random characteristics of index j, especially prominent in the initial rounds of RC4 PRGA.

$$\Pr(j_1 = v) = \begin{cases} \frac{1}{N}, & \text{if } v = 0; \\ \frac{1}{N}\left(\frac{N-1}{N} + \frac{1}{N}\left(\frac{N-1}{N}\right)^{N-2}\right), & \text{if } v = 1; \\ \frac{1}{N}\left(\left(\frac{N-1}{N}\right)^{N-2} + \left(\frac{N-1}{N}\right)^{v}\right), & \text{if } v > 1. \end{cases}$$

$$\Pr(j_2 = v) = \begin{cases} \Pr(S_0[1] = 2) & \\ + \sum_{w \neq 2} \Pr(S_0[1] = w \ \wedge \ S_0[2] = v - w), & \text{if } v = 4; \\ \sum_{w \neq 2} \Pr(S_0[1] = w \ \wedge \ S_0[2] = v - w), & \text{if } v \neq 4. \end{cases}$$

We did not explicitly analyze or provide closed form expressions of j_r for $r \geq 3$ as our experiments clearly showed that j_r becomes closer to random as r increases beyond 2. This is depicted in Figure 9.4 (same as Figure 4.3).

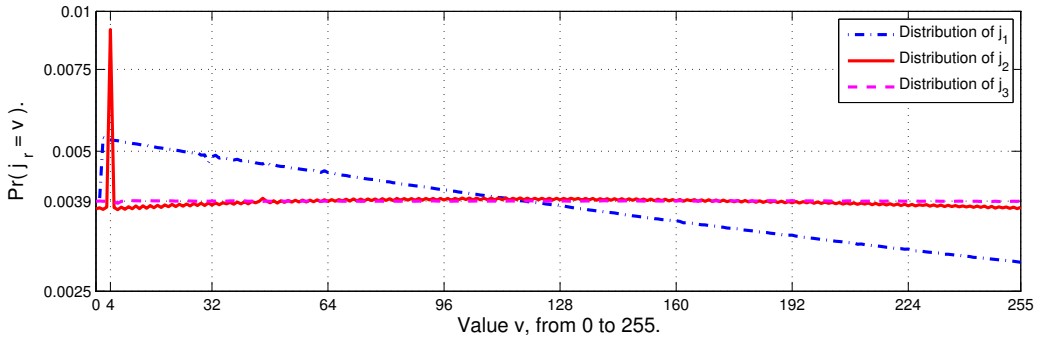

Figure 9.4: Probability distribution of j_r for $1 \leq r \leq 3$.

Glimpse into the state from the keystream

In connection with the state biases and the non-randomness of j proved in Chapter 4, we further investigated for relations between the state variables of RC4 to its keystream bytes. These type of relations are termed as 'glimpse' correlations, as they provide the facility for identifying a state variable using some keystream bytes with probability more than that of random guessing. In Chapter 4, we identified and proves the following glimpse correlations.

Short-term glimpse: We exploited the significant bias in $(j_2 = 4)$ to prove the following correlation between the state byte $S_2[2]$ and the keystream byte Z_2.

$$\Pr\left(S_2[2] = 4 - Z_2\right) \approx \frac{1}{N} + \frac{4/3}{N^2}.$$

Long-term glimpse: In 1996, Jenkins [75] pointed out that the RC4 keystream provides a long-term glimpse of the RC4 state, based on the biased relation $\Pr(S_r[j_r] = i_r - Z_r) = \Pr(S_r[i_r] = j_r - Z_r) \approx 2/N$. While exploring for further long-term glimpse in RC4, we identified the proved the following result.

$$\Pr(S_r[r+1] = N - 1 \mid Z_{r+1} = Z_r \ \wedge \ Z_{r+1} = r + 2) \ \approx \ \frac{3}{N}.$$

Note that no other long-term glimpse bias of magnitude more than $2/N$ has been reported in the literature since Jenkins [75].

9.1.3 Chapter 5 – RC4 biases in keystream bytes

This chapter dealt with four problems: Problems 1e–1h.

Problem 1e – Justification of the sinusoidal distribution of Z_1

> *Theoretically justify the sinusoidal probability distribution of Z_1 and its negative bias towards zero, as observed by Mironov [112].*

In Chapter 5, we proved the complete sinusoidal distribution of the first keystream byte Z_1, observed by Mironov [112]. We got the following result.

Complete distribution of the first keystream byte Z_1:

$$\Pr(Z_1 = v) = Q_v + \sum_{X \in \mathcal{L}_v} \sum_{Y \in \mathcal{T}_{v,X}} \Pr(S_0[1] = X \wedge S_0[X] = Y \wedge S_0[X+Y] = v),$$

$$\text{with} \quad Q_v = \begin{cases} \Pr(S_0[1] = 1 \ \wedge \ S_0[2] = 0), & \text{if } v = 0; \\ \Pr(S_0[1] = 0 \ \wedge \ S_0[0] = 1), & \text{if } v = 1; \\ \Pr(S_0[1] = 1 \ \wedge \ S_0[2] = v) & \\ \quad + \Pr(S_0[1] = v \ \wedge \ S_0[v] = 0) & \\ \quad + \Pr(S_0[1] = 1 - v \ \wedge \ S_0[1 - v] = v), & \text{otherwise,} \end{cases}$$

where $v \in \{0, \ldots, N-1\}$, $\mathcal{L}_v = \{0, 1, \ldots, N-1\} \setminus \{1, v\}$ and $\mathcal{T}_{v,X} = \{0, 1, \ldots, N-1\} \setminus \{0, X, 1-X, v\}$, and the probabilities involving S_0 are calculated based on the results of Mantin [100].

The above result on the complete distribution of Z_1 produced a theoretical curve that closely matched the observed sinusoidal pattern of the first byte for a full-length ($l = 256$) secret key. The theoretical and experimental results (for $l = 256$) are shown in Figure 9.5 (same as Figure 5.2).

Our analysis of the complete distribution of Z_1 also proved its negative bias towards zero, $\Pr(Z_1 = 0) \approx 1/N - 1/N^2$, observed by Mironov [112] in 2002.

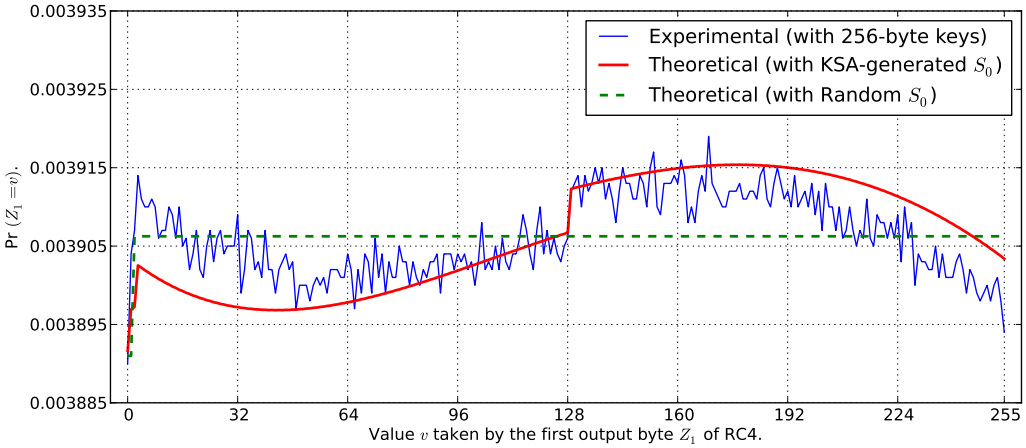

Figure 9.5: The probability distribution of the first output byte Z_1.

Problem 1f – Identification and proof of zero bias in initial bytes

> *Identify all significant biases towards zero in the initial bytes of RC4 keystream (Z_3 to Z_{255}), and prove all subsequent results.*

In 2001, Mantin and Shamir [103] proved the famous $2/N$ bias towards the value 0 for the second byte of RC4 keystream, and also claimed that $\Pr(Z_r = 0) = \frac{1}{N}$ at PRGA rounds $3 \leq r \leq 255$. In Chapter 5, contrary to this claim, we showed (in Theorem 5.5) that $\Pr(Z_r = 0) > \frac{1}{N}$ for all rounds r from 3 to 255. In fact, we explicitly proved that in PRGA rounds $3 \leq r \leq N-1$,

$$\Pr(Z_r = 0) \approx \frac{1}{N} + \frac{c_r}{N^2},$$

$$\text{where} \quad c_r = \begin{cases} \frac{N}{N-1}\left(N \cdot \Pr(S_{r-1}[r] = r) - 1\right) - \frac{N-2}{N-1}, & \text{for } r = 3; \\ \\ \frac{N}{N-1}\left(N \cdot \Pr(S_{r-1}[r] = r) - 1\right), & \text{otherwise.} \end{cases}$$

This result produced a theoretical distribution of $(Z_r = 0)$ as shown in Figure 9.6 (same as Figure 5.4), which closely matched the experimentally observed values of $\Pr(Z_r = 0)$ for $N = 256$ and $3 \leq r \leq 255$.

We further used the biases in bytes Z_3 to Z_{255} to extract information about the state array S_{r-1} using the RC4 keystream byte Z_r. This was done by exploiting the conditional probability $\Pr(S_{r-1}[r] = r \mid Z_r = 0)$ for $3 \leq r \leq 255$.

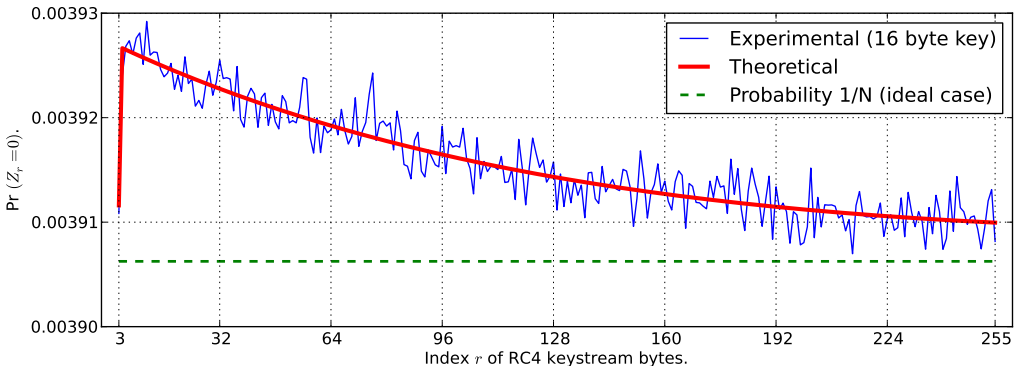

Figure 9.6: $\Pr(Z_r = 0)$ versus r during RC4 PRGA ($3 \leq r \leq 255$).

Problem 1g – Proof of significant empirical short-term biases

Prove all significant biases in initial bytes of RC4 keystream, identified by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5].

Recently there have been two independent attempts at identifying all significant biases in the initial keystream bytes of RC4; one by AlFardan, Bernstein, Paterson, Poettering and Schuldt [5], targeted towards attacking TLS protocol, and the other by Isobe, Ohigashi, Watanabe and Morii [70], targeted towards a generalized broadcast attack on RC4. The major one is [5], which produced an extensive list of around 65536 single-byte biases [14] in the initial keystream of RC4, out of which around 256 were reported in [70], and one in [127].

In Chapter 5, we considered only the significant biases from the list of [5], with a deviation of around $O(1/N^2)$ from the probability $1/N$ of random occurrence. We proved all open (hitherto unproved in literature) isolated short-term biases from this list, as summarized in Table 9.2 (same as Table 5.1).

Problem 1h – Identification and proof of long-term biases

Experimentally discover and prove biases in RC4 keystream which remain effective even after discarding the initial bytes.

It is a common practice to discard a few hundred initial bytes of the keystream to avoid the aforesaid biases; thus motivating the search for long-term biases that are present even after discarding an arbitrary number of initial bytes.

Table 9.2: Proved short-term single-byte keystream biases of RC4.

Biased event	Discovered	Proof (Chapter 5)	Experiment [5,14,70]
$Z_2 = 129$	[5, 127]	$1/N - 2/N^2$	$1/N - 1.82/N^2$
$Z_2 = 172$	[5]	$1/N + 0.28/N^2$	$1/N + 0.2/N^2$
$Z_4 = 2$	[5]	$1/N + 1/N^2$	$1/N + 0.8/N^2$
$Z_{256} = 0$	[5, 70]	$1/N - 0.36/N^2$	$1/N - 0.38/N^2$
$Z_{257} = 0$	[70]	$1/N + 0.36/N^2$	$1/N + 0.35/N^2$

In Chapter 5, we proved that the event $(Z_{wN+2} = 0 \ \wedge \ Z_{wN} = 0)$ is positively biased for all $w \geq 1$. In fact, we proved that for any integer $w \geq 1$, if we assume that the permutation S_{wN} is truly random, then

$$\Pr(Z_{wN+2} = 0 \ \wedge \ Z_{wN} = 0) \approx 1/N^2 + 1/N^3.$$

After the first long-term bias observed by Golic [52] in 1997, this is the only one that involves non-consecutive bytes of RC4 keystream. Golic [52] proved a strong bitwise (only least significant bit) correlation between Z_{wN} and Z_{wN+2}, while we proved a byte-wise correlation.

9.1.4 Chapter 7 – One byte per clock RC4 hardware

This chapter dealt with Problem 2a.

Problem 2a – hardware design matching existing best throughput

> *Is it possible to provide a simpler alternative to the best existing designs for RC4 hardware that would yield the same throughput?*

Throughput of an RC4 circuit can be measured in terms of the average number of cycles required to produce one keystream byte as output. In 2003, a 3 cycles-per-byte implementation of RC4 on a custom pipelined hardware was proposed by Kitsos, Kostopoulos, Sklavos and Koufopavlou [79]. In the same year, a patent by Matthews Jr. [106] was disclosed, which provided a similar 3 cycles-per-byte architecture using multi-port memory units. Another patent

by Matthews Jr. [105] was disclosed in 2008, which proposed a new design for RC4 hardware using pipeline architecture to increase the efficiency of the cipher to obtain a 1 byte-per-cycle throughput. In this direction, we attempted a simpler hardware design for RC4 that would provide the same throughput.

Design 1: In Chapter 7, we proposed a new design principle – loop unrolling – which has never been used in RC4 hardware earlier. We unrolled and fused two rounds of RC4 PRGA into a single step to achieve a throughput of 1 byte-per-cycle in our design, Design 1, matching the throughput of the best existing architecture proposed by Matthews Jr. [105].

9.1.5 Chapter 8 – Two bytes per clock RC4 hardware

This chapter dealt with Problem 2b.

Problem 2b – hardware design improving existing best throughput

> *Is it possible to design a new RC4 hardware that would yield a better throughput compared to the best existing architecture?*

While carefully studying the existing designs based on hardware pipelining, we noticed that the hardware pipelining approach to obtain 1 byte-per-cycle in RC4 hardware, as in Matthews Jr. [105], is entirely independent of our loop unrolling idea. This motivated us to target for an even better architecture.

Design 2: In Chapter 8, we combined the two approaches – loop unrolling and hardware pipelining – into a single design. On top of an optimized 1 byte-per-cycle structure based on hardware pipeline, we implemented our earlier loop unrolling design to get a combined architecture, Design 2, which offered 2 bytes-per-cycle keystream throughput with the same clock frequency.

We implemented our designs on various technology platforms, and the final hybrid design provided the best throughput compared to any RC4 hardware implementation to date: 10 Gbps on 130 nm technology, 21.92 Gbps on 90 nm technology, and 30.72 Gbps on 65 nm technology. Table 9.3 (similar to Table 8.2) summarizes the throughput of both our designs.

Table 9.3: Throughput of Design 1 and Design 2.

Operations	Number of clock cycles required for each operation	
	Design 1	Design 2
Per KSA round	1	1
Complete KSA	$256 + 1 = 257$	$256 + 1 = 257$
N bytes of PRGA	$N + 2$	$N/2 + 2$
N bytes of RC4	$257 + (N + 2) = N + 259$	$257 + (N/2 + 2) = N/2 + 259$
Cycles per byte	$1 + \frac{259}{N}$	$1/2 + \frac{259}{N}$

9.2 Contribution of our work

In this section, we outline the major contributions of our work included in this thesis by placing them on a time-line, in retrospect with the state-of-the-art literature of RC4 analysis and implementation.

9.2.1 Analysis of RC4 stream cipher

Settling long-standing open problems

We settled a few long-standing open questions in RC4 cryptanalysis:

1. *Keylength dependent anomaly:* The mysterious observations regarding 'anomaly pairs' were made by Mantin [100] back in 2002, and again by Paul [116] in 2009. The relation between the anomalies and the length of the secret key of RC4 was only speculated, but not proved in the literature. We settled this long-standing question by proving the keylength dependent bias of the anomaly pair ($S_0[128] = 127$) in our paper [128]. The details are presented in Chapter 3 of this thesis.

2. *Long-term conditional glimpse:* Jenkins [75] was the first to report a glimpse into RC4 state from the keystream with probability $2/N$, and it has since been the best one that persists in the long-term evolution of the PRGA. In this direction, we proved a new long-term conditional glimpse correlation in the event $(S_r[r+1] = N - 1 \mid Z_{r+1} = Z_r \ \wedge \ Z_{r+1} = r + 2)$, which has a magnitude of $3/N$, the best to date. On a similar note, in

one of our works [98], we also characterized the non-random behavior of the index j, identified a hitherto unknown strong bias in $(j_2 = 4)$, and exploited this bias in proving a new glimpse bias depending on the second keystream byte Z_2. The details are presented in Chapter 4.

3. *Distribution of Z_1:* The observation of the sinusoidal distribution of Z_1, and its negative bias towards zero, was made in 2002 by Mironov [112]. We solved it after a decade [132], and the details are in Chapter 5.

4. *Bias of initial keystream bytes towards zero:* In 2001, Mantin and Shamir [103] identified a strong bias of Z_2 towards zero. However, the existence of any similar bias in the other initial bytes was not investigated ever since. About a decade later, we extended the observation of the second-byte bias [103] to all initial bytes Z_3 to Z_{255} in the RC4 keystream [98]. The details are presented in Chapter 5.

5. *Long-term bias in non-consecutive keystream bytes:* After the first long-term bias observed by Golic [52] in 1997, we identified and proved the only long-term bias of RC4 that involves non-consecutive bytes of the keystream. While Golic [52] proved a strong bitwise correlation between the least significant bits of Z_{wN} and Z_{wN+2}, we proved a byte-wise correlation between Z_{wN} and Z_{wN+2} in [132]. The details are in Chapter 5.

Providing theoretical validation for practical attacks

Majority of the recent results in RC4 analysis have targeted experimental identification of biases and their subsequent application towards practical attacks on RC4 based protocols like WEP, WPA and TLS. This has given rise to an array of interesting open problems, some of which were answered by us.

1. *Proving biases involved in recent WEP and WPA attacks:* In 2010, the work of Sepehrdad, Vaudenay and Vuagnoux [136] produced several empirical (observed experimentally, but not proved) biases in linear relations involving RC4 variables within a single round, which were used to mount practical WEP and WPA attacks. We proved all significant biases of this kind in [131, 132], to complement and complete the state-of-the-art picture of practical WEP and WPA attacks. The details are in Chapter 4.

2. *Proving biases involved in recent TLS attacks:* Quite recently in 2013, AlFardan, Bernstein, Paterson, Poettering and Schuldt [5,14] ran extensive experiments to identify about 65536 single-byte biases in the initial RC4 keystream, and have used those to mount an attack on the TLS protocol. In [128] and Chapter 5, we have proved almost all open short-term single-byte biases that have been exploited in this recent TLS attack [5,14], to provide due theoretical justification towards the observations.

Initiating a new direction of RC4 analysis

The most important contribution of our results obtained during the course of this thesis is to initiate a completely new direction of research in RC4 cryptanalysis – keylength dependent biases. The details are presented in Chapter 3.

1. *Discovery of keylength dependent biases:* While proving an empirical bias in the event $(S_{16}[j_{16}] = 0 \mid Z_{16} = -16)$, observed by Sepehrdad, Vaudenay and Vuagnoux [136], we found the bias dependent on the length of the secret key l. We identified the general pattern $(S_l[j_l] = 0 \mid Z_l = -l)$ as well as several other similar conditional biases. These in turn led to the discovery of the first keylength dependent keystream bias of RC4, in the event $(Z_l = -l)$, for every practical keylength $5 \leq l \leq 32$.

2. *Proof of extended keylength dependent biases:* After our discovery of keylength dependent biases, a recent work of Isobe, Ohigashi, Watanabe and Morii [70] identified the extended version of the same in 2013. The extended keylength dependent biases, present in $(Z_{xl} = -xl)$ for $5 \leq l \leq 32$ and $1 \leq x \leq \lfloor N/l \rfloor$, were conclusively proved by us in [128].

3. *Discovery of keylength dependence in Z_1 bias:* In 2013, AlFardan, Bernstein, Paterson, Poettering and Schuldt [5,14] observed a negative bias in $(Z_1 = 129)$, which seemed to be prominent for 16-byte secret keys, but not for full 256-byte keys. We discovered [128] the keylength dependent pattern of this bias, and found that the bias is present only when the keylength l is a non-trivial divisor of $N = 256$, i.e., when $l = 2, 4, \ldots, 128$.

Table 9.4 depicts the impact of our work in retrospect of the current literature. The shaded rows in the table indicate our results discussed in this thesis.

Table 9.4: Contributions of the thesis with respect to the time-line of related results in the literature of RC4 cryptanalysis.

Year	Result in RC4 cryptanalysis	Ref.
Earlier results in RC4 analysis that motivate our work in this thesis.		
1996	Discovery of glimpse correlations	[75]
1997	Long-term bitwise correlation in non-consecutive bytes	[52]
2001	Proof of bias in $(Z_2 = 0)$	[103]
2002	Observation of sinusoidal distribution of Z_1	[112]
2002	Observation of 'anomaly' in initial state S_0	[100]
Contemporary results in RC4 analysis including our work in this thesis.		
2009	Discussion on 'anomaly pairs' in state S_0	[116]
2010	Empirical biases discovered for WEP and WPA attacks	[136]
2011	Proof of biases in $(Z_r = 0)$ for $3 \leq r \leq 255$	[98]
2011	Non-randomness in j and related glimpse	[98]
2011	Practical attacks on WEP and WPA	[137]
2011	Proof of biases related to WEP and WPA attacks	[131]
2011	Discovery and proof of keylength dependent biases	[131]
2012	Revised and improved attacks on WEP and WPA	[134]
2012	Proof of sinusoidal distribution of Z_1	[132]
2012	Long-term correlation in non-consecutive bytes	[132]
2013	Proof of some short-term single-byte biases	[127]
2013	Observation of extended keylength dependent biases	[70]
2013	Further improvements in practical attacks on WEP	[135]
2013	Empirical biases discovered for TLS attack	[5, 14]
2013	Proof of extended keylength dependent biases	[128]
2013	Proof of biases related to TLS attack	[128]
2013	Proof of 'anomaly' in the event $(S_0[128] = 127)$	[128]
2013	Discovery of keylength dependence in $(Z_1 = 129)$	[128]
2013	Discovery and proof of new long-term glimpse	[99]

Implications towards practical attacks

The results that we have discovered and/or proved in this thesis are based on the basic model of RC4 stream cipher. This takes as input a key (typically 16 bytes long), and outputs a keystream of arbitrary length, as per the generic model of a pseudo-random generator (PRG). However, the three celebrated protocols using RC4 for encryption, namely WEP, WPA and TLS, use RC4 in a way that had not been prescribed by the RC4 author.

In all these protocols, RC4 takes as input an IV along with a random key. In case of WEP, the protocol uses RC4 with a pre-shared key appended to the IV for self-synchronization. Using the technique of related key attacks on RC4, this scheme has been broken through passive full-key recovery attacks, and thus WEP is considered insecure in practice. The goal of WPA was to resolve all security threats of WEP, by introducing a key fixing function to feed the RC4 core with different unrelated keys for each packet. In addition to this, WPA incorporated a packet integrity protection scheme to prevent replay and alteration of the initialization vector, which is a main tool in active attacks. TLS however, hashes the IV and the secret key together to produce a completely random key for RC4 to use. This is the closest model to the original RC4, and one may practically consider this as RC4 with the probabilistic model of uniform random keys.

As most of our results on statistical weaknesses of RC4 are based on the above probabilistic model of uniform random keys, we may consider these biases to hold in TLS. In fact, the most prominent recent attack [5] on the protocol exploits quite a few biases proved in this thesis. In case IV is used in any RC4 protocol, and the base model differs from the uniform random key setup, some of these results get surprisingly stronger. Recent attacks on WPA [115, 130] corroborate this fact by identifying stronger statistical weaknesses in RC4 keystream resulting due to weak construction of the IV through the WPA/TKIP key schedule.

In summary, the results on RC4 analysis presented in this thesis provide significant insight towards systematic investigation for statistical weaknesses in practical protocols based on RC4.

9.2.2 Implementation of RC4 stream cipher

Exploiting loop unrolling idea in RC4 hardware

In the literature of RC4 hardware implementation, there are a few high-throughput designs [79,105,106], all of which rely on efficient hardware pipelining to increase the performance. For the first time in this literature, we exploited the concept of 'loop unrolling', or combining two rounds of RC4 into a single-stage operation. The design based on this idea, first proposed in our paper [133], provides the same throughput as that of the best existing design [105]. The details are presented in Chapter 7 of this thesis.

Designing high-throughput hybrid hardware for RC4

Through an extensive survey of the RC4 architectures designed using efficient hardware pipelining, we identified the independence of this approach with respect to our loop unrolling idea. Thus in our paper [129], we proposed a fusion of the two methods into a single platform. This produced the best throughput (2 bytes-per-cycle) in the literature of RC4 implementation to date. The details are presented in Chapter 8 of this thesis.

The time-line in Table 9.5 depicts the major contributions and impact of our work in retrospect of the earlier literature in RC4 implementation. The shaded rows in the table indicate our results discussed in this thesis.

Table 9.5: Contributions of the thesis with respect to the time-line of related results in the literature of RC4 implementation.

Year	Result in RC4 implementation	Ref.
2003	3 cycles-per-byte design based on custom pipeline	[79]
2003	3 cycles-per-byte design based on multi-port memory	[106]
2008	1 cycle-per-byte design based on hardware pipelining	[105]
2010	1 byte-per-cycle design based on loop unrolling	[133]
2013	2 bytes-per-cycle design based on hardware pipelining combined with loop unrolling in a hybrid model	[129]

9.3 Open problems in RC4

RC4 stream cipher has always been a delight for analysts and practitioners due to its surprising robustness packed within the intriguingly simple design. Even after 25 years of analysis and implementation, the cipher continues to offer research problems of interest to both amateurs and seasoned researchers. The following is a list of few interesting open problems in the RC4 literature.

Key collisions in RC4 keystream

The strongest practical results on full-key collisions were published in 2011, by Chen and Miyaji [31]. This work did not device theoretical constructions of related keys, but experimentally found a 22-byte colliding key pair, the shortest to date. However, two questions in this direction remain open:

1. Give a theoretical construction of short key pairs, of size 22 bytes or less.
2. Find a collision with 16-byte key pairs, the practical keylength of RC4.

Key recovery attacks on WPA

WEP attacks have been discussed quite seriously in the literature, but there are not so many attacks on the revised version of the protocol – WPA. During 2011-12, Sepehrdad, Vaudenay and Vuagnoux [134, 137] presented the first practical key recovery attack on WPA. The authors presented a new attack to recover the full 128-bit temporary key of WPA by using only 2^{38} packets. However, the time complexity of this key-recovery attack was theoretically estimated as 2^{96} in [134, 137]. Hence there remains an open question:

3. Is it possible to improve the theoretical complexity of key recovery attack on WPA to bring it closer to the complexity of the practical attack?

Anomaly pairs and keylength

The anomaly pairs had been first identified by Mantin [100], and later discussed by Paul [116]. We proved the anomaly in $(S_0[128] = 127)$ and conclusively

showed a connection with the keylength of RC4. We also attempted to connect this anomaly to the keylength dependent negative bias of ($Z_1 = 129$), which seemed surprisingly similar in pattern. However, we did not settle the issue completely, and it is interesting to pose the following problems:

4. Completely characterize the keylength dependent anomalies in RC4.
5. Identify and prove all keystream biases resulting from the anomalies.

State recovery attacks on RC4

State recovery attacks on RC4 seem to be most illusive. After a series of attempts [54, 81, 113, 139, 148], the best attack was published by Maximov and Khovratovich [107]. However, a contemporary result by Golic and Morgari [57] made some critical remarks on [107], and claimed to improve the attack of [107] even further. However, both the attacks claimed a theoretical complexity of 2^{200} or more for $N = 256$, and hence no practical verification of the attacks on full-sized RC4 was possible ever since. It will be nice to have the following.

6. Comprehensive analysis of both the works [107] and [57] to obtain exact complexity bounds for $N = 256$, and practical attacks for smaller N.

Short cycles in RC4 keystream

RC4 state evolution may be considered as a finite state machine, with the state consisting of a 256-byte permutation array and two indices i, j of size 1 byte each. It is natural to investigate for cycles within this state evolution pattern of RC4, as short cycles in the state would result in short cycles in the keystream. However, apart from an 'impossible' cycle of RC4 discovered by Finney [42], later discussed in [75, 113], there exists no other cycle in the cipher. Thus it is interesting to ask the following questions:

7. Is there a lower bound on the length of 'possible' cycles in RC4?
8. Is it possible to explicitly find a short cycle in the cipher evolution?

Search for keystream biases

Throughout the thesis, we have observed the existence of numerous keystream biases in RC4. In recent times, there have been a few attempts at experimentally identifying short-term biases in the keystream [5, 70, 136]. However, even for the first $N = 256$ bytes of the keystream, the search space for identifying all possible biases grows exponentially. Thus it is not possible to enumerate all significant biases through experimentation. One may however, try the following problem through extensive experimentation.

9. Enumerate all significant biases of the form $(Z_r \star Z_{r+x} = v)$ in the initial keystream bytes, where $x = 1, 2, \ldots, N - r$ and $v = 0, \ldots, N - 1$, and the operation '$\star$' may denote either byte-wise addition or multiplication modulo N, or bitwise logical operations like AND, OR, XOR etc.

Area optimization for high-throughput designs

The best-throughput RC4 hardware to date is Design 2, as we proposed in [129] and Chapter 8. But there may still be some room for improvement, especially in terms of its hardware footprint and the frequency of operation. These could be further improved if we use macro memory units instead of the register file currently used for the S-box. However, macro memories have port restrictions, typically 2 read-write ports, which makes it impossible to accommodate our loop-unrolled design that requires 6 read and 4 write ports for the S-box. One may refer to Figure 8.3 for the port-sharing model of Design 2.

One solution would be to divide the S-box storage into smaller memory banks and check the probability with which each memory bank is accessed within a clock-cycle. The probability of accessing a single memory bank may be computed by considering the access due to index j to be random, while the one due to index i is deterministic. When the accesses come to different memory banks, we obtain the same throughput as proposed for Design 2. But when two accesses collide, we will have to stall one of them, at least for one pipeline stage. This will give us less than ideal performance in terms of the throughput, but would reduce the hardware area. It will be interesting to attempt the following problem in this direction.

10. Find the hardware footprint and expected throughput of Design 2, proposed in Chapter 8, if the storage for the S-box is distributed over a certain number of memory banks consisting of macro memory units.

This idea for splitting up the storage unit into disjoint memory banks may also be used for similar hardware designs [30] of the HC-family of stream ciphers.

Analysis and implementation of RC4-like ciphers

Note that analysis and implementation of major RC4-like ciphers, namely VMPC [154], RC4A [122], RC4$^+$ [95,116], RC4(n, m) (also called NGG) [114], GGHN [58], Py (pronounced Roo) [19], also generate equally interesting research problems. However, we do not cover the these RC4-like ciphers and related problems within the scope of this thesis.

In a nutshell

In this thesis, we presented our results from the last three years (2010–2013) on analysis and implementation of RC4 stream cipher. We identified long-standing open problems as well as contemporary results from the literature, and provided answers to ten research problems in this thesis. During the course of this thesis, we have also studied several related problems in RC4, and have presented ten open problems towards potential future directions in RC4 research. We believe that the plethora of problems arising from the surprisingly simple structure of the RC4 stream cipher will continue to amaze and motivate the community for years to come.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] 3rd Generation Partnership Project. Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3. ETSI/SAGE Specification – Document 2: ZUC Specification, v1.6, June 28, 2011.

[2] 3rd Generation Partnership Project. Specification of the 3GPP confidentiality and integrity algorithms UEA2 & UIA2. ETSI/SAGE Specification – Document 2: SNOW 3G Specification, v1.1, September 6, 2006.

[3] 3rd Generation Partnership Project. Long term evaluation release 10 and beyond (LTE-Advanced). Proposed to ITU at 3GPP TSG RAN Meeting, Spain, 2009.

[4] Mete Akgün, Pinar Kavak, and Hüseyin Demirci. New results on the key scheduling algorithm of RC4. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 40–52. Springer, 2008.

[5] Nadhem AlFardan, Dan Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C.N. Schuldt. On the security of RC4 in TLS. In *USENIX Security Symposium*, 2013. Presented at FSE 2013 as an invited talk [14] by Dan Bernstein. Full version of the research paper and relevant results are available online at `http://www.isg.rhul.ac.uk/tls/`.

[6] Frederik Armknecht and Matthias Krause. Algebraic attacks on combiners with memory. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 2003.

[7] Subhadeep Banik and Subhamoy Maitra. A differential fault attack on MICKEY 2.0. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES*, volume 8086 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2013.

[8] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A differential fault attack on the Grain family of stream ciphers. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 122–139. Springer, 2012.

[9] Elad Barkan and Eli Biham. Conditional estimators: An effective attack on A5/1. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2005.

[10] Elad Barkan, Eli Biham, and Nathan Keller. Instant ciphertext-only cryptanalysis of GSM encrypted communication. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 600–616. Springer, 2003.

[11] Elad Barkan, Eli Biham, and Nathan Keller. Instant ciphertext-only cryptanalysis of GSM encrypted communication. *J. Cryptology*, 21(3):392–429, 2008.

[12] Riddhipratim Basu, Shirshendu Ganguly, Subhamoy Maitra, and Goutam Paul. A complete characterization of the evolution of RC4 pseudo random generation algorithm. *J. Mathematical Cryptology*, 2(3):257–289, 2008.

[13] Riddhipratim Basu, Subhamoy Maitra, Goutam Paul, and Tanmoy Talukdar. On some sequences of the secret pseudo-random index j in RC4 key scheduling. In Maria Bras-Amorós and Tom Høholdt, editors, *AAECC*, volume 5527 of *Lecture Notes in Computer Science*, pages 137–148. Springer, 2009.

[14] Daniel Bernstein. Failures of secret-key cryptography. Invited talk at *FSE 2013*. Session chaired by Bart Preneel, 2013.

[15] Eli Biham and Yaniv Carmeli. Efficient reconstruction of RC4 keys from internal states. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 270–288. Springer, 2008.

[16] Eli Biham and Orr Dunkelman. Cryptanalysis of the A5/1 GSM stream cipher. In Bimal K. Roy and Eiji Okamoto, editors, *INDOCRYPT*, volume 1977 of *Lecture Notes in Computer Science*, pages 43–51. Springer, 2000.

[17] Eli Biham and Orr Dunkelman. Differential cryptanalysis in stream ciphers. *IACR Cryptology ePrint Archive*, 2007:218, 2007.

[18] Eli Biham, Louis Granboulan, and Phong Q. Nguyen. Impossible fault analysis of RC4 and differential fault analysis of RC4. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 359–367. Springer, 2005.

[19] Eli Biham and Jennifer Seberry. Py (Roo): A fast and secure stream cipher using rolling arrays. *IACR Cryptology ePrint Archive*, 2005:155, 2005.

[20] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[21] Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round DES. In Ernest F. Brickell, editor, *CRYPTO*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496. Springer, 1992.

[22] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[23] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2000.

[24] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In Bruce Schneier, editor, *FSE*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2000.

[25] Richard E. Blahut. *Principles and Practice of Information Theory*. Addison-Wesley, 1983.

[26] Bluetooth™. Bluetooth specification, v4.0, June 2010. E0 encryption algorithm described in volume 2, pages 1072–1081. Available online at http://www.bluetooth.org.

[27] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[28] Marc Briceno, Ian Goldberg, and David Wagner. A pedagogical implementation of the GSM A5/1 and A5/2 "voice privacy" encryption algorithms. Available online at http://www.scard.org/gsm/a51.html, 1998.

[29] Anne Canteaut and Michaël Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In Bart Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588. Springer, 2000.

[30] Anupam Chattopadhyay, Ayesha Khalid, Subhamoy Maitra, and Shashwat Raizada. Designing high-throughput hardware accelerator for stream cipher HC-128. In *ISCAS*, pages 1448–1451. IEEE, 2012.

[31] Jiageng Chen and Atsuko Miyaji. How to find short RC4 colliding key pairs. In Xuejia Lai, Jianying Zhou, and Hui Li, editors, *ISC*, volume 7001 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2011.

[32] Philippe Chose, Antoine Joux, and Michel Mitton. Fast correlation attacks: An algorithmic point of view. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 209–221. Springer, 2002.

[33] Nicolas Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 176–194. Springer, 2003.

[34] Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2003.

[35] Joan Daemen and Paris Kitsos. The self-synchronizing stream cipher MOSQUITO. Submission to ECRYPT call for stream ciphers, 2005. Report 2005/018, eSTREAM, the ECRYPT Stream Cipher Project.

[36] Joan Daemen and Paris Kitsos. The self-synchronizing stream cipher moustique. In Matthew J. B. Robshaw and Olivier Billet, editors, *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 210–223. Springer, 2008.

[37] Christophe Devine and Thomas d'Otreppe. Aircrack-ng. Available online at http://www.aircrack-ng.org/.

[38] Orr Dunkelman and Nathan Keller. Treatment of the initial value in time-memory-data tradeoff attacks on stream ciphers. *Inf. Process. Lett.*, 107(5):133–137, 2008.

[39] Patrik Ekdahl and Thomas Johansson. Another attack on A5/1. *IEEE Transactions on Information Theory*, 49(1):284–289, 2003.

[40] ECRYPT Stream Cipher Project eSTREAM. The current eSTREAM portfolio. Available online at http://www.ecrypt.eu.org/stream/index.html.

[41] ECRYPT Stream Cipher Project eSTREAM. Software performance results from the eSTREAM project. Available online at http://www.ecrypt.eu.org/stream/perf/#results.

[42] Hal Finney. An RC4 cycle that can't happen. Post in sci.crypt, 1994.

[43] Wieland Fischer, Berndt M. Gammel, O. Kniffler, and J. Velten. Differential power analysis of stream ciphers. In Masayuki Abe, editor, *CT-*

RSA, volume 4377 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 2007.

[44] Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2001.

[45] Scott R. Fluhrer and David A. McGrew. Statistical analysis of the alleged RC4 keystream generator. In Bruce Schneier, editor, *FSE*, volume 1978 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 2000.

[46] Michalis D. Galanis, Paris Kitsos, Giorgos Kostopoulos, Nicolas Sklavos, and Constantinos E. Goutis. Comparison of the hardware implementation of stream ciphers. *Int. Arab J. Inf. Technol.*, 2(4):267–274, 2005.

[47] Ian Goldberg and David Wagner. Lucky green – the (real-time) cryptanalysis of A5/2. Presented at the Rump Session of Crypto, 1999.

[48] Jovan Dj. Golic. Intrinsic statistical weakness of keystream generators. In Josef Pieprzyk and Reihaneh Safavi-Naini, editors, *ASIACRYPT*, volume 917 of *Lecture Notes in Computer Science*, pages 91–103. Springer, 1994.

[49] Jovan Dj. Golic. Towards fast correlation attacks on irregularly clocked shift registers. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *EUROCRYPT*, volume 921 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 1995.

[50] Jovan Dj. Golic. Correlation properties of a general binary combiner with memory. *J. Cryptology*, 9(2):111–126, 1996.

[51] Jovan Dj. Golic. Cryptanalysis of alleged A5 stream cipher. In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 1997.

[52] Jovan Dj. Golic. Linear statistical weakness of alleged RC4 keystream generator. In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 226–238. Springer, 1997.

[53] Jovan Dj. Golic. Linear models for a time-variant permutation generator. *IEEE Transactions on Information Theory*, 45(7):2374–2382, 1999.

[54] Jovan Dj. Golic. Iterative probabilistic cryptanalysis of RC4 keystream generator. In Ed Dawson, Andrew Clark, and Colin Boyd, editors, *ACISP*, volume 1841 of *Lecture Notes in Computer Science*, pages 220–233. Springer, 2000.

[55] Jovan Dj. Golic. Correlation analysis of the shrinking generator. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 440–457. Springer, 2001.

[56] Jovan Dj. Golic and Miodrag J. Mihaljevic. A generalized correlation attack on a class of stream ciphers based on the Levenshtein distance. *J. Cryptology*, 3(3):201–212, 1991.

[57] Jovan Dj. Golic and Guglielmo Morgari. Iterative probabilistic reconstruction of RC4 internal states. *IACR Cryptology ePrint Archive*, 2008:348, 2008.

[58] Guang Gong, Kishan Chand Gupta, Martin Hell, and Yassir Nawaz. Towards a general RC4-like keystream generator. In Dengguo Feng, Dongdai Lin, and Moti Yung, editors, *CISC*, volume 3822 of *Lecture Notes in Computer Science*, pages 162–174. Springer, 2005.

[59] Tim Good and Mohammed Benaissa. ASIC hardware performance. In Matthew J. B. Robshaw and Olivier Billet, editors, *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 267–293. Springer, 2008.

[60] Alexander L. Grosul and Dan S. Wallach. A related-key cryptanalysis of RC4. Technical Report TR-00-358, Department of Computer Science, Rice University, 2000.

[61] Frank K. Gurkaynak, Peter Luethi, Nico Bernold, Rene Blattmann, Victoria Goode, Marcel Marghitola, Hubert Kaeslin, Norbert Felber, and Wolfgang Fichtner. Hardware evaluation of eSTREAM candidates: Achterbahn, Grain, MICKEY, MOSQUITO, SFINKS, Trivium, VEST,

ZK-Crypt. Report 2006/015, eSTREAM, the ECRYPT Stream Cipher Project, 2006.

[62] Panu Hamalainen, Marko Hannikainen, Timo Hamalainen, and Jukka Saarinen. Hardware implementation of the improved WEP and RC4 encryption algorithms for wireless terminals. In *European Signal Processing Conference*, pages 2289–2292, 2000. Available online at http://www.eurasip.org/Proceedings/Eusipco/Eusipco2000/SESSIONS/FRIPM/SS3/CR1539.PDF.

[63] Philip Hawkes and Gregory G. Rose. Rewriting variables: The complexity of fast algebraic attacks on stream ciphers. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 390–406. Springer, 2004.

[64] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. The grain family of stream ciphers. In Matthew J. B. Robshaw and Olivier Billet, editors, *The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 2008.

[65] Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *IJWMC*, 2(1):86–93, 2007.

[66] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.

[67] Jonathan J. Hoch and Adi Shamir. Fault analysis of stream ciphers. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 240–253. Springer, 2004.

[68] Michal Hojsík and Bohuslav Rudolf. Differential fault analysis of Trivium. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2008.

[69] Jin Hong and Palash Sarkar. New applications of time memory data tradeoffs. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 353–372. Springer, 2005.

[70] Takanori Isobe, Toshihiro Ohigashi, Yuhei Watanabe, and Masakatu Morii. Full plaintext recovery attack on broadcast RC4. In *Fast Software Encryption (FSE)*, 2013. To appear.

[71] Thomas Johansson and Fredrik Jönsson. Fast correlation attacks based on turbo code techniques. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 1999.

[72] Thomas Johansson and Fredrik Jönsson. Improved fast correlation attacks on stream ciphers via convolutional codes. In Jacques Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 347–362. Springer, 1999.

[73] Antoine Joux and Frédéric Muller. Loosening the KNOT. In Thomas Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 87–99. Springer, 2003.

[74] Antoine Joux and Frédéric Muller. Chosen-ciphertext attacks against MOSQUITO. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 390–404. Springer, 2006.

[75] Robert J. Jenkins Jr. ISAAC and RC4. Published on the Internet at `http://burtleburtle.net/bob/rand/isaac.html`, 1996.

[76] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[77] Shahram Khazaei and Willi Meier. On reconstruction of RC4 keys from internal states. In Jacques Calmet, Willi Geiselmann, and Jörn Müller-Quade, editors, *MMICS*, volume 5393 of *Lecture Notes in Computer Science*, pages 179–189. Springer, 2008.

[78] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE public key cryptosystem by relinearization. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1999.

[79] Paraskevas Kitsos, Giorgos K. Kostopoulos, Nicolas Sklavos, and Odysseas G. Koufopavlou. Hardware implementation of the RC4 stream cipher. In *46th IEEE Midwest Symposium on Circuits & Systems*, volume 3, pages 1363–1366, Cairo, Egypt, 2003. Available online at http://dsmc.eap.gr/en/members/pkitsos/papers/Kitsos_c14.pdf.

[80] Andreas Klein. Attacks on the RC4 stream cipher. *Des. Codes Cryptography*, 48(3):269–286, 2008. Published online in 2006, and accepted in WCC 2007 workshop.

[81] Lars R. Knudsen, Willi Meier, Bart Preneel, Vincent Rijmen, and Sven Verdoolaege. Analysis methods for (alleged) RC4. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT*, volume 1514 of *Lecture Notes in Computer Science*, pages 327–341. Springer, 1998.

[82] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[83] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[84] Korek. Need security pointers. Published online at http://www.netstumbler.org/showthread.php?postid=89036#pos%t89036, 2004.

[85] Korek. Next generation of WEP attacks? Published online at http://www.netstumbler.org/showpost.php?p=93942&postcount=%35, 2004.

[86] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951. Available online at http://projecteuclid.org/euclid.aoms/1177729694.

[87] Joseph Lano, Nele Mentens, Bart Preneel, and Ingrid Verbauwhede. Power analysis of synchronous stream ciphers with resynchronization mechanism. In *Workshop Record of SASC 2004 – The State of the Art of Stream Ciphers*, pages 327–333, 2004. Available online at http://www.ecrypt.eu.org/stvl/sasc/record.html.

[88] Gregor Leander, Erik Zenner, and Philip Hawkes. Cache timing analysis of LFSR-based stream ciphers. In Matthew G. Parker, editor, *IMA Int. Conf.*, volume 5921 of *Lecture Notes in Computer Science*, pages 433–445. Springer, 2009.

[89] Jun-Dian Lee and Chih-Peng Peng Fan. Efficient low-latency RC4 architecture designs for IEEE 802.11i WEP/TKIP. In *International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS) 2007*, pages 56–59, 2007.

[90] Yi Lu, Willi Meier, and Serge Vaudenay. The conditional correlation attack: A practical attack on Bluetooth encryption. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 97–117. Springer, 2005.

[91] Yi Lu and Serge Vaudenay. Cryptanalysis of Bluetooth keystream generator two-level E0. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 483–499. Springer, 2004.

[92] Yi Lu and Serge Vaudenay. Faster correlation attack on Bluetooth keystream generator E0. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 407–425. Springer, 2004.

[93] Yi Lu and Serge Vaudenay. Cryptanalysis of an E0-like combiner with memory. *J. Cryptology*, 21(3):430–457, 2008.

[94] Thomas Lynch and Earl E. Swartzlander Jr. A spanning tree carry lookahead adder. *IEEE Trans. Computers*, 41(8):931–939, 1992.

[95] Subhamoy Maitra and Goutam Paul. Analysis of RC4 and proposal of additional layers for better security margin. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2008.

[96] Subhamoy Maitra and Goutam Paul. New form of permutation bias and secret key leakage in keystream bytes of RC4. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 253–269. Springer, 2008.

[97] Subhamoy Maitra, Goutam Paul, Santanu Sarkar, Michael Lehmann, and Willi Meier. New results on generalization of Roos-type biases and related keystreams of RC4. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *AFRICACRYPT*, volume 7918 of *Lecture Notes in Computer Science*, pages 222–239. Springer, 2013.

[98] Subhamoy Maitra, Goutam Paul, and Sourav Sen Gupta. Attack on broadcast RC4 revisited. In Antoine Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 199–217. Springer, 2011.

[99] Subhamoy Maitra and Sourav Sen Gupta. New long-term glimpse of RC4 stream cipher. In Aditya Bagchi and Indrakshi Ray, editors, *ICISS*, volume 8303 of *Lecture Notes in Computer Science*, pages 230–238. Springer, 2013.

[100] Itsik Mantin. Analysis of the stream cipher RC4. Master's thesis, The Weizmann Institute of Science, Israel, 2001. Available online at `http://www.wisdom.weizmann.ac.il/~itsik/RC4/RC4.html`.

[101] Itsik Mantin. A practical attack on the fixed RC4 in the WEP mode. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 395–411. Springer, 2005.

[102] Itsik Mantin. Predicting and distinguishing attacks on RC4 keystream generator. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 491–506. Springer, 2005.

[103] Itsik Mantin and Adi Shamir. A practical attack on broadcast RC4. In Mitsuru Matsui, editor, *FSE*, volume 2355 of *Lecture Notes in Computer Science*, pages 152–164. Springer, 2001.

[104] Mitsuru Matsui. Key collisions of the RC4 stream cipher. In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 38–50. Springer, 2009.

[105] Donald P. Matthews, Jr. Methods and apparatus for accelerating ARC4 processing. Patent, 07 2008. US 7403615.

[106] Donald P. Matthews, Jr. and Campbell CA. System and method for a fast hardware implementation of RC4. Patent, 04 2003. US 6549622.

[107] Alexander Maximov and Dmitry Khovratovich. New state recovery attack on RC4. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 297–316. Springer, 2008.

[108] Matthew E. McKague. Design and analysis of RC4-like stream ciphers. Master's thesis, University of Waterloo, Canada, 2005.

[109] Willi Meier and Othmar Staffelbach. Fast correlation attacks on certain stream ciphers. *J. Cryptology*, 1(3):159–176, 1989.

[110] Willi Meier and Othmar Staffelbach. Correlation properties of combiners with memory in stream ciphers. *J. Cryptology*, 5(1):67–86, 1992.

[111] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, August 2011 edition, 1996. Fifth Printing.

[112] Ilya Mironov. (Not So) random shuffles of RC4. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 304–319. Springer, 2002.

[113] Serge Mister and Stafford E. Tavares. Cryptanalysis of RC4-like ciphers. In Stafford E. Tavares and Henk Meijer, editors, *Selected Areas in Cryptography*, volume 1556 of *Lecture Notes in Computer Science*, pages 131–143. Springer, 1998.

[114] Yassir Nawaz, Kishan Chand Gupta, and Guang Gong. A 32-bit RC4-like keystream generator. *IACR Cryptology ePrint Archive*, 2005:175, 2005.

[115] Kenneth G. Paterson, Bertram Poettering, and Jacob C.N. Schuldt. Plaintext recovery attacks against WPA/TKIP. *IACR Cryptology ePrint Archive*, 2013:748, 2013.

[116] Goutam Paul. *Analysis and Design of RC4 and Its Variants*. PhD thesis in Engineering, Jadavpur University, 2009. Work done at Indian Statistical Institute, thesis submitted at Jadavpur University.

[117] Goutam Paul and Subhamoy Maitra. Permutation after RC4 key scheduling reveals the secret key. In Carlisle M. Adams, Ali Miri, and

Michael J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 360–377. Springer, 2007.

[118] Goutam Paul and Subhamoy Maitra. On biases of permutation and keystream bytes of RC4 towards the secret key. *Cryptography and Communications*, 1(2):225–268, 2009.

[119] Goutam Paul and Subhamoy Maitra. *RC4 Stream Cipher and Its Variants*. CRC Press, Taylor & Francis Group, A Chapman & Hall Book, 2012.

[120] Goutam Paul, Subhamoy Maitra, and Rohit Srivastava. On non-randomness of the permutation after RC4 key scheduling. In Serdar Boztas and Hsiao feng Lu, editors, *AAECC*, volume 4851 of *Lecture Notes in Computer Science*, pages 100–109. Springer, 2007.

[121] Goutam Paul, Siddheshwar Rathi, and Subhamoy Maitra. On non-negligible bias of the first output byte of RC4 towards the first three bytes of the secret key. *Des. Codes Cryptography*, 49(1-3):123–134, 2008. Initial version in proceedings of WCC 2007.

[122] Souradyuti Paul and Bart Preneel. A new weakness in the RC4 keystream generator and an approach to improve the security of the cipher. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2004.

[123] Souradyuti Paul and Bart Preneel. On the (in)security of stream ciphers based on arrays and modular addition. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2006.

[124] Ronald L. Rivest. RSA security response to weaknesses in key scheduling algorithm of RC4. Technical note, RSA Data Security, Inc., 2001. [The structure of RC4 was never published officially, it was leaked in 1994 to the Internet. This note confirms that the leaked code is indeed RC4.].

[125] Andrew Roos. A class of weak keys in the RC4 stream cipher. Two posts in sci.crypt, message-id 43u1eh$1j3@hermes.is.co.za and

44ebge$llf@hermes.is.co.za, 1995. Available online at http://www.impic.org/papers/WeakKeys-report.pdf.

[126] Markku-Juhani Olavi Saarinen. A time-memory tradeoff attack against LILI-128. In Joan Daemen and Vincent Rijmen, editors, *FSE*, volume 2365 of *Lecture Notes in Computer Science*, pages 231–236. Springer, 2002.

[127] Santanu Sarkar. Further non-randomness in RC4, RC4A and VMPC. In *International Workshop on Coding and Cryptography (WCC)*, 2013.

[128] Santanu Sarkar, Sourav Sen Gupta, Goutam Paul, and Subhamoy Maitra. Proving TLS-attack related open biases of RC4. *IACR Cryptology ePrint Archive*, 2013:502, 2013.

[129] Sourav Sen Gupta, Anupam Chattopadhyay, Koushik Sinha, Subhamoy Maitra, and Bhabani P. Sinha. High-performance hardware implementation for RC4 stream cipher. *IEEE Trans. Computers*, 62(4):730–743, 2013.

[130] Sourav Sen Gupta, Subhamoy Maitra, Willi Meier, Goutam Paul, and Santanu Sarkar. Some results on RC4 in WPA. *IACR Cryptology ePrint Archive*, 2013:476, 2013.

[131] Sourav Sen Gupta, Subhamoy Maitra, Goutam Paul, and Santanu Sarkar. Proof of empirical RC4 biases and new key correlations. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 151–168. Springer, 2011.

[132] Sourav Sen Gupta, Subhamoy Maitra, Goutam Paul, and Santanu Sarkar. (Non–)random sequences from (non–)random permutations – analysis of RC4 stream cipher. *Journal of Cryptology*, 2013. To appear. Published online in December 2012. DOI: 10.1007/s00145-012-9138-1.

[133] Sourav Sen Gupta, Koushik Sinha, Subhamoy Maitra, and Bhabani P. Sinha. One byte per clock: A novel RC4 hardware. In Guang Gong and Kishan Chand Gupta, editors, *INDOCRYPT*, volume 6498 of *Lecture Notes in Computer Science*, pages 347–363. Springer, 2010.

[134] Pouyan Sepehrdad. *Statistical and Algebraic Cryptanalysis of Lightweight and Ultra-Lightweight Symmetric Primitives*. PhD thesis No. 5415, École Polytechnique Fédérale de Lausanne (EPFL), 2012. Available online at http://lasecwww.epfl.ch/~sepehrdad/Pouyan_Sepehrdad_PhD_Thesis.pdf.

[135] Pouyan Sepehrdad, Petr Susil, Serge Vaudenay, and Martin Vuagnoux. Smashing WEP in a passive attack. In *Fast Software Encryption (FSE)*, 2013. To appear.

[136] Pouyan Sepehrdad, Serge Vaudenay, and Martin Vuagnoux. Discovery and exploitation of new biases in RC4. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 74–91. Springer, 2010.

[137] Pouyan Sepehrdad, Serge Vaudenay, and Martin Vuagnoux. Statistical attack on RC4 - distinguishing WPA. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 343–363. Springer, 2011.

[138] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

[139] Yoshiaki Shiraishi, Toshihiro Ohigashi, and Masakatu Morii. An improved internal-state reconstruction method of a stream cipher RC4. In M.H. Hamza, editor, *Communication, Network, and Information Security, Track 440–088*, New York, USA, December 2003.

[140] Bhabani P. Sinha and Pradip K. Srimani. Fast parallel algorithms for binary multiplication and their implementation on systolic architectures. *IEEE Trans. Computers*, 38(3):424–431, 1989.

[141] François-Xavier Standaert, Eric Peeters, Cédric Archambeau, and Jean-Jacques Quisquater. Towards security limits in side-channel attacks. In Louis Goubin and Mitsuru Matsui, editors, *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 2006.

[142] Douglas R. Stinson. *Cryptography: Theory and Practice*. CRC Press, third (November 2005) edition, 1995.

[143] Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. Using the Fluhrer, Mantin, and Shamir attack to break WEP. In *NDSS*. The Internet Society, 2002.

[144] Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. A key recovery attack on the 802.11b wired equivalent privacy protocol (WEP). *ACM Trans. Inf. Syst. Secur.*, 7(2):319–332, 2004.

[145] Erik Tews. Attacks on the WEP protocol. *IACR Cryptology ePrint Archive*, 2007:471, 2007.

[146] Erik Tews and Martin Beck. Practical attacks against WEP and WPA. In David A. Basin, Srdjan Capkun, and Wenke Lee, editors, *WISEC*, pages 79–86. ACM, 2009.

[147] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit WEP in less than 60 seconds. In Sehun Kim, Moti Yung, and Hyung-Woo Lee, editors, *WISA*, volume 4867 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2007.

[148] Violeta Tomasevic, Slobodan Bojanic, and Octavio Nieto-Taladriz. Finding an internal state of RC4 stream cipher. *Inf. Sci.*, 177(7):1715–1727, 2007.

[149] Serge Vaudenay and Martin Vuagnoux. Passive-only key recovery attacks on RC4. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 344–359. Springer, 2007.

[150] David A. Wagner. My RC4 weak keys. Post in sci.crypt, message-id 447o1l$cbj@cnn.Princeton.EDU, 1995. Available online at http://www.cs.berkeley.edu/~daw/my-posts/my-rc4-weak-keys.

[151] Hongjun Wu and Bart Preneel. Differential cryptanalysis of the stream ciphers Py, Py6 and Pypy. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 276–290. Springer, 2007.

[152] Hongjun Wu and Bart Preneel. Differential-linear attacks against the stream cipher Phelix. In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 87–100. Springer, 2007.

[153] Erik Zenner. A cache timing analysis of HC-256. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2008.

[154] Bartosz Zoltak. VMPC one-way function and stream cipher. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 210–225. Springer, 2004.