A Survey on Some of the Existing Attacks on RC4

a dissertation submitted in partial fulfillment of the requirements
for the Master of Technology in Computer Science degree
of the Indian Statistical Institute

by

Atanu Acharyya
(mtc0509)

under the supervision of

Prof Subhamoy Maitra

Indian Statistical Institute
203, Barrackpore Trunk Road
Kolkata, 700 108

CERTIFICATE OF APPROVAL

This is to certify that the thesis entitled "*A Survey on some of the Existing Attacks on RC4"* is submitted by Atanu Acharyya towards partial fulfillment of the requirement for the award of the degree of the Masters in Technology in Computer Science at Indian Statistical Institute, Kolkata, embodies the work done under my supervision.

Dated: July 20, 2007

Supervisor: ______________________________

Professor Subhamoy Maitra

.

Acknowledgements

With great pleasure and sense of obligation I express my heartfelt gratitude to my guide and supervisor Prof Subhamoy Maitra of Applied Statistics Unit, Indian Statistical Institute, Kolkata. I thank him for the guidance, for the endless patience that was required for his precise revisions of my work, for his support when things seemed not to work out.

I thank all Computer Science graduates of Indian Statistical Institute for being enjoyable comrades. I thank them for making my time at ISI a most pleasant one, and for the long and interesting discussions on academic and non-academic issues. Special thanks to Kaushik Nath for his all-round support.

Lastly I sincerely thank all my friends and well wishers who helped me directly or indirectly towards the completion of this work.

I owe a lot to the Indian Statistical Institute for giving me the opportunity to providing me with the best environment for doing that.

Atanu Acharyya

Abstract

RC4 is the most widely deployed stream cipher in software applications, due to its simplicity and efficiency. It has a huge internal state but it has very light-weight key scheduling and output generation processes, which motivated our cryptanalytic efforts.

In this thesis we analyze the KSA (key scheduling algorithm) of RC4, and describe several weaknesses in it. We identify a large number of weak keys, in which knowledge of a small number of key bits suffices to determine many state and output bits with non-negligible probability. We use these weak keys to construct new distinguishers for RC4, and to mount related key attacks with practical complexities.

Another weakness of RC4 initialization mechanism is a major statistical bias in the distribution of the first output words. This bias makes it trivial to distinguish between several hundred short outputs of RC4 and random strings by analyzing their second word. This weakness can be used to mount a practical ciphertextonly attack on RC4 in some broadcast applications, in which the same plaintext is sent to multiple recipients under different keys. This unique statistical behavior is independent of the KSA, and remains applicable even when RC4 starts with a totally random permutation.

Preface

My first meeting with RC4 was due to a lecture as part of our course "Cryptology" in the 3rd semester, delivered by Prof Subhamoy Maitra at the Indian Statistical Institute. I had no idea then that this cipher is one of the most popular in the world. The algorithm was unexpectedly short in size with a great simplicity. Later I came to know about its huge application power in the world of secure e-communication. The futile attacks mounted on it so far bears its class in the family of all the existing stream ciphers. By the middle of our course work I got the opportunity from Prof Subhomay Maitra to work on RC4 under his guidance. Later, after looking deeply at previous work on RC4 published by other researchers, I was amazed to figure out that most of it was quite rudimentary culminating in a paper by Fluhrer and Mcgrew that described a distinguishing algorithm which was based on simple counting of output pairs. I figured out that RC4 still had not received the attention it deserves.

Since there was a lot of pressure due to the course works during the final semester, I had never been able to concentrate much on RC4. Still I have tried to understand its nature and existing cryptanalysis, applying them analytically and experimentally, failing most of the time but succeeding here and there, though finally unable to mount an attack of my own. Eventually, I managed to analyze and implement some of the existing techniques for analyzing this unique cipher, pointing out both its weaknesses and its strength. This is my modest contribution to the world of cryptography.

List of Tables

List of Figures

Table of Contents

Chapter 1

Introduction

1.1 Introduction to Cryptography

Cryptography is a remarkable field in computer science which deals with very human issues such as of privacy, authenticity, and trust. The word "cryptography" comes from the Latin *crypt*, meaning *secret*, and *graphia*, meaning *writing*. So "cryptography" is literally "*secret writing*": the study of how to obscure what you write so as to make it unintelligible to those who should not read it.

Mathematically to say, a *cryptosystem* is a five-touple (P,C,K,E,D), where the following conditions are satisfied:
1. P is a finite set of possible *plaintexts*
2. C is a finite set of possible *ciphertext*
3. K, the keyspace, is a finite set of possible *keys*
4. For each $k \in K$, there is an *encryption rule* $e_k \in E$ and a corresponding *decryption rule* $d_k \in D$. Each $e_k : P \rightarrow C$ and $d_k : C \rightarrow P$ are functions such that $d_k(e_k(x)) = x \ \forall x \in P$.

In the last few decades cryptographic algorithms, being mathematical by nature, have become sufficiently advanced that they can only be handled by computers. The encryption scheme that pioneered the modern age of cryptography was the Digital Encryption Standard (DES), which was designed in IBM laboratories at the early seventies. Although there were many ciphers which were designed and used before DES, the only similarity between ciphers like Caesar's code or Enigma on one side, and DES and Rijndeal on the other side, is that all of them are solutions to the same fundamental problem.

1.1.1 Symmetric and Asymmetric Schemes

Encryption schemes are divided into two main types, symmetric schemes (sometimes called secret-key schemes) and asymmetric schemes (sometimes called public-key schemes). The main difference between these types is the requirement for a shared piece of secret information (the key) between the encryptor and the decryptor in symmetric schemes. Symmetric and asymmetric encryption schemes have various advantages and disadvantages, some of which are common to both of them.

Advantages of Symmetric-Key Cryptography

1. Throughput rates for the most popular asymmetric encryption methods are several orders of magnitude slower than those of the best known symmetric schemes.

2. Key sizes for asymmetric schemes are typically much longer than those required for symmetric schemes.

3. Symmetric ciphers can be employed as primitives to construct various cryptographic mechanisms including pseudorandom number generators, hash functions, and computationally efficient digital signature schemes, to name just a few.

4. Symmetric ciphers can be composed to produce stronger ciphers. Simple transformations, which are easy to analyze, but on their own weak, can be used to construct strong product ciphers.

Advantages of Asymmetric-Key Cryptography

1. In a two-party communication, there is only one secret key, and this key is generated and used by the same party. The key agreement stage can be carried out using regular channels, and requires no "out of band" interaction.

2. In order to achieve pairwise privacy in a large network, a symmetric cipher would require a quadratic number of keys (one key per pair), while an asymmetric cipher would require a linear number of keys (one key per user).

3. Depending on the mode of operation, the keys of an asymmetric scheme may remain unchanged for a considerable period of time. For symmetric schemes, it is common practice to change keys frequently, sometimes for each communication session.

Summary of Comparison

Symmetric and asymmetric encryption have a number of complementary advantages. The most significant disadvantage of asymmetric schemes is their low efficiency, while the most significant advantages are the ability to securely communicate without any previous interaction, and the relatively simple key management (a linear number of keys which are rarely changed). Current cryptographic systems exploit their strengths by integrating both types into protocols that run in two phases. During the handshaking phase, the parties use an asymmetric encryption technique to set up a connection and to establish a symmetric key. During the second phase, this key is used for an efficient interaction using a symmetric scheme.

If this two phase protocol is executed for every session, the parties take advantage of the long term nature of the keys of the asymmetric scheme and the efficiency of the symmetric scheme, since the asymmetric part of the protocol is a small fraction of the total encryption time.

1.1.2 Stream Ciphers vs Block ciphers

Stream ciphers are an important class of encryption algorithms. They encrypt individual characters (usually binary digits) of a plaintext message one at a time, using a simple time-dependent encryption transformation.

Block ciphers simultaneously encrypt groups of characters of a plaintext message using a fixed encryption transformation. Stream ciphers are generally faster than block ciphers in hardware, and have less complex hardware circuitry. The Blum-Goldwasser probabilistic public-key encryption scheme described in [6] is an example of a asymmetric stream cipher. However, most stream ciphers are based on symmetric schemes. Block ciphers are memoryless whereas in stream ciphers the encryption function may vary as the plaintext is processed. Stream ciphers are sometimes called state ciphers since encryption depends not only on the key and plaintext, but also on the current state. This distinction between block and stream ciphers is not definitive; adding a small amount of memory to a block cipher (as in the CBC mode) results in a stream cipher with large blocks.

Stream ciphers are more appropriate, and in some cases mandatory (e.g., in some telecommunications applications), when buffering is limited or when characters must be individually processed as they are received. Because they have limited or no error propagation, stream ciphers may also be advantageous in situations where transmission errors are highly probable. There is a vast body of theoretical knowledge on stream ciphers, and various design principles for stream ciphers have been proposed and extensively analyzed. However, there are relatively few fully-specified stream cipher algorithms in the open literature. This unfortunate state of affairs can be partially explained by the fact that most stream ciphers used in practice tend to be proprietary and confidential. By contrast, numerous concrete block cipher proposals have been published, some of which have been standardized or placed in the public domain. Nevertheless, because of their significant advantages, stream ciphers are widely used today, and one can expect many more proposals in the coming years.

1.1.3 Stream Ciphers

The only encryption scheme that is information theoretically secure is the Vernam cipher, or in its more popular name, the One Time Pad scheme (OTP). Using this scheme requires a key that is as long as the message, and the ciphertext is produced by XORing the plaintext with the key. An obvious drawback of the OTP is that the huge key length increases the difficulty of key distribution and storage. This motivates the design of stream ciphers in which the keystream is pseudorandomly generated from a smaller secret key (seed), so that the keystream appears random to a computationally bounded adversary.

Stream ciphers are commonly classified as being *synchronous* or *asynchronous*. A synchronous stream cipher is one in which the keystream is generated independently of the plaintext message and of the ciphertext. An asynchronous stream cipher (denoted also as a self-synchronizing stream cipher) is one in which the keystream is generated as a function of the key and a fixed number of previous ciphertext digits. The main difference between these types is the ability of the self synchronizing stream ciphers to continue the decryption, even when some part of the ciphertext was lost. However most of the popular stream ciphers are of the synchronous type, and in particular RC4 is such.

An inherent property of stream ciphers is the absolute loss of security when encrypting more than one message with the same key. A single key k produces a single keystream Z, and knowing the encryptions e_1 and e_2 of the messages m_1 and m_2, makes it easy to calculate

$$e_1 \oplus e_2 = (Z \oplus m_1) \oplus (Z \oplus m_2) = (Z \oplus Z) \oplus (m_1 \oplus m_2) = m_1 \oplus m_2$$

which contains significant information about the plaintexts. There are two main approaches to overcome this problem; the first one is to use a single stream, and the second is to change keys after every session. To implement the first approach, the parties must share the exact position in the stream, from which the next keystream words are going to be taken for the next session. However, communication problems may cause loss of data and loss of synchronization, and when this happens the parties cannot communicate until they re-synchronize. There are keystream generators that have the Random Access property, which means that it is possible to reach every point in the stream, within a logarithmic amount of time. For schemes of this type, it is possible to divide the stream into long finite intervals, and use one interval per session. An example to such a stream cipher is Leviathan, designed by Fluhrer and McGrew ([10]).

The second approach is more popular, and includes a mechanism that takes as its input the secret key and an additional piece of data, and derives a session-key from them. This piece of data is usually called as the Initialization Vector, and is usually transmitted in the clear (see Figure 1.1).

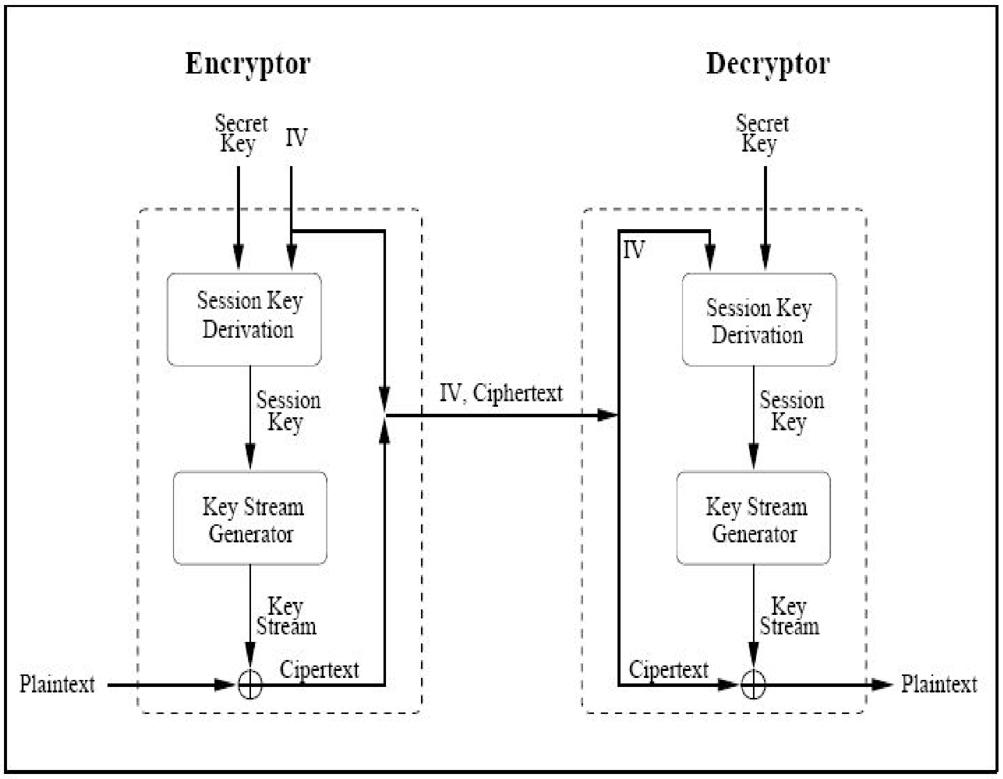

Figure 1.1: Typical Key Management in stream ciphers

The complexity of the session-key derivation varies from simple methods such as concatenation and XORing, to complex methods which hash the two values. The hash function approach preserves the security of most ciphers, but the concatenation approach sometimes reduces their security. An example of this phenomenon is the IV attack on RC4. Even though there are no known practical attacks on this stream cipher, it becomes extremely weak when combined with a concatenation-based session-key derivation mechanism.

1.2 The RC4 Stream Cipher

A large number of stream ciphers were proposed and implemented over the last twenty years. Most of these ciphers were based on various combinations of linear feedback shift registers, which were easy to implement in hardware, but relatively slow in software. In 1987 Ron Rivest designed the RC4 stream cipher, which was based on a different and more software friendly paradigm. RC4 is most commonly used to protect Internet traffic using the SSL (Secure Sockets Layer) protocol. Moreover, it was integrated into Microsoft Windows, Lotus Notes, Apple AOCE, Oracle Secure SQL, and many other applications. In addition, it was chosen to be part of the Cellular Digital Packet Data specification. Indeed, these uses of RC4 may make RC4 the most widely-used stream cipher in the world. Its design was kept a trade secret until 1994, when someone anonymously posted its source code to the Cypherpunks mailing list. The correctness of this unofficial description was confirmed by comparing its outputs to those produced by licensed implementations.

RC4 has a secret internal state which is a permutation $S \in S[N]$ of all the $N = 2^n$ possible n bits words, and two indices $i, j \in [N]$ in it. The initial state is derived from a variable-size key by a Key-Scheduling Algorithm (KSA), and then RC4 alternately modifies the state (by exchanging two out of the N values) and produces an output (by picking one of the N values).

In practical applications n is typically chosen as 8, and thus RC4 has a huge state of

$$log_2 (|[256]|^2 . |S_{256}|) = log_2 (2^{16} .256!) \approx 1700$$

bits. It is thus impractical to guess even a small part of this state, or to use standard time/memory/data tradeoff attacks. In addition, the state evolves in a complex nonlinear way, and thus it is difficult to combine partial information about states that are far away in time. Consequently, all the techniques developed to attack stream ciphers based on linear feedback shift registers seem to be inapplicable to RC4.

Since RC4 is such a widely used stream cipher, it had attracted considerable attention in the research community, but so far no one had found an attack on RC4 which is even close to being practical: For $n = 8$ and sufficiently long keys, the best known attack requires more than 2700 time to find its initial state.

1.2.1 Description of RC4

RC4 consists of two parts (described in Figure 1.2): A Key-Scheduling Algorithm KSA which turns a random key (whose typical size is 40-256 bits) of ℓ words into an initial permutation $S \in S_N$, and a pseudo-random generation part PRGA which uses this permutation to generate a pseudo-random output sequence.

The PRGA initializes two indices *i* and *j* to 0, and then loops over four simple operations which increment *i* as a counter, increment *j* pseudo randomly, exchange the two values of *S* pointed to by *i* and *j*, and output the value of *S* pointed to by $S[i] + S[j]$. Note that every entry of *S* is swapped at least once (possibly with itself) within any *N* consecutive rounds, and thus the permutation *S* evolves fairly rapidly during the output generation process.

The KSA consists of *N* loops which are similar to the PRGA round operation. It initializes *S* to be the identity permutation and *i* and *j* to 0, and applies a PRGA-like round operation *N* times, stepping *i* across *S*, and updating *j* by adding $S[i]$ and the next word of the key (in cyclic order).

```
KSA(K)
Initialization:
    $S \leftarrow \langle 0, 1, \ldots, N-1 \rangle$
    $j \leftarrow 0$
Scrambling:
    For $i \leftarrow 0 \ldots N-1$
        $j \leftarrow j + S[i] + K[i \bmod \ell]$
        $S[i] \leftrightarrow S[j]$
```

```
PRGA(S)
Initialization:
    $i \leftarrow 0$
    $j \leftarrow 0$
Generation loop:
    $i \leftarrow i + 1$
    $j \leftarrow j + S[i]$
    $S[i] \leftrightarrow S[j]$
    Output $z \leftarrow S[S[i] + S[j]]$
```

Figure 1.2: The Key-Scheduling Algorithm and the Pseudo-Random Generation Algorithm

1.2.2 Related Work

Attacks on the Keystream Generation

A branch and bound attack which is based on the "Guess on Demand" paradigm is analyzed in [4] and [9]. The attack simulates the generation process, and keeps track of all the known values in S that had been deduced so far. Whenever an unknown entry in S is needed in order to continue the simulation, the attacker tries all the possible values (or guesses a value). Notice that the number of trials is typically smaller than N since known values in the permutation S cannot be repeated. Actual outputs are used by the simulation to either deduce additional values in S (if the pointed output value is unknown) or to backtrack when the pointed value is known and different from the actual output, thus contradicting this branch.

The time complexity of this attack was analyzed analytically in [9], and the analytic results were found compatible with experimental estimations.

This tree search is simple to implement, and needs only N output words. However, it is very inefficient in time, and its enormous running time, makes it worse than exhaustive search for typical key sizes, and completely impractical for $\mathrm{RC4}_{n>4}$. For redundant regions of the stream, branches representing incorrect guesses tend to shorten, and the time complexity improves, threatening the security of $\mathrm{RC4}_5$.

Distinguishers of RC4 Streams from Randomness

A different research direction was to analyze the statistical properties of RC4 outputs, and in particular to construct distinguishers between RC4 and truly random bit generators. Goli'c described in [5] a linear statistical weakness of RC4, caused by a positive correlation between the second binary derivative of the LSB and 1. This weakness implies that RC4 outputs of length $2^{6n-7.8}$ ($2^{40.2}$ for the typical $n = 8$) can be reliably distinguished from random strings. This result was subsequently improved by Fluhrer and McGrew in [3]. They analyzed the distribution of triplets consisting of the two outputs produced at times t and $t + 1$ and the known value of $i \equiv t \pmod N$), and found small biases in the distribution of $(7N - 8)$ of these N^3 triplets: some of these probabilities are positively biased ($1/N^3 .(1 + 1/N)$), and some of these probabilities are negatively biased ($1/N^3 .(1 - 1/N)$). They used information theoretic methods to prove that these biases can be used to distinguish between RC4 and a truly random source by analyzing sequences of $2^{30.6}$ output words. This number of output words was estimated rigorously, considering a success probability of 90%.

Their analysis of this typical behavior of RC4 streams yielded a classification of special RC4 states which they denoted as fortuitous states, which are the source of most of these biases. They noted that these states can be used to extract parts of the internal state with non-trivial probability, but since RC4 states are huge this does not lead to practical attacks on RC4 for $n > 5$.

Weaknesses of Initialization Mechanism of RC4

The major difference between the effective key size of RC4 and the real key, with the simplicity of the key extension mechanism, stimulated considerable research on the initialization mechanism of RC4.

In particular, Andrew Roos noted in [11] that for keys which have $K[0] + K[1] \equiv 0 \pmod N$, the first output is equal to $K[2] + 3$ with probability $2^{-2.85}$. The cryptanalyst can use this fact to deduce two bytes of information about the key ($K[0] + K[1]$ and $K[2]$) with probability $2^{-10:85}$ instead of the trivial 2^{-16}, and thus reduce the effective key length in exhaustive search by about five bits.

Grosul and Wallach showed in [8] that for large keys whose size is close to N bytes, similar keys produce similar initial states and similar streams. This observation implies that for large keys, RC4 is vulnerable to a related key attack. However, for typical short keys, every byte of the key is used fairly early during the KSA execution, and consequently a change in one byte, causes a difference in the index j from a relatively early stage. The difference in the values of j implies that different entries are swapped in the rounds that succeed this change. This causes significant difference in the output of the KSA (which is the initial permutation) and in the generated stream.

Other Results

Interesting properties of RC4 were described in several papers. Finney specified in [7] a class of states that RC4 can never enter. This class contains all the states for which $i = a, j = a + 1$ and $S[a + 1] = 1$ (a fraction of $1/N^2$ of RC4 states are in this class). Analysis of states of this type indicates that this class is closed under RC4 round operation. Given some arbitrary state in this class, corresponding to the value a, the round operation will transfer this state into another state in the class, corresponding to $a' = a + 1$. The round operation will give $i' = i + 1 = a + 1 = a'$, $j' = j + S[i'] = (a + 1) + S[a + 1] = (a + 1) + 1 = a' + 1$ (satisfying the first condition), and the swap will transfer the value 1 from a' to $a' + 1$, actually implying $S[a' + 1] = 1$ and satisfying the second condition. Thus these states are connected by short cycles of length $N(N - 1)$. Since the state transition in RC4 is invertible and the initial state ($i = j = 0$) is not of this type, RC4 can never enter these states for any key. However, if RC4 would have been initialized to such a state, it would be totally insecure. The generated stream would have a very short period. Given such an RC4 stream, Pudovkina noticed that the initial permutation can be elegantly extracted, by looking at the outputs with indices that are equivalent to $(-1) \bmod N - 1$ ($z_{(N-1)-1}, z_{2(N-1)-1}, \ldots\ldots\ldots, z_{N(N-1)-1}$). It appears that these N output values are a shift of the initial permutation, and since one of the permutation values is known ($S_{i+1} = 1$), the exact shift can be also isolated.

1.3 General Notations and Conventions

We denote the number of bits in a keystream word as n, and the size of the permutation S by $N := 2^n$. ℓ stands for the number of key words and whenever we discuss an RC4 version with a specified n and ℓ, we add a subscript indicating these parameters (e.g. $\mathrm{RC4}_{8,16}$ is the commonly used 8-bit version with a 128-bit key). Sometimes we omit ℓ (typically when it is irrelevant to the discussion) and write $\mathrm{RC4}_8$, or even simply RC4.

To denote the symmetric group of permutations of $\{0,\ldots\ldots,N - 1\}$ we use the standard notation S_N. The set of indices in $S \in S_N$ is denoted by $[N]$.

$S \in S_N$, $i \in [N]$ and $j \in [N]$ are used to specify the components of RC4 states and r is usually used to indicate round numbers (the last four notations are used for KSA and PRGA). We use the triplets (S, i, j) to indicate the permutation and indices of specific states.

The rounds of the KSA as well as those of the PRGA are numbered according to the value of i, which means that the KSA has rounds $\langle 0, \ldots\ldots, N-1 \rangle$, whereas the PRGA has rounds 1, 2,…….. To indicate RC4 state after round r (which is the state after $r + 1$ KSA rounds or after r PRGA rounds), we use the subscript r in S_r, i_r and j_r.

Chapter 2

A Practical Attack on Broadcast RC4

In this chapter we describe a major statistical bias in the distribution of the initial bytes of RC4 streams, and discuss its cryptanalytic applications.

2.1 The Bias

Our main observation is that the second output word of RC4 has a very strong bias, viz. it takes on the value 0 with twice the expected probability ($2/N$ instead of $1/N$). Other values of the second output and all the values of other outputs have almost uniform distributions.

2.1.1 The Biased Second Output of RC4

Theorem 2.1.1 *Assume that the initial permutation S is randomly chosen from the set of all the possible permutations of S_N. Then the probability that the second output word of RC4 is 0 is approximately $2/N$.*

Proof: Notation: Denote the permutation S after it has been updated in round t by S_t (S_0 is the initial permutation) and the output of this round as z_t.

Claim 2.1.2 *When $S_0[2] = 0$ and $S_0[1] \neq 2$, the second output(z_2) is 0 with probability 1.*

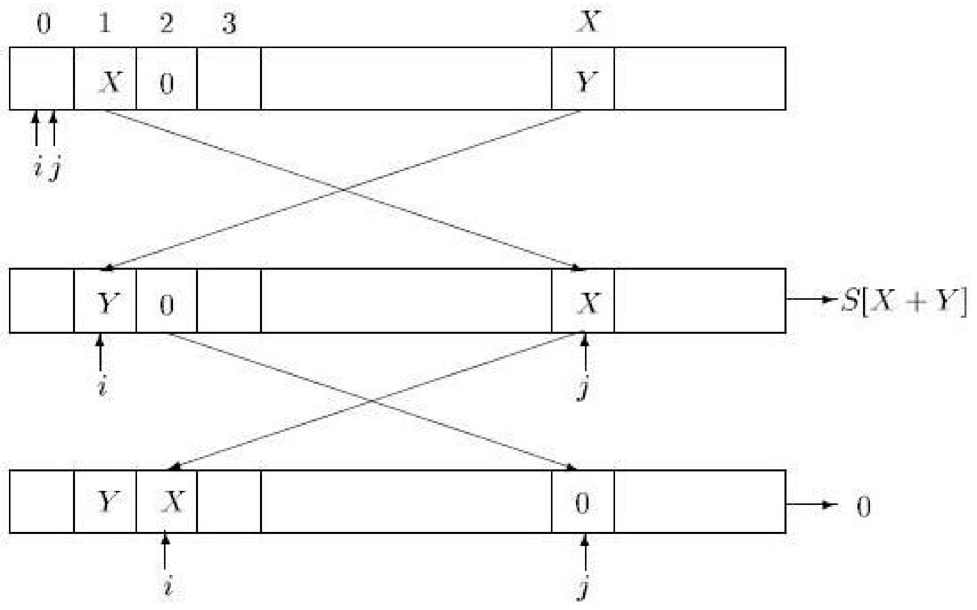

Figure 2.1: The first two rounds of RC4 when $S_0[2] = 0$ and $S_0[1] \neq 2$

Proof: *__When $S_0[1] \neq 2$__* :

Round 0: $i_0 = 0 = j_0$.
Denote $S_0[1]$ by X.

Round 1: $i_1 = 1$

$j_1 = j_0 + S_0[i_1] = 0 + S_0[1] = X$

$S_0[1] \leftrightarrow S_0[X]$

$\Rightarrow S_1[1] = S_0[X] = Y$, say.

$S_1[X] = S_0[1] = X$

$z_1 = S_1[S_1[1] + S_1[X]] = S_1[Y + X]$

Round 2: $i_2 = 2$

$j_2 = j_1 + S_1[i_2] = X + S_1[2] = X$ (since $S_1[2] = S_0[2] = 0$).

$S_1[2] \leftrightarrow S_1[X]$

$\Rightarrow S_2[2] = S_1[X] = X$

$S_2[X] = S_1[2] = 0$

$z_2 = S_2[S_2[2] + S_2[X]] = S_2[X + 0] = 0$

__When $S_0[1]=2$__ :

Round 1: $i_1 = 1$

$j_1 = j_0 + S_0[i_1] = 0 + S_0[1] = 2$
$S_0[1] \leftrightarrow S_0[2]$
$\Rightarrow S_1[1] = S_0[2] = 0$
$S_1[2] = S_0[1] = 2$
$z_1 = S_1[S_1[1] + S_1[2]] = S_1[0 + 2] = 0$

Round 2: $i_2 = 2$
$j_2 = j_1 + S_1[i_2] = 2 + S_1[2] = 2 + 2 = 4$
$S_1[2] \leftrightarrow S_1[4]$
$\Rightarrow S_2[2] = S_1[4] = U \neq 0 =$, say $(U \neq 0 = S_1[1])$
$S_2[4] = S_1[2] = 2$
$z_2 = S_2[S_2[2] + S_2[4]] = \underline{S}_2[U + 2] = 0$, only when $U = 255$
since then $z_2 = S_2[1] = 0$.
Thus $z_2 = 0$ with probability $1/N$. □

Proof(of Theorem 2.1.1):

$P[z_2 = 0]$
$= P[z_2 = 0 \mid S_0[2] = 0] . P[S_0[2] = 0]$
$+ P[z_2 = 0 \mid S_0[2] \neq 0] . P[S_0[2] \neq 0]$
$\approx 1 .1/N + 1/N .(1 - 1/N)$
$= 1/N. (1 + 1 - 1/N)$
$\approx 2/N$
which is twice its expected probability. □

An Interesting Observation Based On This Bias

By applying Bayes rule to the above result, we get

$P[S_0[2] = 0 \mid z_2 = 0]$
$= P[S_0[2] = 0] . P[z_2 = 0 \mid S_0[2] = 0] / P[z_2 = 0]$
$\approx ((1/N). 1) / (2/N)$
$= 1/2$

Consequently, whenever the second output byte is 0 we can extract an entry of S with probability 1/2, which significantly exceeds the trivial probability of $1/N$. This fact can be used to accelerate most of the known attacks by a factor of $N/2$, but this improvement does not suffice to mount a practical attack on $RC4_{n>5}$.

2.1.2 The Biased First and Second Output of RC4

A similar phenomenon appears in the distribution of the first pair of values, where the output pair ($z_1 = 0$, $z_2 = 0$) has probability that is three times the expected $1/N^2$.This result is described in the form of the following theorem:

Theorem 2.1.3 *Assume that the initial permutation S is randomly chosen from the set of all the possible permutations of S_N. Then the probability that both the first and second output word of RC4 is 0 is approximately $3/N^2$.*

Proof: The probability mass is distributed as per the following three cases:

Claim2.1.4 *When $S_0[2] \neq 0$, $P(z_1 = 0, z_2 = 0) = 1/N^2$.*

Proof:

Round 0: $i_0 = 0 = j_0$.

Let $S_0[1] = X$, $S_0[2] = Y$

Round 1: $i_1 = 1$

$j_1 = j_0 + S_0[i_1] = 0 + S_0[1] = X$

$S_0[1] \leftrightarrow S_0[X]$

$\Rightarrow S_1[1] = S_0[X] = Y$, say

$S_1[X] = S_0[1] = X$

$z_1 = S_1[S_1[1] + S_1[X]] = S_1[Y + X]$ ($= 0$ with probability $1/N$)

Round 2: $i_2 = 2$

$j_2 = j_1 + S_1[i_2] = X + S_1[2] = X + Y$ (since $S_1[2] = S_0[2] = Y$)

$S_1[2] \leftrightarrow S_1[X + Y]$

$\Rightarrow S_2[2] = S_1[X + Y] = U$, say

$S_2[X] = S_1[2] = Y$

$z_2 = S_2[S_2[2] + S_2[X]] = S_2[U + Y]$ ($= 0$ with probability $1/N$)

Thus $P(z_1 = 0, z_2 = 0) = 1/N \,.\, 1/N = 1/N^2$. □

Claim 2.1.5 *When $S_0[2] = 0$, $S_0[1] \neq 1$, $P(z_1=0, z_2=0) = 1/N$.*

Proof:

Round 0: $i_0 = 0 = j_0$.

Let $S_0[1] = X$

Round 1: $i_1 = 1$

$j_1 = j_0 + S_0[i_1] = 0 + S_0[1] = X$

$S_0[1] \leftrightarrow S_0[X]$

$\Rightarrow S_1[1] = S_0[\mathrm{X}] = Y$, say

$S_1[X] = S_0[1] = X$

$z_1 = S_1[S_1[1] + S_1[X]] = S_1[Y + X]$ (= 0 with probability $1/N$)

Round 2: $i_2 = 2$

$j_2 = j_1 + S_1[i_2] = X + S_1[2] = X$ (since $S_1[2] = S_0[2] = 0$)

$S_1[2] \leftrightarrow S_1[X]$

$\Rightarrow S_2[2] = S_1[\mathrm{X}] = X$

$S_2[X] = S_1[2] = 0$

$z_2 = S_2[S_2[2] + S_2[X]] = S_2[X + 0] = 0$

Thus $\mathrm{P}(z_1{=}0, z_2{=}0) = 1/N \,.\, 1 = 1/N$. □

Case 3: $\mathrm{S}_0[2] = 0, \mathrm{S}_0[1] = 1$

Claim 2.1.6 *$S_0[2] = 0$, $S_0[1] = 1$, $P(z_1{=}0, z_2{=}0) = 1$.*

Proof:

Round 0: $i_0 = 0 = j_0$.

Round 1: $i_1 = 1$

$j_1 = j_0 + S_0[\mathrm{i}_1] = 0 + S_0[1] = 1$

$S_0[1] \leftrightarrow S_0[1]$

$\Rightarrow S_1[1] = S_0[1] = 1$

$z_1 = S_1[S_1[1] + S_1[1]] = S_1[1 + 1] = 0$ (since $S_1[2] = S_0[2] = 0$)

Round 2: $i_2 = 2$

$j_2 = j_1 + S_1[\mathrm{i}_2] = 1 + S_1[2] = 1$ (since $S_1[2] = 0$)

$S_1[2] \leftrightarrow S_1[1]$

$\Rightarrow S_2[1] = S_1[2] = 0$

$S_2[2] = S_1[1] = 1$

$z_2 = S_2[S_2[1] + S_2[2]] = S_2[0 + 1] = 0$

Thus $\mathrm{P}(z_1 = 0, z_2 = 0) = 1$ □

Proof (of Theorem 2):

$\mathrm{P}[z_1 = 0, z_2 = 0]$

$= \mathrm{P}\,[z_1{=}0, z_2{=}0 \mid S_0[2] \neq 0] \,.\, \mathrm{P}\,[S_0[2] \neq 0]$

$+ \mathrm{P}\,[z_1{=}0, z_2{=}0 \mid S_0[2] = 0, S_0[1] \neq 1] \,.\, \mathrm{P}\,[S_0[2] = 0, S_0[1] \neq 1]$

$+ \text{P}[z_1=0, z_2=0 \mid S_0[2] = 0, S_0[1] = 1] \,.\, \text{P}[S_0[2] = 0, S_0[1] = 1]$
$\approx 1/N^2.(1 - 1/N) + 1/N\,.1/N\,.\,(1 - 1/N) + 1.\,1/N.\,1/N$
$= 1/N^2.(1 - 1/N + 1 - 1/N + 1)$
$= 1/N^2\,.(3 - 2/N)$
$\approx 3/N^2$
which is thrice its expected probability. □

An Interesting Observation Based On This Bias

By applying Bayes rule to the above result, we get

$\text{P}[S_0[2] = 0 \mid z_1 = 0, z_2 = 0]$
$= \text{P}[S_0[2] = 0] \,.\, \text{P}[z_1 = 0, z_2 = 0 \mid S_0[2] = 0] / \text{P}[z_1 = 0, z_2 = 0]$
$= ((1/N).\,(1/N + 1/N)) / (3/N^2)$
$= 2/3$

Consequently, whenever the first and second output byte is 0 we can extract two entries of *S* with probability 2/3, which significantly exceeds the trivial probability of $1/N^2$.

2.2 Cryptanalytic Applications

The strong bias described so far has several practical cryptanalytic applications.

2.2.1 Distinguishing RC4 from Random Sources

The following observation can be used to construct a strong distinguisher for RC4 which requires only $O(N)$ output words.

Theorem 2.2.1 *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.*

Proof: Let X_e, Y_e be the random variables specifying the number of occurrences of e in t samples. Then X_e and Y_e have binomial distributions with parameters (t, p) and $(t, p(1 + q))$, and their expectations, variances and standard deviations are:

$$\begin{aligned}
&\mathrm{E}[X_e] = tp,\\
&\mathrm{E}[Y_e] = tp(1 + q)
\end{aligned}$$

$$\begin{aligned}
&\mathrm{V}(X_e) = tp(1 - p) \approx tp\\
&\mathrm{V}(Y_e) = tp(1 + q)(1 - p(1 + q))\\
&\qquad = tp(1 + q - p(1 + q)^2)\\
&\qquad \approx tp(1 + q)
\end{aligned}$$

$$\begin{aligned}
&\mathrm{SD}(X_e) = \sqrt{tp}\\
&\mathrm{SD}(Y_e) = \sqrt{tp(1+q)} \approx \sqrt{tp}
\end{aligned}$$

We'll analyze the size of t that implies a difference of at least one standard deviation between the expectations of the two distributions:

$$\begin{aligned}
&\mathrm{E}[X_e] - \mathrm{E}[Y_e] \geq \mathrm{SD}(X_e)\\
\Leftrightarrow\ & tp(1 + q) - tp \geq \sqrt{tp}\\
\Leftrightarrow\ & tpq \geq \sqrt{tp}\\
\Leftrightarrow\ & t \geq 1/pq^2
\end{aligned}$$

Consequently, $O(1/pq^2)$ samples (The constant depends on the desired success probability) suffice for the distinguishing. □

Let X be the probability distribution of the second output in uniformly distributed streams, and let Y be the probability distribution of the second output in streams produced by RC4 for randomly chosen keys. The event e denotes an output value of 0, which happens with probability of $1/N$ in X and $2/N$ in Y. By using the previous theorem with $p = 1/N$ and $q = 1$, we can conclude that we need about $(1/pq^2) = N$ outputs to reliably distinguish the two distributions.

2.2.2 A Ciphertext-Only Attack on Broadcast RC4

There are many broadcasting protocols which are used today in a variety of applications. For example, many users send the same email message to multiple recipients (encrypted under different keys), and many groupware applications enable multiple users to synchronize their documents by broadcasting encrypted modification lists to all the other group members. All these applications are vulnerable to this attack.

Theorem 2.2.2 *Let M be a plaintext, and let $C_1, C_2, \ldots\ldots\ldots, C_k$ be the RC4 encryptions of M under k uniformly distributed keys. Then if $k = \Omega(N)$, the second byte of M can be reliably extracted from $C_1, C_2, \ldots\ldots\ldots, C_k$.*

Proof: Let K_i be the i^{th} key and the corresponding RC4 output stream is Z_i, where $i = 1, \ldots\ldots, k$. Then

$$C_i[2] = M[2] \oplus Z_i[2]$$

Since, according to theorem 2.1.1, $Z_i[2] = 0$ with probability $2/N$ with the restriction that $k = \Omega(N)$, therefore

$$C_i[2] = M[2] \text{ with probability } 2/N.$$

Thus, a fraction of $2/N$ of the second ciphertext bytes are expected to have the same value as the second plaintext byte, and thus the most frequent character in $C_1[2], \ldots\ldots, C_k[2]$ is likely to be $M[2]$ itself. □

An improvement can be done on the above attack if we use theorem 2.1.3 and the result is as follows

Theorem 2.2.3 *Let M be a plaintext, and let $C_1, C_2, \ldots\ldots\ldots, C_k$ be the RC4 encryptions of M under k uniformly distributed keys. Then if $k = \Omega(N^2)$, the first and second byte of M can be reliably extracted from $C_1, C_2, \ldots\ldots\ldots, C_k$.*

Proof: $C_i[1] = M[1] \oplus Z_i[1]$ and

$$C_i[2] = M[2] \oplus Z_i[2]$$

Since, according to theorem 2.1.3, $Z_i[1] = 0$ and $Z_i[2] = 0$ with probability $3/N^2$ with the restriction that $k = \Omega(N^2)$, therefore

$$C_i[1] = M[1] \text{ and } C_i[2] = M[2] \text{ with probability } 3/N^2.$$

Thus, a fraction of $3/N^2$ of the second ciphertext bytes are expected to have the same value as the second plaintext byte, and thus the most frequent character in $C_1[1]$,……,$C_k[1]$ is likely to be $M[1]$ itself and that in $C_1[2]$,……,$C_k[2]$ is likely to be $M[2]$ itself.

Chapter 3

Weaknesses in the Key Scheduling Algorithm of RC4

Here we present a special kind of weakness, called invariance weakness in the key scheduling algorithm of RC4. We identify a large number of weak keys, in which knowledge of a small number of key bits suffices to determine many state and output bits with non-negligible probability.

3.1 The Invariance Weakness

3.1.1 The Weakness in KSA*

We prove here the invariance weakness only for a simplified variant of the KSA, which we denote as KSA* as described in Figure 1. The only difference between them is that KSA* updates i at the beginning of the loop, whereas KSA updates i at the end of the loop. After formulating and proving the existence of this weakness in KSA*, we describe the modifications required to apply this analysis to the real KSA.

```
KSA(K)[a]
    For $i = 0 \ldots N-1$
        $S[i] = i$
    $i = 0$
    $j = 0$
    Repeat $N$ times
        $j = j + S[i] + K[i \bmod \ell]$
        $Swap(S[i], S[j])$
        $i = i + 1$
```

```
KSA*(K)
    For $i = 0 \ldots N-1$
        $S[i] = i$
    $i = 0$
    $j = 0$
    Repeat $N$ times
        $i = i + 1$
        $j = j + S[i] + K[i \bmod \ell]$
        $Swap(S[i], S[j])$
```

[a] KSA is rewritten in a way which clarifies the relation to KSA*

Figure 3.1. KSA vs. KSA*

Definition 3.1.1 Let S be a permutation of S_N, t be an index in S and b be some integer. Then if $S[t] \equiv_{\text{mod } b} t$, the permutation S is said to *b-conserve the index t*. Otherwise, the permutation S is said to *b-unconserve* the index t.

Denote the permutation S and the indices i and j after round t of *KSA** as S_t, i_t and j_t respectively. Denote the number of indices that a permutation b-conserves as $I_b(S)$. For the sake of simplicity, we often write i_t instead of $I_b(S_t)$.

Definition 3.1.2 A permutation S of S_N, is *b-conserving* if $I_b(S) = N$, and is *almost b-conserving* if $I_b(S) \geq N - 2$.

Definition 3.1.3 Let b,l be integers, and let K be an ℓ words key. Then K is called a *b-exact key* if for any index t $K[t \bmod \ell] \equiv_{\text{mod } b} (1 - t)$. In case $K[0] = 1$ and $MSB(K[1]) = 1$, K is called a *special b-exact key*.

Proposition 3.1.1 For a b-exact key, it is necessary (but not sufficient) that $b \mid \ell$.

Proof: $K[i] \equiv_{\text{mod } b} (1 - i)$

$$K[i + \ell] \equiv_{\text{mod } b} (1 - i)$$
$$\equiv_{\text{mod } b} (1 - i - \ell)$$
$$\Rightarrow K[i+\ell] = (1 - i) + \lambda b$$
$$= (1 - i - \ell) + \mu b$$

=> $\ell = (\mu - \lambda)b$
=> $b \mid \ell$.

Theorem 3.1.2 *Let $q \leq n$ and ℓ be integers and $b := 2^q$. Suppose that $b \mid \ell$ and let K be a b-exact key of ℓ words. Then the permutation $S = KSA^*(K)$ is b-conserving.*

Before getting to the proof itself, we will prove an auxiliary lemma.

Lemma 3.1.3 *If $i_{t+1} \equiv j_{t+1} \pmod b$, then $i_{t+1} = i_t$, i.e. $I_b(S_{t+1}) = I_b(S_t)$.*

Proof: The only operation that might affect S (and maybe I) is the swapping operation. However, when i_{t+1} and j_{t+1} are equivalent (mod b), S_{t+1} b-conserves i_{t+1} (j_{t+1}) if and only if S_t b-conserved j_t (i_t). Thus the number of indices S_t b-conserves remains the same.

$S_t(i_{t+1})$, and $S_t(j_{t+1})$ are swapped to get $S_{t+1}(j_{t+1})$and $S_{t+1}(i_{t+1})$ respectively. Thus $S_{t+1}(i_{t+1}) = S_t(j_{t+1})$. Now the following two cases may arise:

Case1: Let S_t b-conserves index j_{t+1}.
=> $S_t(j_{t+1}) \equiv_{\text{mod } b} j_{t+1}$
$\equiv_{\text{mod } b} i_{t+1}$ (since $j_{t+1} \equiv_{\text{mod b}} i_{t+1}$)
=> $S_{t+1}(i_{t+1}) \equiv_{\text{mod } b} \mathrm{i}_{t+1}$
=> S_{t+1} b-conserves index i_{t+1}.

Case2: Let S_t does not b-conserve index j_{t+1}.
=> $S_t(j_{t+1}) \not\equiv_{\text{mod } b} j_{t+1}$
=> $S_t(j_{t+1}) \not\equiv_{\text{mod } b} i_{t+1}$ (since $j_{t+1} \equiv_{\text{mod } b} i_{t+1}$)
=> $S_{t+1}(i_{t+1}) \not\equiv_{\text{mod } b} i_{t+1}$
=> S_{t+1} does not b-conserve index i_{t+1}. □

Proof(of Theorem 3.1.2): We will prove by induction on t that for any $1 \leq t \leq N$, it turns out that $I_b(S_t) = N$ and $i_t \equiv j_t \pmod b$. This in particular implies that $I_N = N$, which makes the output permutation b-conserving.

For $t = 0$ (before the first round), the claim is trivial because $i_0 = j_0 = 0$ and S_0 is the identity permutation which is b-conserving for every b.
Suppose that $j_t \equiv i_t$ and S_t is b-conserving. Then
$i_{t+1} = i_t + 1$ and

$$\begin{aligned} j_{t+1} &= j_t + S_t[i_{t+1}] + K[i_{t+1} \bmod \ell] \\ &\equiv_{\bmod b} i_t + i_{t+1} + (1 - i_{t+1}) = i_t + 1 = i_{t+1} \end{aligned}$$

Thus, $i_{t+1} \equiv j_{t+1}$ (mod b) and by applying Lemma 1 we get $i_{t+1} = i_t = N$ and therefore S_{t+1} is b-conserving. □

KSA* thus transforms special patterns in the key into corresponding patterns in the initial permutation.

3.1.2. The Weakness in KSA

The small difference between KSA* and KSA is essential in that KSA, applied to a b-exact key, does not preserve the equivalence (mod b) of i and j even after the first round. Analyzing its execution on a b-exact key gives

$$j_1 = j_0 + S_0[i_1] + K[i_1] = 0 + S_0[0] + K[0] = K[0] \equiv_{\bmod b} 1 \not\equiv_{\bmod b} 0 = i_1$$

and thus the structure described in Section 3.1.1 cannot be preserved by the cyclic use of the words of K. However, the invariance weakness can be adjusted to the real KSA, and the proper modifications are formulated in the following theorem:

Theorem 3.1.4 *Let $q \leq n$ and ℓ be integers and $b := 2^q$. Suppose that $b \mid \ell$ and let K be a special b-exact key of ℓ words. Then*

P[KSA(K) is almost b-conserving] $\geq 2/5$

when the probability is over the rest of the key bits.

Extensive experimentation indicates that this bound is not tight, and the probability is actually very close to one half.

First we prove the following lemma that indicates special properties of RC4 round operation (for the KSA, as well as the PRGA).

Lemma 3.1.5 *Let i_r and j_r be the indices of round r of PRGA (or KSA). Let X be the value pointed to by j in this round before the swap (i.e., $S_{r-1}[j_r] = X$). Then X will not be involved in determining j during rounds $r + 1,\ldots,r + N-1$.*

Proof. We can consider the permutation S as a queue of elements used to update j. Assuming a random behavior of the entries of S, this queue has the following properties:

Random Entering A new value that enters the queue is entered into a random position. When the value X is pointed to by i (before the swap), it is used to forward j and afterwards it enters the permutation in the pseudo-random position j (relative position $j + 1 - i$).

Turn Loss On every round, a randomly chosen element in the queue loses its turn and is thrown to the end of the line. When the value X is pointed to by j (before the swap), it is swapped to position i, which is the worst relative position ($N - 1$). The choice of this deprived value is pseudo-random.

No Overpass The k^{th} element in the queue must wait at least k rounds for its turn. The only transfers in the queue are of the first two types, and consequently no element can move forward.

The correctness of the lemma stems from these properties of RC4 round operation. The value pointed to by j (before the swap) has a Turn Loss and must wait at least N rounds before being used as $S[i]$ (No Overpass). □

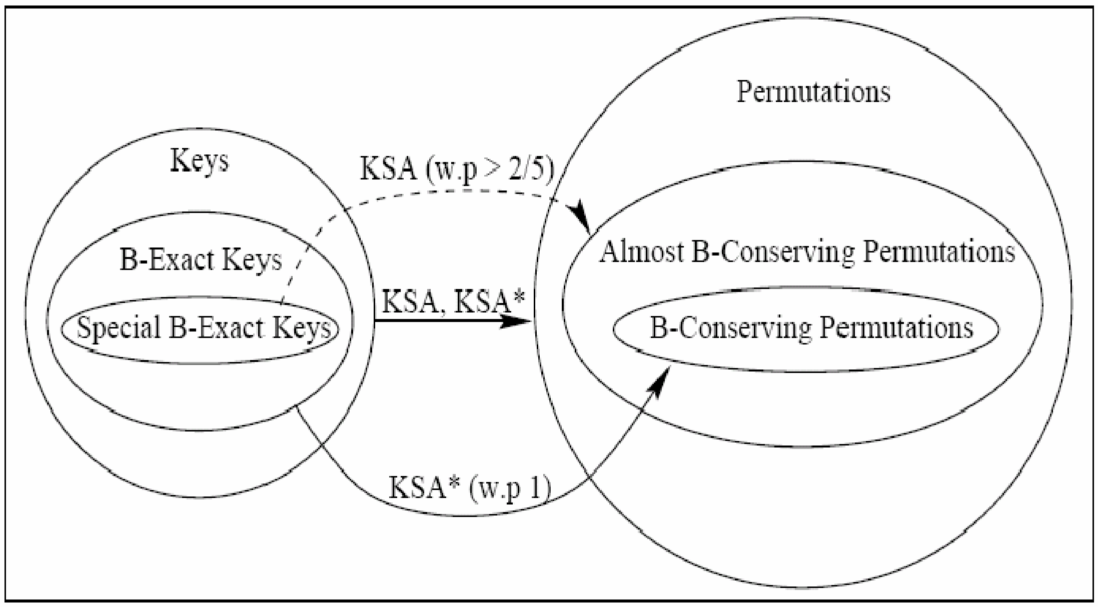

Figure 3.2 Schematic representation of Theorem 3.1.2 and Theorem 3.1.4

Recall that the swap operation of the first round caused the entries i_0 and j_0 to be unconserved, and ruined the equivalence of i and j. However, the additional constraints on the key ensure that in the second round j becomes equivalent to i, and that the unconserved entries will not affect the preservation of the structure during the rest of the KSA. The following lemma summarizes this scenario.

Lemma 3.1.6 *If K is a special 2-exact key for which K[1] = N-2, then the indices i and j of rounds 1,...,N-1 are equivalent (mod 2).*

Proof: $K[0] = 1$, which causes the corrupted indices after round 0 to be $i_0 = 0$ and $j_0 = j_{-1} + S_{-1}[0] + K[0] = 0 + 0 + 1 = 1$. The discrepancy of $S_1[1]$ is used to fix the non-equivalence of i and j during round 1:

$$i_1 = 1$$
$$j_1 = j_0 + S_0[1] + K[1] = 1 + 0 + (N-2) = N-1 \equiv_{\text{mod } 2} i_1$$

Consequently, $i_1 \equiv j_1$, $I_1 = N-2$ and the unconserved indices in S_1 are 0 and $N-1$. Lemma 3.1.5 ensures that these values (that are almost the last ones in the queue at this point) are not involved in determining j during rounds $2,\dots\dots\dots,N-2$, a fact that ensures that only conserved indices are involved in determining $j_2,\dots\dots\dots,j_{N-2}$. This property, along with the exactness of K, preserves the equivalence of j and i (just as in the proof of Theorem 3.1.2) at least until after round $N-2$. □

Lemma 3.1.7 *Let K be a special 2-exact key of an even size. Suppose that K[1] = N-2 (this constraint is compatible with the requirements of special 2-exact keys). Then the outcome of the KSA applied to K is an almost 2-conserving permutation, regardless of the other bits of the key.*

Proof: $I_1 = N-2$. Thus according to lemma 3.1.3 we get $I_{N-2} = I_1 = N-2$.

We analyze two possible scenarios for the last round (round N - 1). If S_{N-1} conserves the index N - 1, j is updated appropriately and we can reuse Lemma 3.1.4 to conclude that $I_{N-1} = I_{N-2}$. Otherwise, the index N - 1 is unconserved by S_{N-2} causing j to be updated inappropriately, i.e., $i_{N-1} \not\equiv j_{N-1}$. However, N - 1 is unconserved, and thus, swapping it with the non-equivalent index j_{N-1}, causes the index j_{N-1} (independently of its apriori status) to become conserved (we use here the fact that $b = 2$, which implies that if $a \not\equiv c$ and $c \not\equiv d$ then $a \not\equiv d$). Thus, at most one of the indices i_{N-1}, j_{N-1} was conserved by S before round N - 1 and at least one of them is conserved after round N - 1. Consequently, the number of conserved indices cannot decrease and $i_{N-1} \geq i_{N-2} = N - 2$, implying that S_N is almost 2-conserving. □

Lemma 3.1.8 *Let K be a special 2-exact key and let S = KSA(K). Then*

$$P[KSA(K) \text{ is almost 2-conserving}] \geq 2/5$$

Proof: Recall that the conditions of this lemma include predetermining the whole $K[0]$, but only the *MSB* of $K[1]$. The eliminated condition (with respect to Lemma 3.1.6) is that $K[1] = N - 2$, which sent the corrupted entry to the last position, making it unlikely that it will affect the other entries. Without this condition, the unconserved entries might interfere with the updates of j, ruin its equivalence to i, and generate more unconserved indices. However, with relatively high probability, this problematic scenario will be prevented even under these weakened conditions. As in the previous case, the corrupted entry in $i_0 = 0$ is promised not to be touched by i during the remaining $N - 2$ rounds. The second corrupted entry (j_1) might be used to update j in round j_1, which would ruin the equivalence of the indices. However, if this location is pointed to by j before round j_1, the discrepancy in moved to an entry which i will not visit (it is possible that this entry will be pointed to by j on some intermediate round r, and the discrepancy will move from j_1 to $i_r = r$, but the entry r will be untouchable by i, and thus the discrepancy can move only between entries which i will never visit), and will not interfere with the updates of j. Notice that $j_1 = 1 + K[1] > N/2$ (recall that $MSB(K[1]) = 1$), and thus j has many opportunities (at least $N/2$) to visit position j_1 before i does. The probability of $N/2$ pseudo random j's to reach some specific value, is well approximated by $1 - 1/\sqrt{e} \approx 2/5$. This gives a lower bound on the probability that $KSA(K)$ is almost 2-conserving. □

Finally, we derive the proof of Theorem 3.1.4 from the proof of Lemma 3.1.8. The equivalence of the indices after the second round (round 1) is independent of b, and the only part of the proof that might change is the probability of the unconserved entries to corrupt this equivalence. However, the corrupted entries are still 0 and $j_2 > N/2$ and thus this probability can be still bounded from below by 2/5. □

KSA thus transforms special patterns in the key into corresponding patterns in the initial permutation. For RC4$_n$ that uses a key of $\ell = 2^p. m$ words (of n bits each), we found that for every $q \leq p$, there exists an assignment of $n + 1 + q(\ell - 1)$ bits (the first word, q LSBs of each of the other $\ell - 1$ words, and the MSB of the second word) of K that determines $\Theta(qN)$ bits of S_0 with a significant probability of one half. For the commonly used RC4$_8$ with a key of 6 bytes, 14 bits of K determine 238 bits of S_0, 19 bits of K determine 472 bits of S_0, etc. This correlation implies that the KSA of RC4 does not mix the bits of the key equally between the bits of the permutation, and this phenomenon induces a weakness of the KSA.

3.2 Key-Output Correlation

Here we will analyze the propagation of the weak key patterns into the generated outputs. First we prove Theorem 3.2.1 which deals with the highly biased behavior of a weakened variant of the PRGA, applied to a b-conserving permutation. Next, we will argue that the prefix of the output of the original PRGA is highly correlated to the prefix of the swapless variant (on the same initial permutation), which implies the existence of biases in the PRGA distribution for these weak keys.

3.2.1 Correlation for PRGA*

Definition 3.2.1 Let PRGA* be a weakened variant of PRGA with no swap operations. RC4* be a weakened variant of RC4 with that uses PRGA*.

Definition 3.2.2 Let $q \in N$, $b := 2^q$. Let $\{x_r\}_{r=1 \text{ to } \infty}$ be the stream of q-bit words, produced by applying the q-bit PRGA* to the identity permutation in S_b. Then the stream $\{x_r\}$ is called a b-pattern, and every stream $\{X_r\}_{r=1 \text{ to } h}$ of n bit words ($n \geq q$) that satisfies

$$\forall r \leq h,\ X_r \equiv x_r \pmod{b}$$

is called a b-patterned stream.

Claim 3.2.1 *Any stream produced by PRGA* from a b-conserving permutation is b-patterned.*

Proof: Let $\{X_r\}_{r=1 \text{ to } \infty}$ be this stream. Denote the state components sequences induced by this stream by $\{S_r\}_{r=1 \text{ to } \infty}$, $\{I_r\}_{r=1 \text{ to } \infty}$ and $\{J_r\}_{r=1 \text{ to } \infty}$. Denote the same sequences induced by the q-bit stream $\{x_r\}_{r=1 \text{ to } \infty}$ by $\{s_r\}$, $\{i_r\}$, $\{j_r\}$. We first prove by induction on r that $\forall r\ I_r \equiv i_r \pmod{b}$ and $J_r \equiv j_r \pmod{b}$. We use the fact that there are no swap operations and thus the permutations S and s do not change and remain b-conserving throughout the generation process.

Base Case For $r = 0$ (before the first round), $I_0 = i_0 = J_0 = j_0 = 0$.

Inductive Step Suppose that $I_{r-1} \equiv i_{r-1}$ and $J_{r-1} \equiv j_{r-1}$. Then

$$I_r = I_{r-1} + 1 \equiv i_{r-1} + 1 \equiv i_r$$
$$Jr = J_{r-1} + S_{r-1}[I_r] \equiv j_{r-1} + I_r \equiv j_{r-1} + i_r \equiv j_{r-1} + s[i_r] \equiv j_r$$

Having these equivalences, we can easily derive the equivalence of the streams

$$x_r = s_r[s_r[i_r] + s_r[j_r]] \equiv s_r[i_r] + s_r[j_r] \equiv i_r + j_r$$
$$X_r = S_r[S_r[I_r] + S_r[J_r]] \equiv S_r[I_r] + S_r[J_r] \equiv I_r + J_r = i_r + j_r = x_r \qquad \square$$

Analysis of this q-bit stream shows that it is periodic with period $2b$.

Theorem 3.2.2 *Let $q \leq n$, $b := 2^q$ and S_0 be a b-conserving permutation. Let $\{X_t\}_{t=1 \text{ to } \infty}$ be the output sequence generated by applying PRGA* to S_0, and $x_t := X_t \bmod b$. Then the sequence $\{x_t\}_{t=1 \text{ to } \infty}$ is periodic with period 2b.*

Proof: Since there is no swap operation, the permutation does not change and remains b-conserving throughout the generation process. Notice that all the values of S_0 are known (mod b), as well as the initial indices $i_0 = j_0 = 0 \equiv 0$ (mod b), and thus the round operation (and the output values) can be simulated (mod b), independently of S_0.

$$i_t = t$$
$$j_t = j_{t-1} + S[i_t] = j_{t-1} + S[t] \equiv_{\text{mod } b} j_{t-1} + t$$

Thus we have

$$\begin{aligned} j_t &\equiv_{\text{mod } b} j_{t-1} + t \\ j_{t-1} &\equiv_{\text{mod } b} j_{t-2} + t-1 \\ &\cdots\cdots\cdots\cdots \\ j_1 &\equiv_{\text{mod } b} j_0 + 1 \end{aligned}$$

By adding the above equations we have

$$j_t \equiv_{\text{mod } b} j_0 + \sum_{i=1}^{t} i = t\,(t+1)\,/\,2$$

Now $x_t = S[S[i_t] + S[j_t]]$
$= S[\equiv_{\text{mod } b}\ i_t + j_t]$
$\equiv_{\text{mod } b}\ i_t + j_t$
$\equiv_{\text{mod } b}\ t + t\,(t+1)\,/\,2$
$= t\,(t+3)\,/\,2$

Thus $x_{2b+1} \equiv_{\text{mod } b} (2b+1)(2b+4)/2$
$= (2b+1)(b+2)$
$\equiv_{\text{mod } b}\ 2$

Also $x_1 \equiv_{\text{mod } b} 1.(1+3)/2 = 2 \equiv_{\text{mod } b} x_{2b+1}$

Hence proving the periodicity. □

The following figure illustrates the above theorem for $b = 2$.

i	j	$S[i]$	$S[j]$	$S[i] + S[j]$	Out
0	0	0	0	0	/
1	1	1	1	0	0
0	1	0	1	1	1
1	0	1	0	1	1
0	0	0	0	0	0
1	1	1	1	0	0
⋮	⋮	⋮	⋮	⋮	⋮

Table 3.1: Important parameters(mod 2) of the first few rounds of RC4*, applied to a 2-conserving permutation

3.2.2 Correlation for PRGA

After proving this biased behavior of PRGA* on *b*-conserving permutations, we analyze the expression of this phenomenon when using the original PRGA. Recall that at each round of the PRGA *S* changes in at most two locations. Thus we can expect a diminishing correlation between the sequences of permutations produced by PRGA and PRGA* from the same initial permutation.

This correlation fades out when *r* is increased, since as more swaps that are made by the PRGA, more entries are "spoiled" by these swaps. This diminishing correlation is expressed also in the output words, which are completely determined by the correlated permutations. Consequently special exact keys are likely to be transformed by the KSA into almost *b*-conserving permutations, which are likely to be transformed by the PRGA into relatively long *b*-patterned streams. The correlation between special exact keys and patterned stream prefixes is demonstrated in Figure 5.2, where the function $h \rightarrow \mathrm{P}[1 \leq \forall r \leq h\ Z_r \equiv x_r \bmod 2^q]$ (for special 2^q-exact keys) is empirically derived for $n = 8$, $\ell = 16$ and different *q*'s. For example, a special 2-exact key completely determines 20 output bits (the LSBs of the first 20 outputs) with probability $2^{-4.2}$ instead of 2^{-20}, and a special 16-exact key completely determines 40 output bits (4 LSBs from each of the first 10 outputs) with probability $2^{-2.3}$, instead of 2^{-40}.

We have thus demonstrated a strong probabilistic correlation between some bits of the secret key and some bits of the output stream for a large class of weak keys. An important observation about this correlation is its unexpected dependency on n. Notice that the size of the correlated prefix depends on the probability that all the entries that are used to produce the output (three entries per round) are "virgin", that is were not swapped in an earlier round. The probability that the three indices which are used to generate the h^{th} output, were not swapped in the previous $h - 1$ rounds, has negative linear dependence on the probability of a random entry to hit a specific entry, which is proportional to $1/N$. Thus the size of the correlated output depends linearly on N (and exponentially in n). This is somewhat surprising since one would expect that enlarging the array would strengthen the overall security. However, the expression of the invariance weakness is amplified, which counterbalances this expectation. Moreover, assuming a fixed-size key, the fraction of 2^q-exact keys for q > 1 "prefers" large n's. This is due to the fact that the dependency on $\ell = \#key\ bits\ /\ n$ is stronger than the dependency on n (recall that a 2^q-exact key requires fixing $n + 1 + q\ (\ell - 1)$ bits). Consequently using a 128-bit key in $RC4_{8,16}$ is more immune to this weakness than using this key in $RC4_{16,8}$. This phenomenon, where the expression of the invariance weakness is amplified when n is increased, is demonstrated in Figure 3.3.

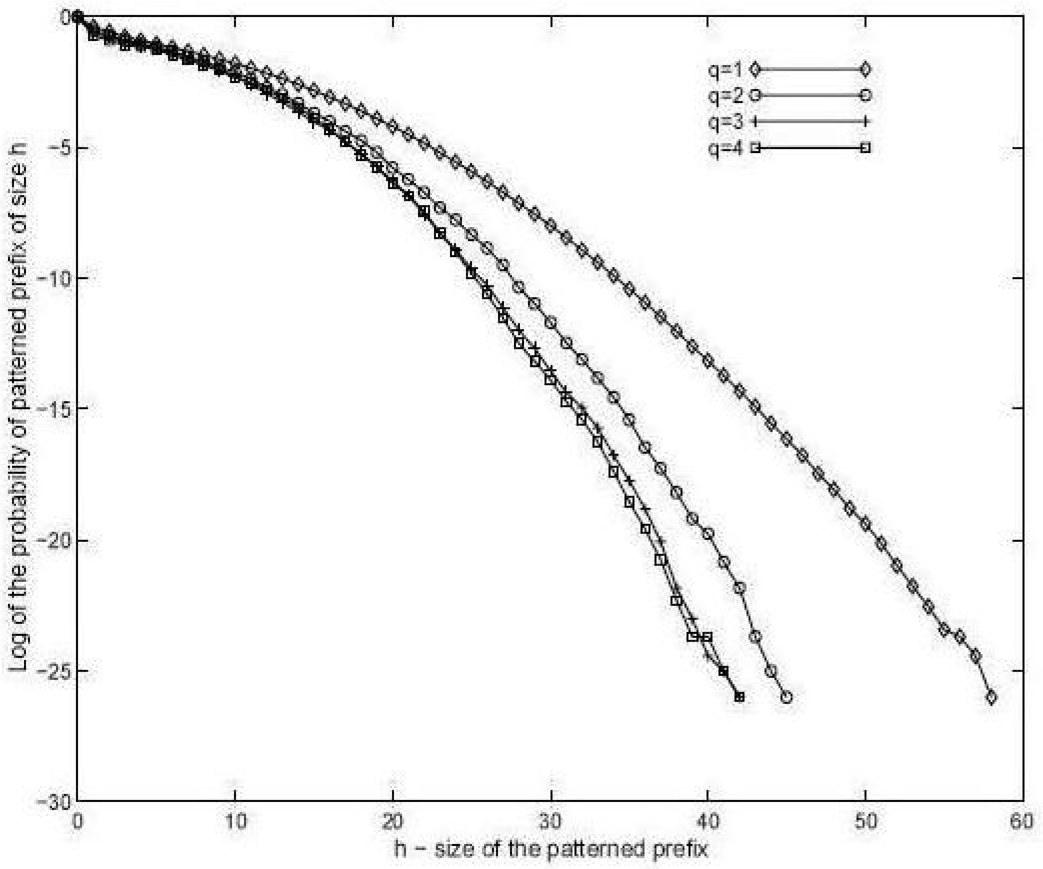

Figure 3.3: This graph demonstrates the probabilities of special 2^q-exact keys of $RC4_{8,16}$ to produce streams with long 2^q-patterned prefixes

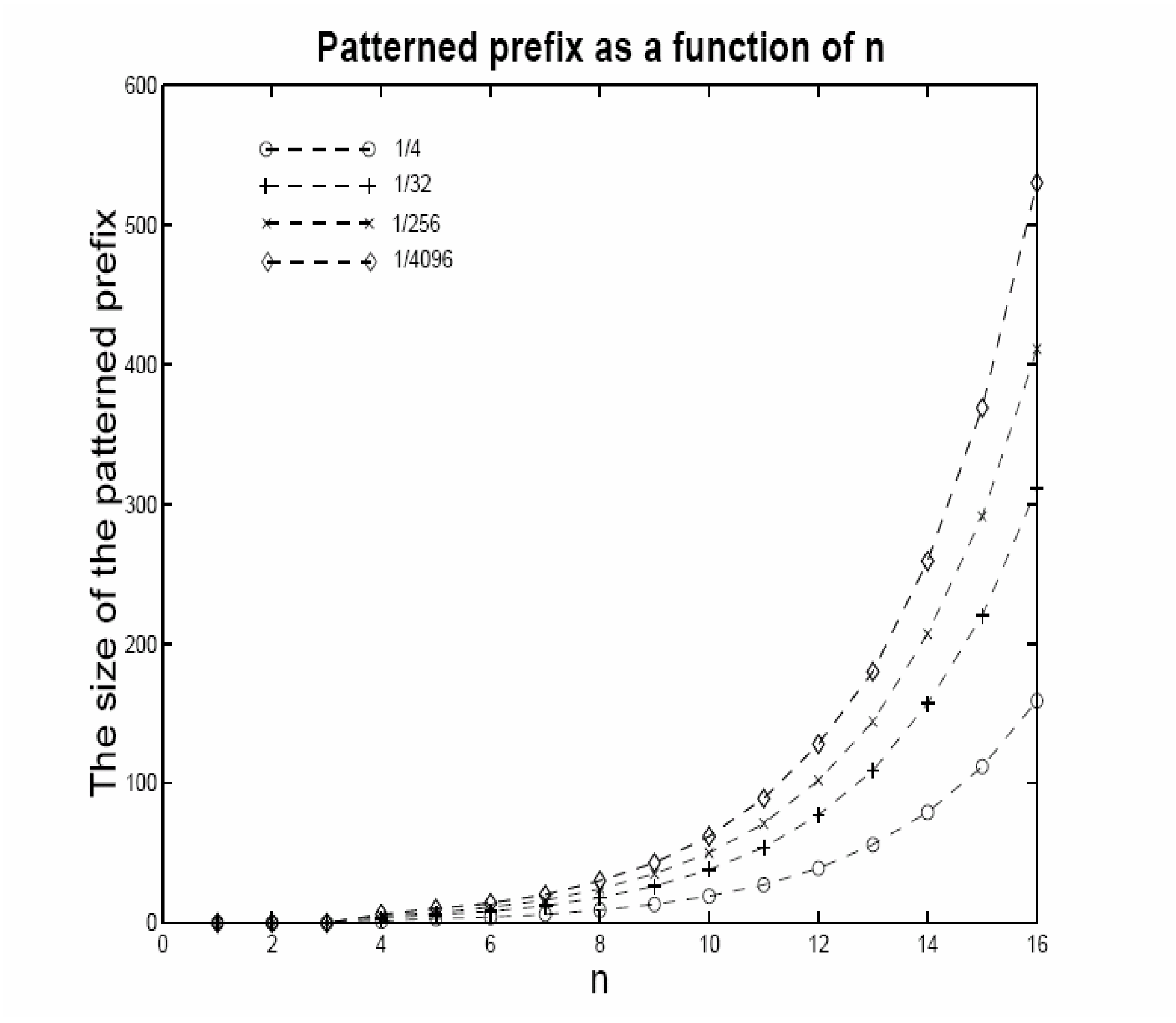

Figure 3.4: This graph demonstrates the output prefix that is 2-patterned with fixed probabilities (1/4, 1/32, 1/256 and 2^{-12}) when $RC4_{n,16}$ is applied to a 2-exact key, as a function of *n*.

3.3 Cryptanalytic Applications of the Invariance Weakness

3.3.1 Distinguishing RC4 Streams from Randomness

In [2] Mantin and Shamir described a significant statistical bias in the second output word of RC4. They used this bias to construct an efficient algorithm which distinguishes between RC4 outputs and truly random sequences by analyzing only one word from $\boldsymbol{O}(N)$ different outputs streams. This is an extremely efficient distinguisher, but it can be easily avoided by discarding the first two words from each output stream. If these two words are discarded, the best known distinguisher requires about 2^{30} output words (see [3]). Our new observation yields a significantly better distinguisher for most of the typical key sizes. The new distinguisher is based on the fact that for a significant fraction of keys, a significant number of initial output words contain an easily recognizable pattern. This bias is flattened when the keys are chosen from a uniform distribution, but it does not completely disappear and can be used to construct an efficient distinguisher even when the first two words of each output sequence are discarded.

Notice that the probability of a special 2^q-exact key to be transformed into a 2^q-conserving permutation does not depend of the key length ℓ (see Theorem 3.1.4). However, the number of predetermined bits is linear in ℓ, and consequently the size of this bias (and thus the number of required outputs) also depends on ℓ. In Table 3.2 we specify the quantity of data required for a reliable distinguisher, for different key sizes. In particular, for 64 bit keys the new distinguisher requires only 2^{21} data instead of the previously best number of 2^{30} output words.

It is important to notice that the specified output patterns extend over several dozen output words, and thus the quality of the distinguisher is almost unaffected by discarding the first few words. For example, discarding the first two words causes the data required for the distinguisher to grow by a factor of between $2^{0.5}$ and 2^2 (depending on ℓ). Another important observation is that the biases in the LSB's distribution can be combined in a natural way with the biased distribution of the LSB 's of English texts into an efficient distinguisher of RC4 streams from randomness in a ciphertext-only attack in which the attacker does not know the actual English plaintext which was encrypted by RC4. This type of distinguishers is discussed in the next section.

ℓ	q	b	k_1 [a]	k_2 [b]	p [c]	P_{RND} [d]	P_{RC4} [e]	Data
4	1	2	12	15	2^{-3}	2^{-15}	$2 \cdot 2^{-15}$	2^{15}
6	1	2	14	18	2^{-4}	2^{-18}	$2 \cdot 2^{-18}$	2^{18}
8	1	2	16	21	2^{-5}	2^{-21}	$2 \cdot 2^{-21}$	2^{21}
10	1	2	18	24	2^{-6}	2^{-24}	$2 \cdot 2^{-24}$	2^{24}
12	1	2	20	27	2^{-7}	2^{-27}	$2 \cdot 2^{-27}$	2^{27}
14	1	2	22	30	2^{-8}	2^{-30}	$2 \cdot 2^{-30}$	2^{30}
16	1	2	24	34	2^{-10}	2^{-34}	$2 \cdot 2^{-34}$	2^{34}

Table 3.2: Data required for a reliable distinguisher, for different key sizes

[a] number of predetermined bits $(q(\ell - 1) + n + 1)$
[b] number of determined output bits
[c] probability of these k_1 key bits to determine these k_2 output bits
[d] $= 2^{-k_2}$
[e] $\approx P_{RND} + 2^{-k_1} p$

3.3.2 Ciphertext-Only Distinguishers based on the Invariance Weakness

The distinguishers we presented in Section 3.3.1, as well as most of the distinguishers mentioned in the literature (for RC4 and other stream ciphers) assume knowledge of the plaintext in order to isolate the XORed keystream.

However, in practice the only information the attacker has is typically some statistical knowledge about the plaintext, e.g., that it contains English text. Combining the non-random behaviors of the plaintext and the keystream is not always possible, and there are cases where XORing biased streams result with a totally random stream (e.g. when one stream is biased in its even positions and the other stream is biased in its odd positions). We prove here that if the plaintexts are English texts, it is easy to construct a ciphertext-only distinguisher from aforesaid biases. The intuition of this construction is that the biases described in Section 3.3.1 are in the distribution of the LSBs, and consequently they can be combined with the non-random distribution of the LSBs of English texts.

There are many major biases in the distribution of the LSBs of English texts, and they can be combined with biases of the keystream words in various ways. In theorem 3.3.1 we estimate the bias caused by XORing two streams which are biased in their LSBs distribution.

Theorem 3.3.1 *Let $\{m_t\}$, $\{z_t\}$ and $\{c_t\}$ be the plaintext, keystream and ciphertext (mod 2) of a stream cipher respectively (we assume independence between the plaintext and the keystream). Suppose that z_t and m_t have positive biases $b_z > 0$, $b_m > 0$ towards the bits v_z; v_m respectively, i.e, $Pr[z_t = v_z] = 0.5 + b_z$ and $Pr[m_t = v_m] = 0.5 + b_m$. Then*

$$Pr[c_t = v_m \oplus v_z] = 0.5 + 2b_m b_z$$

Proof:

$$\begin{aligned}
\Pr[c_t = v_m \oplus v_z] &= \Pr[m_t = v_m, z_t = v_z] + \Pr[m_t = \overline{v_m}, z_t = \overline{v_z}] \\
&= \Pr[m_t = v_m] \cdot \Pr[z_t = v_z] + \Pr[m_t = \overline{v_m}] \cdot \Pr[z_t = \overline{v_z}] \\
&= (\tfrac{1}{2} + b_m) \cdot (\tfrac{1}{2} + b_z) + (\tfrac{1}{2} - b_m) \cdot (\tfrac{1}{2} - b_z) \\
&= \tfrac{1}{2} + 2b_m b_z
\end{aligned}$$

□

Next we exemplify constructions of ciphertext-only distinguishers, by concentrating on specific biases of the LSBs distributions in English texts.

Corollary 3.3.2 *Let C be the ciphertext generated by RC4 from a random key and the ASCII representation of plaintexts, distributed according to the first order statistics of English texts. Let p_ℓ be the probability of a random ℓ-words key to be special 2- exact. Then C can be distinguished from a* *random stream by analyzing the first few words of about* $\frac{800}{p_\ell^2}$ *different RC4 streams.*

Proof: The first order statistics of English texts gives a 55% probability of a character to have LSB 0, and thus $b_m \approx 0.05$. Analyzing the results from section 3.2 we get

$$\begin{aligned} b_z &= \Pr[c_1 = 0] - 0.5 \\ &= \Pr[c_1 = 0 \mid \text{key is special } b\text{-exact}] + \\ &\quad \Pr[c_1 = 0 \mid \text{key is not special } b\text{-exact}] \,.\, (1 - p_\ell) \\ &\quad - 0.5 \\ &\approx 0.75 \cdot p_\ell + 0.5 \cdot (1 - p_\ell) - 0.5 \\ &= 0.25 \cdot p_\ell \end{aligned}$$

Thus, $\Pr[c_1 = 0] = 0.5 + 2 \cdot 0.05 \cdot 0.25 \cdot p_\ell = 0.5 + 0.025 \cdot p_\ell = 0.5 \cdot (1 + 0.05\, p_\ell)$ and the data required for a reliable ciphertext-only distinguisher is (again we use Theorem 2.2.1)

$$O\left(\frac{1}{pq^2}\right) = O\left(\frac{1}{0.5 \cdot (0.05 \cdot p_\ell)^2}\right) = O\left(\frac{800}{p_\ell^2}\right)$$

□

For the typical $RC4_{8,8}$, $p_\ell = 2^{-16}$ and we get a $2^{41.6}$ ciphertext-only distinguisher. A drawback of this approach is that our distinguisher works on prefixes of messages that have distribution different from first order statistics of English. For example, the space character is not likely to occur at the beginning of a message, whereas in first order statistics of English, it is the most probable character. Experiments we made imply that the 4th character of English texts has probability of 60% for having an LSB of 0. Thus we get $b_m = 0.1$, and $b_z = \Pr[z_4 = 0] - 0.5 \approx 0.25 \cdot p_\ell$ (from the graph), which gives $\Pr[c_4 = 0] = 0.5 + 2 \cdot 0.1 \cdot 0.25 \cdot p_\ell = 0.5 \cdot (1 + 0.1 \cdot p_\ell)$. Thus the data required for the distinguisher is $2^{39.6}$ outputs.

It is important to note that we did not use all the statistical information that is available in either the keystream or the plaintext distributions, and consequently this analysis does not represent the best possible attack.

Chapter 4

Conclusion and Scope of Future Works

4.1 Conclusion

In this thesis we have described several flaws in the initialization mechanism of RC4, which are caused by its extreme simplicity. The complicated task of the KSA, which is to extend relatively short random keys into large pseudorandom permutations, is reasonably but only partially fulfilled. The initialization of the pseudo-random index j to 0 seems to be the most problematic operation, and the second byte bias could be avoided by using a more complex initialization of j. Possible methods for initializing j are to use j from the end of the KSA or to give it the value of one of the key words. The invariance weakness is the inherent consequence of the structure of the KSA.

A perfect initialization mechanism is not easy to achieve. We would like to avoid patterns that are independent of the key (like the second byte bias), while on the other hand we do not want any trivial dependency between the key and the first output bits (like the invariance weakness). A common mode of operation to achieve these contradicting goals is to discard a prefix of output bits. These mute rounds usually disconnect the generated stream from the initialization process, and improve the "randomness" of the generated stream. Discarding the first two bytes voids the practical attacks, but retains the invariance weakness. Consequently, it is recommended to discard at least complete sequence of N words.

One can notice that when enlarging RC4 words into 16 bits (which is sometimes recommended for faster encryption of large amount of data), the discarded prefix should also grow in the same way (exponentially). The expression of the invariance weakness spreads over several hundred words in $RC4_{16}$ and eliminating only 256 words is not sufficient when N is larger.

The final and a positive conclusion about the security of RC4 is that using RC4 in this way seems to be secure, even when using a concatenation-based session-key derivation. It is believed that no information leaks about either the key or any part of the encrypted messages.

4.2 Scope of Future Works

Many new observations about the internal state, the output distribution and the correlation between them in RC4 are there in the literature. Several methods are being described to extract partial information about the internal state, and we believe that these methods can be further improved.

Excluding prefix distinguishers, the best known distinguisher for RC4 ([3]) is based on counting the number of occurrences of output pairs in periodic positions in the stream, which is an amazingly simple method. A non-negligible portion of our thesis was dedicated to describing better distinguishers.

The most promising way to constructing a distinguisher is based on the correlations of Jenkins. We know that whenever a value X is pointed to by j before the swap, it will not be pointed to by i (before the swap) for at least N rounds. However, the values that are pointed to by i can be guessed to be $i - z$. Thus knowing $S[j] = X$ can be used to predict that $i - z$ will differ from X during at least N rounds. Furthermore, the value that was used as S[j] has a constant probability (1/e) to be used as S[i] after exactly N rounds, and a doubled probability to be equal to $i - z$ in this round. The lower bound to success probability of guessing S[j] with success probability is $\frac{1}{N}(1 + \frac{c}{N})$ (for a small constant c), but not strict and unfortunately a tighter bound is not yet found.

A promising research direction is to extend the analysis of the KSA in the view of transfer function. Calculating (recursively or explicitly) the pairs transfer function, i.e. the probability that the values in positions a and b to reach positions c and d respectively within r rounds, might lead to new observations. This function is at least as biased as the singles transfer function, and might be more useful. It is possible that the pairs transfer function can be easily derived from the singles transfer function. If this is the case, it will provide new insights into the complete distribution of KSA outputs, which might be surprisingly biased.

Bibliography

[1] Scott R. Fluhrer, Itsik Mantin, and Adi Shamir, Weaknesses in the key scheduling algorithm of RC4, SAC: Annual International Workshop on Selected Areas in Cryptography, SAC'2001, LNCS, 2001.

[2] Itsik Mantin and Adi Shamir, A practical attack on broadcast RC4, FSE: Fast Software Encryption, FSE'2001 (Yokohama, Japan), Springer-Verlag, April 2001.

[3] Scott R. Fluhrer and David A. McGrew, Statistical analysis of the alleged RC4 keystream generator, FSE: Fast Software Encryption, FSE'2000, Springer-Verlag, 2000, pp. 19-30.

[4] Serge Mister and Stafford E. Tavares, Cryptanalysis of RC4-like ciphers, SAC: Selected Areas in Cryptography, SAC'98 (Kingston, Ontario, Canada), LNCS, vol. 1556, Springer-Verlag, August 1999, pp. 131-143.

[5] Jovan Dj. Goli'c, Linear statistical weakness of alleged RC4 key-stream generator, EUROCRYPT: Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, EUROCRYPT'97 (Konstanz, Germany), LNCS, vol. 1233, Springer-Verlag, May 1997, pp. 226-238.

[6] Manuel Blum and Shafi Goldwasser, An efficient probabilistic public-key encryption scheme which hides all partial information, Advances in Cryptology: Proceedings of CRYPTO'84 (Santa Barbara, California, USA), Springer-Verlag, August 1985, pp. 289-302.

[7] Hal Finney, an RC4 cycle that can't happen, September 1994.

[8] Alexander L. Grosul and Dan S. Wallach, a related-key cryptanalysis of RC4, Technical Report TR-00-358, Department of Computer Science, Rice University, October 2000.

[9] Lars R. Knudsen, Willi Meier, Bart Preneel, Vincent Rijmen, and Sven Verdoolaege, Analysis methods for (alleged) RC4, ASIACRYPT: Advances in Cryptology, International Conference on the Theory and Applications of Cryptology and Information Security, ASIACRYPT'98 (Beijing, China), LNCS, vol. 1514, Springer, October 1998, pp. 327-341.

[10] David A. McGrew and Scott R. Fluhrer, The stream cipher LEVIATHAN, https://www.cosic.esat.kuleuven.ac.be/nessie/, October 2000, NESSIE project submission.

[11] Andrew Roos, A class of weak keys in the RC4 stream cipher, sci.crypt posting, September 1995.

[12] Itsik Mantin, Analysis of the stream cipher RC4, November 2001