Report on Security Evaluation of RC4 Stream Cipher

Dr. Miodrag J. Mihaljević
Research Professor at
Mathematical Institute, Serbian Academy of Sciences and Arts
Kneza Mihaila 35, Belgrade, Yugoslavia
E-mail: miodragm@turing.mi.sanu.ac.yu

August 1, 2002

Abstract

Main results of this report are the following: (i) collecting and summarization of the reported results relevant for security evaluation of RC4; (ii) analysis of the techniques relevant for cryptanalysis of RC4 and estimation of theirs practical impacts on RC4 security; (iii) developing of a novel technique relevant for cryptanalysis of RC4; (iv) more insight into RC4 keystream generator via identification of certain critical points for RC4 security; (v) estimation of current security level of RC4; (vi) estimation of security of SSL encryption based on RC4. The features of RC4 algorithm, disclosed in this report, are at least undesirable and potentially dangerous, and accordingly, appropriate modifications are strongly recommendable, as well as further security evaluation of RC4.

Report on Security Evaluation of RC4 Stream Cipher

Content

Part I: Introduction and RC4 Specification

1 Introduction

This report gives a security evaluation of RC4 Stream Cipher according to the following requests from the document no. 14-IPA-505:

- "Investigate the security of RC4 by applying various cryptanalytic techniques for stream ciphers. We also expect the evaluation using new effective attacks for RC4 and the summary of RC4's previous attacks. Additionally, since RC4 is one of major SSL encryption algorithms, estimate the practical security of RC4 when it is used as an SSL encryption algorithm."

In order to fulfil the above requests, this report contains the following:

1. specification of stream ciphers security evaluation methods relevant for RC4;
2. summary of the reported results on RC4 security evaluation;
3. analysis of various cryptanalytic techniques relevant for RC4;
4. proposal of a novel technique relevant for cryptanalysis of RC4;
5. an estimation of practical security of SSL encryption based on RC4.

Executive Summary of the Report Results

This report originates from a collection of all up to now reported results relevant for security evaluation of RC4. These results are used for evaluation of RC4 resistance on various cryptanalytic attacks.

Main results of this report are the following:

- collecting and summarization of the reported results relevant for security evaluation of RC4;
- analysis of the techniques relevant for cryptanalysis of RC4 and estimation of theirs practical impacts on RC4 security;
- developing of a novel technique relevant for cryptanalysis of RC4;
- more insight into RC4 keystream generator via identification of certain critical points for RC4 security;

- estimation of current security level of RC4;
- estimation of security of SSL encryption based on RC4.

The features of RC4 algorithm, disclosed in this report, are at least undesirable and potentially dangerous, and accordingly, appropriate modifications as well as further security evaluation of RC4, are strongly recommendable.

Organization of the Report

The report is organized as follows. It contains of five parts and each part contains one or more sections. Part I (Sections 1 and 2) contains the introductory notes and description of RC4. Part II (Section 3) yields specification of stream ciphers security evaluation methods relevant for RC4. Analysis of various cryptanalytic techniques relevant for RC4 is considered in Part III (Sections 4, 5 and 6). A novel technique relevant for cryptanalysis of RC4 is proposed in Part IV (Section 7). Part V (Sections 8 and 9) contains an estimation of practical security of SSL encryption based on RC4, conclusions and the references.

Section 2 yields an overview of RC4. Section 3 specifies the underlying approach for security evaluation. Previously reported results on RC4 security are summarized in Section 4. In detail discussion of various attacks on RC4 Key Scheduling Algorithm (KSA), and Pseudo-Random Generation Algorithm (PRGA) is given in Sections 5 and 6, respectively. A novel technique for cryptanalysis of RC 4 based on the time-memory-data trade-off is proposed and considered in Section 7. A discussion of SSL security based on RC4 encryption is presented in Section 8. Summary of this report results and related conclusions are given in Section 9.

2 RC4 Overview

The first part of this section yields precise description of RC4 based on the reported results. The second part yields a number of preliminary observations relevant for analysis of RC4.

2.1 Description

RC4 is a keystream generator for a binary additive stream cipher. It uses a variable sized secret key that usually consists of 40 - 256 bits. According to [8], RC4 consists of two main parts:

- Key Scheduling Algorithm (KSA),
- Pseudo-Random Generation Algorithm (PRGA).

KSA performs the initialization function which accepts a key of variable size and uses it to create the initial state of the keystream generator.

PRGA is the core algorithm for encryption and decryption. PRGA generates the keystream, a sequence of bits that are then, during encryption phase, combined with the plaintext with XOR yielding the ciphertext. Decryption consists of re-generating this keystream and XOR'ing it to the ciphertext.

RC4 is actually a class of algorithms parameterize on the size of its block. This parameter, n, is the word size of the algorithm. This is recommended to be $n = 8$, but it can take smaller, as well as greater values. Also, for extra security it is possible to increase this value.

Given the parameter n, the internal state of RC4 consists of a balanced table (permutation) of different binary words of dimension n^2 and two pointer binary words of the same dimension n which, at each time, define the positions of two words in the table to be swapped to produce the table at the next time. The internal memory size is thus practically given as $n^2 + 2n$. One of the pointers is updated by using the table content at the position defined by the other, which is in turn updated in a known way by a counter. Initially, the two pointer words are set to zero and the table content is defined by the secret key in a specified way. At each time, the output of RC4 is a binary word of dimension which is taken from an appropriate position in the table. The output word is then bitwise added to the plaintext word to give the ciphertext word.

The internal state of RC4 consists of a table of size 2^n words and two wordsized counters. The table is known as the S-box, and will be known as S. It always contains a permutation of the possible 2^n values of the n-bit words. The two counters are known as i and j. The Key Schedule Algorithm of RC4 works as follows. It accepts as input the key K, which is ℓ bytes long. It starts with the identity permutation in S and, using the key, continually swapping value to produce a new unknown key-dependent permutation. Since the only action on S is to swap two values, the fact that S contains a permutation is always maintained.

The initialization function is given bellow where all the additions are $mod 2^n$.

```
Initialization: For $i = 0$ to $2^n - 1$
$S[i] = i$
$j = 0$
Scrambling: For $i = 0$ to $2^n - 1$
$j = j + S[i] + K[i \bmod \ell]$
$Swap(S[i], S[j])$
```

RC4 Initialization: the Key Schedule Algorithm.

The RC4 keystream generator works as follows. It works by continually shuffling the permutation stored in S as time goes on, each time picking a different value from the S permutation as output. One round of RC4 outputs an n-bit word as keystream symbol, which can then be XOR'ed with the plaintext to produce the ciphertext.

```
**Initialization:**
$i = 0$
$j = 0$
**Generation Loop:**
$i = i + 1$
$j = j + S[i]$
$Swap(S[i], S[j])$
Output $z = S[S[i] + S[j]]$
```

RC4 Pseudo Random Generation Algorithm

Alternatively, following [35] RC4 can be described as follows.

The internal state of RC4 at time t consists of a table (permutation) $S_t = (s_t(\ell))_{\ell=0}^{2^n-1}$ of 2^n different n-bit words and two pointer n-bit words i_t and j_t. Let the output n-bit word of RC4 at time t be denoted by Z_t. The n-bit words are treated as binary representations of integers. Let initially $i_0 = j_0 = 0$. Then the next-state and output functions of RC4 are for every defined by

$$i_t = i_{t-1} + 1 \tag{1}$$

$$j_t = j_{t-1} + S_{t-1}(i_t) \tag{2}$$

$$S_t(i_t) = S_{t-1}(j_t)$$

$$S_t(j_t) = S_{t-1}(i_t) \tag{3}$$

$$Z_t = S_t(S_t(i_t) + S_t(j_t)) \tag{4}$$

where all the additions are modulo 2^n. It is assumed that all the words except for the swapped ones remain the same (swapping itself is effective only if $i_t \neq j_t$). The output n-bit word sequence is $Z = (Z_t)_{t=1}^{\text{inf}}$.

The initial table S_0 is defined in terms of the key string $K = (K_\ell)_{\ell=0}^{2^n-1}$ by incorporating this string into the RC4 next-state function starting from the table (identity permutation) $(\ell)_{\ell=0}^{2^n-1}$. More precisely, set $j_0 = 0$ and for every $1 \leq t \leq 2^n$, compute $j_t = (j_{t-1} + S_{t-1} + K_{t-1}) mod 2^n$ and then swap $S_{t-1}(t-1)$ with $S_{t-1}(j_t)$. The last produced table represents S_0. The key string K is composed of the secret key, possibly repeated, and of the randomizing key which is sent in the clear for resynchronization purposes.

2.2 Observations on the Design Principles

- S starts as a permutation (the identity permutation) of all the 2^n possibly words. Since the only operation on S is $Swap$, S always remains a permutation. However, this permutation changes with time. This is where the strength of RC4 comes from.

- The internal state is stored in a memory $M = n^2 + 2n$ bits. However, since S is a permutation, this state holds effectively $log_2(2^n) + 2n \approx 1700$ bits of information.

- Knowing the entire M bits of the state at a given time is enough to predict all of the keystream bits in the future. Knowing the entire initial state is enough to predict all of the keystream bits, effectively breaking the cipher.

- The initial state is dependent solely on the key K. Therefore knowing the key is sufficient to break RC4.

- Although the KSA is reversible, it is difficult to determine the key K from the initial state of S. However, this is unnecessary to break the cipher.

- The period of RC4 is difficult to predict, and dependent on the key. However, empirical evidence from [21] suggests that the period is normally very long.

- Also, according to [11], note the following. The known pointer sequence j ensures that every element in the table S affected by swapping at least once at any 2^n successive times and, also, that the next-state function is invertible (one-to-one). Accordingly, the state diagram consists of cycles only, which, can be expected to have average length close to 2^{M-1} and are very unlikely to be short if $n \geq 5$. Of course, since the next-state function of RC4 is not randomly chosen, this remains to be proved, if possible at all.

Part II: Security Evaluation Methods Relevant for RC4

3 Security Evaluation Preliminaries

This section summarizes methodological issues relevant for security evaluation of RC4.

3.1 Security Evaluation Preliminaries I

According to the currently widely used security criteria (see [27] and [4]), the general methodological issues for security analysis include the following.

Resistance to Cryptanalysis. A stream cipher should be resistant at the relevant security level against to possible cryptanalytic attacks. However, when assessing the relevance of a cryptanalytic attack a number of factors should be included such as the overall complexity of the attack (time and space complexity), and volume and type of data required to mount the attack, for example. Any cryptanalytic attack is based on the assumption that complete structure of a stream cipher is known, and that the only one unknown element is the secret key.

Design Philosophy and Transparency. An important consideration when assessing the security of a stream cipher is the design philosophy and transparency of the design. It is easier to have confidence in the assessment of the security if the design is clear and straightforward, and is based on well-understood mathematical and cryptographic principles.

Strength of Modified Primitives. One common technique to assess the strength of a stream cipher is to assess a modified one, obtained by changing or removing a component of the considered stream cipher. Conclusions about the original stream cipher based on assessment of the modified one have to be carefully considered as the influence may or may not be straightforward.

Testing. The purpose of testing is to highlight anomalies in the operation of a stream cipher that my indicate cryptographic weakness and require further investigation.

The techniques for security evaluation of a stream cipher are based on the following:
• consideration of certain characteristics of the underlying structure of a stream cipher, as well as the keystream sequence itself;
• consideration of the resistance against the attacks for predicting a keystream, or either recovering or reducing uncertainty of the secret key or the plaintext; these attacks can be:
- general attacks applicable to a class of stream ciphers;
- specialized attacks developed for a particular stream cipher.

It is customary when analysing stream ciphers to consider known plaintext attacks which essentially means that an attacker, a cryptanalyst, knows a large volume of keystream.

The cryptanalyst's task is then usually classified in one of the following three ways.

(i) Distinguishing Attack. The cryptanalyst gives a method that allows the keystream of certain length to be distinguished from a "random" sequence of the same length.

(ii) Prediction. The cryptanalyst gives a method that allows the prediction of further keystream elements more accurately than guessing.

(iii) Key Recovery. The cryptanalyst gives a method that recovers the secret key from the keystream sequence.

The techniques used to analyse stream ciphers typically use either mathematical and statistical properties or certain approximations. The security evaluation should include the main general characteristics of a stream cipher and the attacks to which it should be resistant as given below. A good stream cipher must pass all these consideration, but it is only a necessary request and final security evaluation result should include other aspects including specific issues related to a particular stream cipher.

Finally note the following.

• It is very important to consider specialized attacks dedicated to a particular keystream generator because there are a number of very illustrative examples where certain keystream generator is fully resistant against all general attacks but is breakable by the dedicated attacks.

• Also, if according to the present analysis, a keystream generator does not show any weakness, it is very important to consider security of the modified versions of a keystream generator because this consideration can indicate a lot about the structural characteristics and critical points for security of the original keystream generator.

3.2 Security Evaluation Preliminaries II: Attack Models

According to the previous discussion, this section summarizes a number of common attack models. These attack models are considered as the good models to study the security of stream ciphers, but note that they do not cover all possible attacks.

For example, one thing to remember is that stream ciphers (and most encryption algorithms in general) do not provide for message integrity. This must be done by some external algorithm, such as a MAC (Message Authentication Code). Note that it is possible to use the lack of message integrity checks to partially determine some of the plaintext. A MAC is particularly important for binary additive stream ciphers. For this class of ciphers, flipping a bit of the ciphertext will flip the corresponding bit of the plaintext, and only affect that one bit. This can be used by an attacker to change messages. A MAC will prevent this type of attack. This property, that a change in the ciphertext will produce a known predictable change in the plaintext, is called malleability.

3.2.1 Searches over Secret Keys Space

The basic attack against any symmetric key cryptosystem is the brute force attack. In this attack, the attacker keeps guessing what the key is until they guess correctly. In general, one known plaintext, or the ability to recognize a correct plaintext is all that is needed for this attack. However, all good cryptosystems should be designed such that this attack is impractical. For a key of size n, a brute force search would consist of trying all 2^n keys to see which one works. If it is possible to recognize the correct plaintext, then on average the correct key will be found in 2^n.

Recently, a more efficient brute force attack method for stream ciphers based on the time/memory/data tradeoffs is reported in [1]. Time/memory/data tradeoffs are based on a speed up of a brute force search by pre-computing some values and storing them in memory and employment a large data collections. Accordingly, this is a trading of memory and data usage for time to perform the attack.

In [1] and [2], the sampling resistance is discussed in the context of time/memory/data tradeoffs. The sampling resistance shows how easy it is to find keys that generate keystreams with certain property of the output.

3.2.2 Distinguishing Attacks

A distinguisher is a probabilistic polynomial-time algorithm A. A takes as input N bits of data, which are either from the real stream cipher or are completely random data. A then has to, in polynomial time, output either Real or Random decision. If this is correct with a non-negligible probability, then this is a good distinguisher. The parameter N indicates how good the distinguisher is. In most cases, larger N will produce a more accurate distinguisher.

A distinction could be made between two different type of distinguisher. The first type of distinguisher is what has already been discussed, and is called a polynomial-time distinguisher.

These two notions are equivalent from the information-theoretical viewpoint, though the difference in bias can be signifiant.

3.2.3 Key Weaknesses

There are also some more specific models that should be discussed, because they are directly relevant for the discussion on the security of RC4. These models are related to the secret key characteristics. Recalling that the secret key completely determines the output sequence from a keystream generator, in a good keystream generator, each bit of the output should depend on the entire key, and the relationship between the key and a given bit (or set of bits) should be extremely complicated.

Key-Biased Output

The first condition is that every bit of the output is **dependent** on the entire key. This means that changing any single bit of a key should have a 1/2 probability of affecting each bit of the output. When this property holds, then in order to brute force a key, every possible key must be tried, and there will be no relationship between bits of the key and the output. This means that uncertainties of the individual bits of the key multiply when calculating the total number of possible keys.

Let us see what happens when this property does not hold. Assume that the 8 bits of the output is dependent only on the first 8 bits of the key. This means that the first 8 bits of the key can be recovered independently of all other key bits based on the first 8 output bits.

Accordingly, if we first check all possible values of the first 8 bits of the key, we can determine the correct ones by comparing the given output with the outputs generated by to the considered hypothesis. Once, these bits are determined, then the remained brute force should be preformed only over a reduced key space of dimension 2^{n-8}.

Related Key Attacks

The second condition is that the relationship between the key and each bit of the output should be **complex**. What this means is that a given known relationship between two keys should not produce a known relationship in the output of the keystream generator. This kind of information can also be used to provide an attack by reducing the effective brute keysearch time. These kind of attacks are known as related key attacks.

The following is an illustrative example. Assume we have a keystream generator that has the property that the bitwise compliment of the key produces the bitwise compliment of the keystream. This information can be used to (slightly) speed up a brute force search. For each keystream generated by a guess of the key, complement the keystream and see if that one generates the correct plaintext. If it does, then the complement of the key is correct. In this way, you never have to check the compliment of any key you have already checked. This reduces the search space by a factor of two.

3.2.4 Initialization Vectors Based Attacks

.

One problem with binary additive keystream ciphers is that the same key will always produce the same keystream. This means that repeatedly using the same key is just as bad as reusing a one-time-pad. To solve this problem, the concept of initialization vectors is very important. An initialization vector (IV) is a random value that changes with every instance of the cipher that is used to add some randomness to the output of the cipher. Since this value is random and unique, it makes the output of the stream cipher different than other outputs, even if the same key is used. This is useful when key exchange is expensive. A common method of adding an IV to a cipher is to combine it with the real key in some fashion. For example, prepending a small IV to the real key to form a larger session key is quite common. Another way of using an IV is to encrypt the IV with the real key, or encrypt the real key with the IV, and use this encrypted value as a session key.

It is important to recognize the proper role of the IV. In this kind of usage, the initialization vector is not part of the secret key, and does not need to be kept secret. This means that it can be transmitted in the clear to the recipient.

However, the use of IV's can be security weakness, depending on how the IV is used. A good example is any cipher in which partial knownledge of the key together with some ciphertext can be used to find more of the key. If this is true, then just prepending the IV to the key to form a session key can leak information about the real key. RC4 suffers from this problem.

Part III: Analysis of Various Cryptanalytic Techniques Relevant for RC4

4 Summary of Reported Results on RC4 Security Evaluation

RC4 has been an objective of a number of security evaluation analysis. These analysis were focused on all main approaches for the security evaluation including testing, consideration of resistance on cryptanalysis and strength of the modified primitive. Particularly, the reported results includes the following issues

- weak and related keys ([33], [37], [12], [19], [8])
- weaknesses of the key scheduling algorithm and recovering secret keys ([8])
- distinguishing RC4 outputs from the random ones ([14], [10]-[11], [7], [26])
- prediction of RC4 internal states or outputs ([6], [14], [7], [19])
- recovering RC4 internal state ([15], [21])

This section gives a summary of all these results, and the next two sections discusses some of them in more details.

The first part of this section deals with attacks based on the key schedule.

The second part of this section discusses attacks based solely on the pseudo-random generator algorithm. It includes observations about the progression of states as the algorithm runs and the related cryptanalytic algorithms, as well as the distinguisher of RC4 outputs from the randomness.

4.1 KSA Oriented Analysis

The fact that the initialization of RC4 is very simple stimulated considerable research on this mechanism of RC4. In particular, [33] pointed out a class of weak keys that reduces their effective size by five bits.

In [8] several weaknesses in the key scheduling algorithm of RC4, are reported and their cryptanalytic significance are described. We identify a large umber 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. These weak keys are employed to construct new distinguisher for RC4, and to mount related key attacks with practical complexities. Finally, it is shown that RC4 is completely insecure in a common mode of operation which is used in the widely deployed Wired Equivalent Privacy protocol (WEP, which is part of the 802.11 standard), in which a fixed secret key is concatenated with known IV modifiers in order to encrypt different messages. A new passive ciphertext-only attack on this mode can recover an arbitrarily long key in a negligible amount of time which grows only linearly with its size, both for 24 and 128 bit IV modifiers.

[8] reports an analysis of KSA and describes two significant weaknesses of this process.

The first weakness is the existence of large classes of weak keys, in which a small part of the secret key determines a large number of bits of the initial permutation (KSA output).

The second weakness is a related key vulnerability, which applies when part of the key presented to the KSA is exposed to the attacker. It consists of the observation that when the same secret part of the key is used with numerous different exposed values, an attacker can rederive the secret part by analyzing the initial word of the keystreams with relatively little work. This concatenation of a long term secret part with an attacker visible part is a commonly used mode of RC4, and in particular it is used in the WEP (Wired Equivalent Privacy) protocol, which protects many wireless networks.

The following issues are considered/presented in [8]:
- a slightly modified variant of the Key Scheduling Algorithm, and it is proved that a particular pattern of a small number of key bits suffices to completely determine a large number of state bits;
- it is shown that this weakness of KSA which we denote as the invariance weakness, exists (in a weaker form) also in the original KSA;
- it is shown that with high probability, the patterns of initial states associated with these weak keys also propagate into the first few outputs, and thus a small number of weak key bits determine a large number of bits in the output stream;
- several cryptanalytic applications of the invariance weakness are described, including a new type of distinguisher;
- initialization vector (IV) weakness is described and it is shown that a common method of using RC4 i vulnerable to a practical attack due to this weakness;
- it is shown how both of the above weaknesses can separately be used in a related key attack;
- also, it is examined how the IV weakness can be used to attack a real system, how the invariance weakness can be used to construct a ciphertext-only distinguisher and to prove that RC4 has low sampling resistance, and how to derive the secret key from an early permutation state.

4.2 PRNG Oriented Analysis

A class of states that RC4 can never enter is reported in [6]. This class contains all the states for which $j = i + 1$ and $S[j] + 1$ (a fraction of approximately N^{-2} of the RC4 states are in this class). 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. More detailed analysis of the state transition graph is reported in [21].

In [14] a probabilistic correlation between the secret information (S, j) and the public information (i, z) (in known message attacks), is reported in which the events $S[j] = i - z$

and $j - S[i] = z$ occur with approximately twice their expected probability.

A branch and bound attacks that are based on the "guesss on demand" paradigm are reported in [15] and [21]. These attacks simulates the generation process, and keeps track of all the known values S which had been deduced so far. Whenever an unknown entry in S is needed in order to continue the simulation, the attacker tries al the possible values (their number is typically smaller than N since known values in the permutation S can not be repeated). Actual outputs are used by the simulation to either deduce additional values in S (if the pointed value is unknown), or to backtrack (if the pointed value is known and different from the actual output). This tree search is simple to implement, and needs only $O(N)$ outputs. However, its enormous running time (which was analyzed both analytically and experimental in [15] makes it worse than exhaustive search for typical key sizes, and completely impractical for the values of n above 5.

Due to the huge effective key of RC4, attacking the PRGA seems to be infeasible (for $n = 8$, the best known attack on this part requires time that exceeds 2^{700}).

A different research direction was to analyze the statistical properties of RC4 outputs, and in particular to construct distinguisher between RC4 and truly random bit generators. In [10]-[11], a linear statistical weakness of RC4, caused by a positive correlation between the second binary derivative of the least significant bit (LSB) and 1, is reported. It is claimed that this weakness implies that RC4 outputs of length proportional to $2^{6n-7.8}$ ($2^{40.2}$ for the typical $n = 8$) can be reliably distinguished from random strings. This result was improved in [7], and a method of explicitly computing digraph probabilities is given. The distribution of triplets consisting of the two outputs produced at times t and $t + 1$ and the known value of $i = t(modN)$, are analyzed and it is found a small bias in the distribution of $(7N - 8)$ of these N^3 triplets: some of these probabilities were $2^{-3n}(1 + 2^{-n})$ (with a small positive bias), and some of these probabilities were $2^{-3n}(1 - 2^{-n})$ (with a small negative bias). The detected irregularities are used to mount a method for distinguishing 8-bit RC4 from randomness. The method from [7] requires less keystream output than previously published attacks, requiring $2^{30.6}$ bytes of output. Also, it is claimed that the estimation from [10]-[11] which will allow an attacker to distinguish RC4/8 from randomness with approximately 2^{40} successive outputs, appears to be somewhat optimistic. In [7] an information theoretic lower bound on the number of bytes needed to distinguish RC4 from randomness, is given implying the following: The required number of bytes of RC4/8 output is $2^44.7$ using anomaly from [10]-[11], while the irregularities in the digraph distribution require just $2^{30.6}$ bytes to achieve the same result. Also it is identified a special class of states, denoted as "fortuitous states", that are the source of most of these biases. These states can be used to extract part of the internal state with non-trivial probability, but since RC4 states are huge this does not lead to practical attacks on RC4 for $n > 5$.

A statistical weakness of RC4 outputs is reported in [19] which makes it trivial to distinguish between short outputs of RC4 and random strings by analyzing the second RC4 output byte. This weakness can be used to mount a practical cipher-text only attack on RC4 in some broadcast applications, in which the same plaintext is sent to multiple recipients under different keys.

According to the reported results, it seems that the only practical results related to the PRGA deal with the construction of distinguishers: [7] describes how to distinguish

RC4 outputs from random strings with 2^{30} data, and a better distinguisher which requires 2^8 data is described in [19] but this distinguisher could only be used to mount a partial attack on RC4 in broadcast applications.

Very recently, in [26] a necessary condition for existence of strong distingushers in the idealized RC4 model is reported. It yields a conservative estimate for the number of bytes that should be discarded in order to be safe, and it is is based on the distinguisher that detects a bias in the first byte. This phenomenon is almost as persistent and reliable as the bias in the second byte (reported in [19]) even though they are completely unrelated. Based on analysis in [26] it is recommended dumping at least 512 bytes.

5 Attacks Based on Key Scheduling Algorithm

5.1 Weak Keys

[33] has reported, a class of weak keys, for which the initial byte of the keystream is highly correlated with the first few key bytes. It is shown that the keys for which $K[0]+K[1] = 0 mod 2^n$ are weak keys. Assuming $n = 8$, when a weak key appears, 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 trivial 2^{-16}, and thus reduce the effective key length in exhaustive search about five bits. These weak keys occur because the keystream initialization algorithm swaps a given entry of the S-box exactly once (corresponding to when the pointer i points to the entry) with probability $1-\frac{255}{256}$. Another class of weak keys is described in [37]. Several classes of weak keys reported in [33] and [37] generate either predictable output or when these keys are used, several bytes of the key can be extracted from the output stream.

Recently, another class of weak keys is reported in [8]. A large number of weak keys is identified, such that knowledge of a small number of key bits suffices to determine many state and output bits with non-negligible probability. These weak keys can be used to construct new distinguisher for RC4, and to mount related key attacks with practical complexities. In other words, the weakness reported in [8] is the existence of large classes of weak keys, in which a small part of the secret key determines a large number of bits of the initial permutation (KSA output). In addition, PRGA translates these patterns in the initial permutation into patterns in the prefix of the output stream, and thus RC4 has the undesirable property that for these weak keys its initial outputs are disproportional affected by a small number of key bits. These weak keys have length which is divisible by some non-trivial power of two, i.e., $\ell = 2^q m$ for some $q > 0$. When RC4 with parameter n uses such a weak key of ℓ words, fixing $n + q(\ell - 1) + 1$ bits of K (as a particular pattern) determines $\theta(qN)$ bits of initial permutation with probability of one half and determines various prefixes of the output stream with various probabilities (depending on their length).

5.2 Related Key Analysis

In [12] RC4 is analyzed in a related key attack model. In this model, the keys related in a certain way cause very similar outputs for the first few output bytes.

Let us consider the difference in the initial state of a one byte difference in a full key: That is, $K'[i] = K[i]$ except for $i = t$, where $K'[i] \neq K[i]$. So, up to the point t, the key schedule (and the state) will be the same, but at the point t the value of $j = (j + S[i] + K[i])$ will be different for the two keys, and the remaining key schedule will be completely different. If t is small, then the resulting initial states for the two keys will be completely different. However, if t is near 2^n, then the initial states will be very similar.

Let, j_t denotes the value of j at the t-th step in the key schedule. Consider two complimentary changes in the key to produce a related key: $K'[i] = K[i]+\delta$ and $K'[i+1] = K[i+1] - \delta$. In this case, $j'_t = j_t + \delta$ and most likely $j'_{t+1} = j_{t+1}$. This is called twiddling the key at position t, and K and K' are related keys.

The swap on the t iteration affects future values of j or j' happens whenever $j_t > t$ or $j'_t > t$. The probability of this occurring is $1 - \frac{t}{2^n}$.

Unless $j_t = t + 1$ or $j'_t = t + 1$, immediately following the twiddle $j + t + 1 = j_{t+1}$.

Meaning of a good twiddle is that only three values in S differ from those in S' after the t-th step. Following a good twiddle, the key schedule will be identical until $S[i] \neq S'[i]$. At this point, the two key schedules will diverge.

5.3 The Invariance Weakness

Recall that we have already pointed out that [8] has reported a class of weak keys of RC4. This section summarizes the implications of these weak keys on the key-biased RC4 outputs as a consequence of the so called invariance weakness of KSA.

For this consideration, a variant of the RC4 Key Schedule Algorithm known as KSA* is employed in [8]. 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, or equivalently, i is initialized to 1 in KSA*.

```
**Initialization**:
For $i = 0$ to $2^n$
$S[i] = i$
$i = 0$
$j = 0$
**Scrambling**:
Repeat $N$ times
$i = i + 1$
$j = j + S[i] + K[i mod \ell]$
$Swap(S[i], S[j])$
```

Fig. The Key Schedule Algorithm Variant KSA*.

The key-output correlation implies the propagation of the weak key patterns into the generated outputs. It is considered in [8] according to the following.

Definition. Let S be an RC4 State table, t be an index in S and b be some integer. If $S[t] = t(mod b)$, then the state S is said to be b-conserve the index t. Otherwise, the permutatation S is said to be b-unconserve the index t.

Definition. Let $I_b(S)$ be the indices that the state S b-conserves. A permutation S of $\{0, ..., N-1\}$ is b-conserving if $I_b(S) = N$, and is almost b-conserving if $I_b(S) \geq N-2$.

Definition. Let b, ℓ be integers, and K be an ℓ word key. Then K is called a b-exact key if for any index r, $K[r mod \ell] = (1-r)\ (mod b)$. In case $K[0] = 1$ and $MSB(K[1]) = 1$, K is called a special b-exact key.

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

KSA* thus transforms special patterns of the key into corresponding patterns of the initial permutation. This means that a small subset of the bits of K completely determines a large number of bits in the initial state. The fraction of determined permutation bits is proportional to the fraction of fixed bits. For example, applying this result to RC4$_{n=8,\ell=6}$ and $q = 1$, 6 out of 48 key bits completely determine 252 out of the 1684 permutation bits (this is the number of bits encapsulated in the LSBs).

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

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

Regarding the Theorems note the following. During the first round, two deviations from KSA* execution occur. The first one is the non-equivalence of i and j whih is expected to cause non-equivalent entries to be swapped during the next rounds, thus ruining the delicate structure that was preserved so well during KSA* execution. The second deviation is that S b-unconserves two of the indices i_0 and $j_0 = K[0]$. However, we can cancel the ij discrepancy by forcing $K[0]$ (and j_0) to 1. In this case, the discrepancy in $S[j_0]$ ($S[1]$) causes an improper value to be added to j in round 1, thus repairing its non-equivalence to i during this round. At this point there are still two unconserved indices, and this aberration is dragged across the whole execution into the resulting permutation. Although these corrupted entries might interfere with j updates, the pseudo-random j might reach them before they are used to update j (i.e., before i reaches them), and send them into a region in S where they cannot affect the next values of j. The probability of this lucky event is amplified by the fact that the corrupted entries are $i_0 = 0$ which is not touched (by i) until the transformation of the KSA due to its distance from the current location of i ($i_1 = 1 + K[1] > N/2$) (recall that $MSB(K[1] = 1)$, that is far the position of i ($i_1 = 1$), which gives j many opportunities to reach it before i does. The probability of $N/2$ pseudo random j's to reach an arbitrary value can be bounded from below by 2/5, and extensive experimentations indicates that this probability is actually close to one half.

Accordingly, the primary difference between KSA and KSA* is that KSA does not preserve the equivalence of i and j after the first round. However, if K is a special b-exact key, then $K[0] = 1$, and the value of j after the first round is forced to be 1. Therefore, the equivalence between i and j is preserved except for the first round. Therefore, for a special b-exact key, the resulting initial permutation is almost b-conserving much of the time. If the value of i reaches one of the non-b-conserving indices, then the value of j will diverge from $i(mod b)$. So, if the most significant bit of $K[1]$ is 1, then the non-b-conserving values will be put in the second half of the permutation, where j has plenty of chance to meet it and swap it before i gets to it. Finally, [8] claims that it can be proven that the probability that a special b-exact key produces an almost b-conserving initial permutation is greater than 2/5, and in practice is usually greater than 1/2.

5.3.1 Key-Output Correlation

This section yields an analysis of the propagation of the weak key patterns into RC4 outputs. As the first a claim is given which deals with the highly biased behavior of a significantly weakened variant of the PRGA (where the swaps are avoided), applied to a b-conserving permutation. Next, it is shown that the prefix of the output of the original PRGA is highly correlated to the prefix of this swapless variant, when applied to the same initial permutation. These facts imply the existence of biases in the PRGA distribution for these weak keys.

Claim. Let RC4* be a weakened variant of RC4 with no swap operations. Let $q \leq n$, $b = 2^q$ and S_0 be a b-conserving permutation. Let $\{X_r\}_{r=1}^{\inf}$ be the output sequence generated by applying RC4* to S_0, and $x_r = X_r mod b$. Then the sequence $\{x_r\}_{r=1}^{\inf}$ is independent of the rest of the key bits.

Since there are no swap operations, the permutation does not change and remains b-conserving throughout the generation process. Notice that all the values of S are known $mod b$, as well as the initial indices $i = j = 0$, and thus the round operation (and the output values) can be simulated $mod b$, independently of S. Consequently the output sequence $mod b$ can be predicted, and deeper analysis implies that it is periodic with period $2b$.

Recall that at each round of the PRGA, S changes in at most two locations, and thus we can expect the prefix of the output stream generated by RC4 from some permutation S_0, to be highly correlated with the stream generated from the same S_0 (or a lightly modified one) by RC4*. In particular the stream generated by RC4 from an almost b-conserving permutation is expected to be highly correlated with the (predictable) substream $\{x_r\}$ from the Claim. 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}.

Accordingly, it is 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.

Since in each step of the RC4 swaps at most two values of S, the S permutation remains very similar to the S permutation in RC4*. Therefore, for a substantial number of outputs of RC4, the output will mirror that of RC4*. This fact can be used to form a distinguisher for RC4 from random for a subset of the possible keys. For these keys, the outputs of RC4 will be significantly similar to the outputs of RC4*, much more so than in the random distribution. Since the number of predetermined bits is linear in ℓ, the size of the bias is also dependent on ℓ. Another interesting observation is that the biases in the least significant bits of the output can be combined with the biases in the least significant bits of english text to provide a ciphertext-only distinguisher.

Finally, note the following: The previous also imply that RC4 has *low sampling resistance*. In [2] a new security measure of stream ciphers, which is denoted as their sampling resistance. The strong correlation between classes of RC4 keys and corresponding output patterns can be used to prove that RC4 has relatively low sampling resistance, which improves the efficiency of time/memory/data tradeoff attacks.

5.3.2 Related Keys Attack

In this attack model, the attacker is given a black box that has RC4 with a given key K in it. The attacker can give the black box a value, δ, and the black box will generate output using $K' = K \oplus \delta$ as the key. Accordingly, in this section, we discuss the related-key attacks based on weaknesses discussed previously in this paper. They work within the following paradigm: the attacker is given a black box that has a randomly chosen RC4 key K inside it, an output button and an input tape of $|K|$ words. In each step the attacker can either press the output button to get the next output word, or write δ on the tape, which causes the black-box to restart the output generation process with a new key defined as $K' = K \oplus \delta$. The purpose of the attacker i to find the key K (or some information about it)

Related-key attack based on the invariance weakness works when the number of key words, is a power of two. It consists of n stages where in stage q the q-th bit of every key word is exposed.

This attack works in n stages, where in stage q the Q-th bit of every key word is exposed. This attack has two algorithms that it uses.

The first is the $CheckKey$ algorithm. This algorithm accepts the RC4 black box, a value δ and a parameter q, and determines whether the output of the black box is special 2^q-exact. The second algorithm is $Expand$, which accepts a black box and a δ which form a special 2^{q-1}-exact and makes it special 2^q-exact. This is accomplished by enumerating all possibilities of the q-th bits of each key word and invoking $CheckKey$ to determine if it is a special 2^q-exact key.

The predicate $CheckKey$ takes as input an RC4 blackbox and a parameter q (the stage number) and decides whether the key in the box is special 2^q-exact. This purpose can be achieved by randomly sampling key bits that are irrelevant for the 2^q-exactness of the key and estimating the expected length of q-patterned output. For a special 2^q-exact key the expected length will be significantly longer than in a random output (where it is less than 2) and thus $CheckKey$ works in time $O(1)$. The procedure $Expand$ takes as input an RC4 blackbox and a parameter q (the stage number), assumes that the key in the box is special 2^{q-1}-exact, and makes it special 2^q-exact. The method for doing so is by enumerating all the possibilities for the q-th bits ($2^{\ell-1}$ such possibilities) and invoking $CheckKey$ to decide when the key in the box is special 2^q-exact. $Expand$ works in a slightly different way for $q = 1$ and $q = n$. For $q = 1$, except for the LSBs, it determines the complete $K[0]$ (by forcing it to 1) and MSB ($K[1]$). For $q = n$, there is only one 2^n-exact key and consequently we can calculate the output produced from this key and replace $CheckKey$ by simple comparison. The time complexity of this stage is $O(2^{n+\ell})$ for $q = 1$ and $O(2^{\ell-1})$ for any other q.

The total time required for the attack i thus $O(2^{n+\ell}) + (n-1)O(2^{\ell}) = O(2^{n+\ell})$. For typical RC4$_{n=8}$ key with 32 bytes, the complexity of exhaustive search i completely impractical (2^{256}), whereas the complexity of the new attack is only $O(2^{n+\ell}) = O(2^{40})$.

The second related-key attack proposed in [8] is based on the known IV weakness.

Both attacks are able to completely reconstruct the randomly chosen RC4 key with a number of chosen keys and amount of work that is significantly below that of brute force (except for extremely short RC4 keys). The first attack scales upwards as the key grows longer,while the time complexity of the second attack is independent of key length,with a cross-over point at $\ell = 8$. However, due to the second word weakness, future implementations of RC4 are likely to discard some prefix of the output stream, and in this case the second attack becomes difficult to apply - output word x depends on $2x + 1$ permutation elements immediately after KSA, and all the $2x + 1$ elements must occur before r for the resolved condition to hold. On the other hand, the first attack extends well, in that the probability of the output words being patterned drops modestly as the number of discarded words increases.

5.4 Initialization Based Attacks

This section deals with attacks on RC4 using an initialization vector. Specifically, for brevity, this section will focus on the case where the IV precedes the secret key. The case where the IV follows the secret key is covered in [8], as well as the attack presented here. First, some notation is necessary. Let i_t and j_t be the values of the two counters after t steps of KSA, and also S_t be the state of the permutation after t steps of KSA. The first word output from RC4 is dependant on only three values of S.

In this section, according to [8], we consider related key attacks where the attacker has access to the values of all the bits of certain words of the key. In particular, we consider the case where the key presented to the KSA is made up of a secret key concatenated with an attacker visible value (which we will refer to as an Initialization Vector or IV). It is show that if the same secret key is used with numerous different initialization vectors, and the attacker can obtain the first word of RC4 output corresponding to each initialization vector, he can reconstruct the secret key with minimal effort. How often he can do this, the amount of effort and the number of initialization vectors required depends on the order of the concatenation, the size of the IV, and sometimes on the value of the secret key. This observation is especially interesting, as this mode of operation is used by several deployed encryption systems and the first word of plaintexts is often an easily guessed constant such a the date, the sender's identity, etc, and thus the attack i practical even in a ciphertext-only mode of attack. However, the weakness does not extend to the Secure Socket Layer (SSL) protocol that browsers use, as SSL uses a cryptographic hash function to combine the secret key with the IV.

In terms of keystream output, this attack is interested only in the first word of output from any given secret key and IV. Hence, we can simplify our model of the output.

In addition, we will define the resolved condition as any time within the KSA where i is greater than or equal to $1, X$ and Y ,where X is defined as $X + S_i[1]$ (that is, $X + D$). When this resolved condition occurs,with probability greater than $e^{-3} \approx 0.05$, none of the elements $S[1]$, $S[X]$, $S[Y]$, will participate in any further swaps. In that case, the value will be determined by the values of $S_i[1]$, $S_i[X]$ and $S_i[Y]$. With probability less than $1 - e{-}3 \approx 0.95$, at least one of the three values will participate in a swap, which will destroy the resolved condition and set that element to an effectively random value. This will make the output value effectively random. The attack involves examining messages with specific IV values such that, at some point, the KSA is in a resolved condition, and where the value of $S[Y]$ gives us information on the secret key. When we observe sufficiently many IV values, the actual value of $S[Y]$ occurs detectably often.

Details of the Known IV Attack
The following notation is used for discussion of a concatenation of an IV and a secret key: the secret key is denoted as SK, the size of the IV by I, and the size of SK as $\ell - I$. The variable K still represents the RC4 key, which in this case is the concatenation of these two: $K[1...\ell] = IV[0]...IV[I-1], SK[0], ..., SK[\ell - 1 - I])$.

Recall that the first word output from RC4 is dependent on only three values of S.

Definition. Let $X = S_i[1]$ and $Y = S[X]$. In KSA, if $i \geq 1$, $i \geq X$, and $i \geq X + Y$, and $S_i[1]$, $S_i[X]$, and $S_i[X + Y]$ are all known, then the situation is called resolved. In this case, the first word output from RC4 will be $S[S[1] + S[S[1]]]$ with probability greater than $e^{-3} \approx 0.05$.

This probability comes from the probability that none of the three relevant values will be disturbed in the rest of the key schedule. Since the value of i has already passed the three values in the resolved condition, only by j pointing to one of those values can they be disturbed. Modelling j as a random value for the rest of the key schedule, the relevant probability is the probability that j never equals one of those three specific values.

Let us consider the case where a 3 word IV is prepended to a secret key to form the session key. If K' is the secret key and IV is the initialization vector, then $K = IV||K'$ is the session key. Since the IV is 3 words long, and the secret key is ℓ words long, the session key is always $3 + \ell$ words long. The secret key values are $K[3]...K[l + 2]$, and the (known) IV values are $K[0], K[1], K[2]$.

Let us consider the case where the first A values of the secret key $(K[3]...K[A + 2])$ are known, and let us examine a series of IV's of the form $(A + 3, -1, X)$, where X is a random variable.

In the first step, j is advanced by $S_0[0] = A + 3$. In the next step, i is advanced, and the advance on j is computed to be $j_1 = j_0 + S_0[1] + K[1] = j_0 + 1 + (-1)$. Therefore, j is not moved. Then $S[i]$ and $S[j]$ are swapped. On the next step, j is advanced by $X + 2$. From here on until $i = A + 3$, the key schedule can be computed. Since all the X's are different, each IV acts differently. At this point $(i_{A+2} = A + 2)$, the value of j_{A+2} and of the entire permutation S_{A+2} are known. At this point, if either $S[0]$ or $S[1]$ have been disturbed, the attacker throws out this IV. We also known $S_{A+2}[A + 3]$. The value that will be $S_{A+3}[A + 3]$ depends on $j_{A+3} = j_{A+2} + S_{A+2}[A + 3] + K[A + 3]$, which is two knowns and one unknown. Therefore, with probability approximately 0.05, none of the three important values will be touched by the remaining key schedule, and the first output word will be $S[A+3]$.

In this case, it is possible to calculate $K[A+3]$. Let's expand some formulas and see what can be derived:

$$\begin{aligned} z_0 &= S[S[1] + S[S[1]]] \\ &= S[0 + S[0]] \\ &= S[A+3] \\ &= S_{A+2}[j_{A+3}] \\ &= S_{A+2}[j_{A+2} + S_{A+2}[A+3] + K[A+3]] \end{aligned}$$

At this point, we know S_{A+2}, j_{A+2} and z_0 (the first output word). Therefore we can calculate $K[A+3]$ as:

$$K[A+3] = S^{-1}_{A+2}[z_0] - j_{A+2} - S_{A+2}[A+2] \ .$$

If we use approximately 60 IV's of the form above, we can calculate secret key byte A. If we iterate this attack over all the secret key bytes (ℓ bytes), it takes approximately 60ℓ chosen IV's and their corresponding first output byte to determine the entire secret key.

Generalization of the IV's.
First, let us see what conditions on the 3-byte IV's are necessary to make this attack work. The two most important conditions are:

1. We know the state S_{A+2} in its entirety.
2. The first output byte is $S_{A+3}[A+3] = S[S[1] + S[S[1]]]$. This leads to the condition that $S_3[1] + S_3[S_3[1]] = A+3$.

Also, for the resolved condition to hold, the values at time 3 must all be less than 3. Therefore, the additional condition of $S[1] < 3$ must also hold. Note that the condition 1 always holds if we know the IV.

These conditions are sufficient to carry out this attack. The number of IV's that fit these criteria is much larger than the number of IV's that fit our previous criteria. Therefore, a known IV attack, where the attack does NOT get to choose the IV's, is accelerated by using these criteria.

As a matter of fact, none of these criteria make significant use of the fact that the IV is 3 words long. So, the attack can be generalized to attack an IV of any size. Corresponding criteria for IV's of length I words are the following ones:
1. $S_I[1] + S_I[S_I[1]] = I + B$
2. $S_I[1] < I$.
In this case, the relevant equation can be generalized to the following one:

$$K[A] = S^{-1}_{I+B-1}[z_0] - j_{I+B-1} - S_{I+B-1}[I+B] \ ,$$

where $K[A]$ is the A-th byte of the secret key (not including the IV).

Finally, note the following. The Wireless Encryption Protocol (WEP), uses RC4 with a prepended 3 byte IV for its encryption. Also, the first byte of the plaintext is always a known value. The previous attack has been successfully used to attack WEP encryption, requiring roughly 5,000,000 packets with the same key to determine that key. This can be obtained quite easily and is considered a major weakness in this system.

6 Attacks Based on Pesudo Random Generation Algorithm

6.1 Distinguishing RC4 Streams from Randomness

The keystream sequence length needed to detect the weakness is considerably shorter than the period. Although the weakness may not lead to a significant plaintext uncertainty reduction (in the known ciphertext scenario), it is structure-dependent and may be used as such to distinguish between different types of keystream generators and for secret key reconstruction (in the known plaintext scenario).

6.1.1 Distinguishing Based on the Second Binary Derivative

The first distingusher for RC4 output sequence has been proposed in [10]-[11]. The main objective of [10]-[11] was to derive linear models for RC4 by using the Linear Sequential Circuit Approximation (LSCA) method. The LSCA method consists in determining and solving a linear sequential circuit that approximates a given keystream generator and yields linear models with comparatively large correlation coefficient c, where the probability of the corresponding linear relation among the keystream bits is $(1+c)/2$. It also gives an estimate of c but sometimes, as in the case of RC4, special techniques have to be developed to obtain more accurate estimates.

Let $z = \{z_t\}_{t=0}^{\mathrm{inf}}$ denote the least significant bit output sequence of RC4 and let

$$z' = (z'_t = z_t + z_{t+1})_{t=1}^{\mathrm{inf}}$$

and

$$z'' = (z''_t = z_t + z_{t+2})_{t=1}^{\mathrm{inf}}$$

denote its first and second binary derivatives, respectively. The results [10]-[11] show that z' is correlated neither to 1 nor to 0, and that z'' is correlated to 1 with the correlation coefficient close to $15 \cdot 2^{-3n}$ for large 2^n. Since the output sequence length needed to detect a statistical weakness with the correlation coefficient c is $O(c^{-2})$, the required length is around $64^n/225$. For example, if $n = 8$, as recommended in most applications, the required length is close to $2^{40} \approx 10^{12}$. It is claimed in [11] that the experimental results agree well with the above theoretical predictions.

Also, in [11] some relevant correlation properties of random Boolean functions are derived, and the linear models of RC4 and the corresponding correlation coefficients are determined, as well as central moments of underlying discrete probability distributions needed for estimating the correlation coefficients.

Accordingly, the distinguisher proposed in [10]-[11] should be regarded as a statistical weakness, at least on a theoretical level. Moreover, the output sequence length required for the detection may even be realistic in high-speed applications if $n \leq 8$. Also note that the second binary derivative weakness involves only three successive least significant output bits, which is much smaller than the memory size. The weakness is a consequence of a very simple next-state function of RC4. It is also shown that similar linear relations hold for other output bits as well, but the correlation coefficients are smaller.

6.1.2 Digraph Irregularities

An algorithm for deriving the exact probability of a digraph in the output of RC4 stream cipher is reported in [7]. This algorithm has a running time of approximately 2^{6n}, where n is the number of bits in a single output. Using the computed probabilities of each digraph for the case when $n = 5$, it is indicated that the digraphs have probabilities different from the value expected from a uniform random distribution of digraphs. Extrapolating this knowledge, it is shown how to distinguish the output of the alleged RC4 cipher with $n = 8$ from randomness with $2^{30.6}$ outputs.

This result improves the best known previously reported method for distinguishing RC4 output sequence from a truly random one. In addition, heuristic arguments about the cause of the observed anomalies in the digraph distribution are offered.

The irregularities in the digraph distribution that are reported in [7] allow the recovery of certain RC4 parameters if the attacker happens not to know them. An attacker can use this information in a ciphertext-only attack, to reduce the uncertainty in a highly redundant unknown plaintext.

Also, it is shown how an attacker can learn, with nontrivial probability, the value of some internal variables at certain points by observing large portions of the keystream. Although the previous does not yield a possibility to derive the entire state from this observation, this insight might lead to an exploitable weakness in the cipher.

The probability with which each digraph (that is, each successive pair of n-bit outputs) will appear in the output of RC4 is directly computable, given some reasonable assumptions. The probability of each digraph for each value of the i index is also computable. By taking advantage of the information on i, rather than averaging over all values of i and allowing some of the detail about the statistical anomalies to wash away, it is possible to more effectively distinguish RC4 from randomness.

The full digraph distributions for $n = 3; 4;$ and 5, are computable with about 2^{40} operations, and these distributions were computed and reported in [7]. It is shown that these distributions are significantly different from a uniform distribution. In addition, there is a consistency (across different values of n) to the irregularities in the digraph probabilities. In particular, one type of digraph is more probable than expected by a factor of approximately $1 + 2^{-n+1}$, seven types of digraphs are more probable than expected by approximately $1 + 2^n$, and three types of digraphs are less probable than expected by approximately $1 + 2^n$.

Extrapolating the approach to higher values of n without directly computing the digraph probabilities, is performed in [7] as follows: the probabilities of positive events and negative events by running RC4/8 with several randomly selected keys and counting the occurences of those events in the RC4 output. The observed probabilities (derived using RC4/8 with 10 starting keys for a length of 2^{38} for each key), along with the computed expected probability from a truly random sequence, imply the following: It is possible to distinguish between these two probability distributions with a 10% false positive and false negative rate by observing $2^{30.6}$ successive outputs.

In order to evaluate the effectiveness of the "extrapolation" approach, it is computed in [7] the amount of keystream needed using a test based on the observed positive and negative events, and compare that to the best possible test using the exact probabilities.

These numbers agree to within a small factor, suggesting that the extrapolation approach is close to optimal.

Also note that origins of the above anomalous are also pointed out in [7] by an analysis of the next state function of RC4 and showing mechanisms that cause the increased (or decreased) likelihood of some of the anomalous digraphs.

Finally, note that at this moment it is not known how to extend this distinguishing technique to an attack for find the original key or the entire internal state of the cipher, further research may be able to extend these observations into an attack that is more efficient than exhaustive search.

6.1.3 Second Byte Bias

It has been reported a number of analysis on the probabilities of any given value being output of RC4. Most of these analyses have approached RC4 by looking at a given output. Note that RC4 was subjected to rigorous statistical tests, but they typically consisted of taking a single stream with billions of output words and counting the number of times each value occurred along it. Even if this experiments is repeated many times with different keys, the strong bias of the second output word is not going to be visible in such an experiment.

In [19], a different approach is employed: The focus is shifted to a polynomial-space distinguisher. Recall that, two different distinguisher are: distinguisher in polynomial time and distinguisher in polynomial sampling. Although the difference between these types of statistical experiments is not new, the obtained distinguisher is a new one.

The following result has been reported in [19].

Theorem. If $S_0[2] = 0$ and $S_0[1] \neq 2$, then $Z_1 = 0$ with probability 1.

Theorem. Given a random initial state, the value of the second output of RC4 is 0 with probability approximately equal to $\frac{2}{2^n}$.

This means that when the second output is 0, we can guess the value of $S_0[2]$ with probability 1/2. Unfortunately, there is no one reported result how to use this to speed up an attack, at least for RC4$_{n>5}$. On the other hand, a polynomial-space distinguisher can be constructed from this information based on the following theorem.

Theorem. Let X, Y be probability distributions, and suppose event e happens in X with probability p, and happens in Y with probability $p(1+q)$. Then, for small p and q, the number of samples needed to distinguish between X and Y is $O\frac{1}{pq^2}$ with constant probability of success.

Accordingly, [19] has pointed out 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 $O(N)$ different output 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 [7]).

6.1.4 Easily Recognizable Patterns

A distinguisher based on easily recognizable patterns is reported in [8]. This 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.

Recall that in [19] it is described a significant statistical bias in the second output word of RC4. This bias is used to construct an efficient algorithm which distinguishes between RC4 outputs and truly random sequences by analyzing only one word from $O(N)$ different outputs streams. This i 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 [7]). The 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 i 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. 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 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 LSBs distribution can be combined in a natural way with the biased distribution of the LSBs 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.

Recall that the most of distinguisher reported in the literature (for RC4 and other stream ciphers) assume knowledge of the plaintext in order to isolate the XORed key stream. 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 key-stream is not alway 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). It will be shown here that if the plaintexts are English texts, it is easy to construct a ciphertext-only distinguisher from the considered biases. The intuition of this construction is that the considered biases 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 key-stream words in various ways. An approach is to combine the distribution of the first LSB of the RC4 output stream, with the first order statistics of English texts.

Theorem. Let C be the ciphertext generated y RC4 from a random key and the ASCII representation of plaintexts, distributed according to the first order statistics of English texts. Let p be the probability of a random key to be special 2-exact. Then C can be distinguished from a random stream by analyzing the first few words of about $200/(p^2)$ different RC4 streams.

For example, for $\mathrm{RC4}_{n=8}$ with 8 byte keys, $p = 2^{-16}$, which implies a reliable ciphertext-only distinguisher that works with less than 2^{40} data. The proof of the Theorem is based on the observation that the LSB of a random English text character is zero with probability of about 55%. It is important to note that the Theorem does not use all the statistical information which is available in either the key-stream or the plaintext distributions, and consequently doe not represent the best possible attack.

6.2 Predictive States

[7] and [19] have reported the classes of states which with a non-negligible bias appear in the keystream. These states are known as predictive states. Existence of the predictive states can be considered as a particular weakness of RC4. Each predictive state can give rise to some biased outputs, and we analyze the size of such biases as a function of certain parameters of these states. Also, it can be shown that such states can be used to improve the time and data complexities of branch and bound attacks on the internal state of RC4.

A more detailed discussion of the predictive states issue include the following.

Definition. An a-state is a partially specified RC4 state that includes i, j, and a (not necessarily consecutive) elements of S.

Definition. Let A be an a-state. Suppose that all the RC4 states that are compatible with A produce the same output word after r rounds. Then A is said to predict its r-th output.

Definition. Let A be an a-state and suppose that for some $r_1, ..., r_b \leq 2N$, A predicts the outputs of rounds $r_1, ..., r_b$. Then A is called b-predictive.

Intuitively, a b-predictive a-state can be associated with a sequence of b specific (round, output) pairs. Therefore, a b-predictive a-state is a state where b outputs are predicted accurately by just the values in the state.

The interesting part is that the b values are not necessarily consecutive, and to not necessarily follow directly after a. It is hypothesized that b-predictive a-states only exist when $b \leq a$.

In general, any b-predictive a-state can introduce some bias in the output of RC4. Denote the event E_A that the current state is compatible with a given a-state A. Denote as E_B the event that the outputs in rounds $r_1 ... r_b$ are those predicted by A.

E_A includes $a + 2$ constraints (i, j, and a elements of S) and thus (assuming $N = 2^n$) has a probability of

$$\frac{1}{N^2}\frac{N!}{(N-a)!} \approx N^{-(a+2)} \quad .$$

Whenever E_A occurs, E_B occurs with probability 1, and whenever E_A does not occur, E_B typically happens with the probability $N^{-(b+1)}$.

Accordingly it can be shown that the following holds.

Theorem. For an b-predictive a-state, there exists a distinguisher that distinguishes RC4 from random based on $O(N^{2a-b+3})$ output words.

Cryptanalytic attacks based on the predictive states can be mounted as the following.

Applying Bayes rule we can obtain the following result:

$$P[E_A|E_B] = \frac{P[E_A]}{P[E_B]} P[E_B|E_A] \approx N^{b-a-1} \ .$$

In this case, E_A is the internal event (that the state is a-predictive), and E_B is the external event, an indicator of the internal event E_A. Thus, when we observe E_B, we can conclude with probability N^{b-a-1} that E_A has occurred. With enough occurrences of the appropriate predicted outputs, one of the b-predicted output sequences is likely to correspond to the correct a-state. Intuitively, the value a determines the quantity of information that is revealed, but it also decreases the probability of it occurring.

Note that having j and a values of S we can apply the branch and bound attack [15] to reveal the rest of the secret state of RC4.

Intuitively, a determines the quantity of the revealed information, but increasing a reduces the probability for E_A and increases the required number of outputs. Consequently, the attack s limited to small values of a.

The attack requires an efficient way to find in a preprocessing stage all a-predictive a-states of RC4 for small values of a. This is a non-trivial problem whose complexity is not well understood at the moment.

6.3 Internal State Recovering Oriented Attacks

The cryptanalytic attacks against RC4 targeting recovering the initialization S of RC4 are reported in [15] and [21].

The idea behind these attacks is to guess some of the initial state of the RC4 keystream generator, and by looking for contradictions in the keystream it is possible to detect incorrect guesses and to discover the rest of the initial state.

[15] yields attacks on weakened versions of RC4. The weakened RC4 variants that they studied change their internal state less often than does RC4, though they change it in a similar way. The basic attack backtracks through the internal state, guessing values

of table entries that have not yet been observed, and backtracking upon contradictions. Several variants of the attack are presented, and the runtimes are analyzed. It is estimated that the complexity of the attack is less than the square root of the number of possible RC4 states.

In [21] an analysis of the cycle structure of RC4 is given. It is observed that the state of the permutation can be recovered, given a significant fraction of the full keystream. In addition, it is also presented a backtracking algorithm that can recover the permutation from a short keystream output. The analyses are supported by experimental results on RC4/n for $n < 6$, and showing that an RC4/5 secret key can be recovered after only $2^4 2$ steps, though the internal secret state size is about 160 bits.

6.3.1 Attack A

Let i_t and j_t be the values of the two counters at time t. Also, let S_t be the state at time t. Therefore, the output of RC4 at time t is

$$z_t = S_t[S_t[i_t] + S_t[j_t]] \ . \tag{5}$$

Following [15], let us begin our analysis by looking at simplified versions of RC4. Consider RC4 without any swap operation. (This only applies to the PRGA). This means that the state S_t is the same for all time values. Denote this S. This leads to the following statemnt.

Theorem. If the swap operation of RC4 is omitted, the keystream becomes cyclic with a period of 2^{n+1}.

The algorithm to recover S works as follows. Initially, guess a small subset of the values of S. Use the RC4 equations as time t progresses to derive additional values of S. If a contradiction arises, then the initial guess was incorrect. Repeat this process for all possible guesses. Note that i_0 and j_0 are both known when the algorithm starts. If the guesses of S correctly guess the first x values of S, then j_0 through j_x are known.

Note that, according to the above equation, at each time instant t there are up to four unknowns. At any time t, i_t is always known. The four variables that are possibly unknown are j_t, $S[i_t]$, $S[j_t]$, and $S^{-1}[z_t]$. From any three of these variables, the fourth can be calculated according to the following:

$$S[j_t] = S^{-1}[Z_t] - S[i_t] \ , \tag{6}$$

$$S[S[i_t] + S[j_t]] = Z_t \ , \tag{7}$$

$$j_t = S^{-1}[S^{-1}[Z_t] - S[i_t]] \ , \tag{8}$$

$$S[i_t] = S^{-1}[Z_t] - S[j_t] \ . \tag{9}$$

Once i_t goes past x, if $S[x+1]$ is not known, we lose the knowledge of j_t. We discard the next values of z_t until we can use relevant equation to discover j_t again. We can do this when $S[i_t]$ is known, the value z_t appears in what is known of S, and the value $S^{-1}[S^{-1}[Z_t] - S[i_t]]$ also appears in what is known part of S. Once we have finished working backwards, we continue as we started to discover values of S from the point we lose the value of j_t.

Using this algorithm, the entire state of this variant of RC4 can be determined. This is a fairly simple attack and does not require much time at all. Others variants studied in [15] include having a reduced swap frequency, where the swap operation is only executed once every 2^n iterations. From the results from [15], a swap frequency of 2^{-7} requires 40 correctly guessed values, and a swap frequency of 2^{-1} requires 240 correctly guessed values of S, assuming the success rate 50%.

Also, it is discussed in [15], how a modification of the previous attack can be used for attacking the full RC4. In this modification, no values of S are guessed initially, but are only guessed as needed. Since S is a permutation, this is useful to limit the number of possibilities for each guess, because the value cannot be one that is already in the table at the time. This can be implemented efficiently by a recursive function. It is reported in [15] that the running time of this algorithm is $O(\sqrt{2^n!})$.

6.3.2 Attack B

This section yields a summary of the technique proposed in [21] for recovering RC4 internal state.

Algorithm given below outlines a basic attack which can be mounted against RC4. In essence, the algorithm keeps track of all states which RC4 could be in, given that a particular keystream has been generated.

Algorithm for RC4 Internal State Recovering (Forward Tracking)

- Mark all entries S_t as unassigned.
- Set $i = 0$, $j = 0$, and $z = 0$.
- Repeat:
  1. Set $i = i + 1 \ mod 2^n$.
  2. If S_i is unassigned, continue with the remainder of the algorithm for each possible assignment of S_i.
  3. Set $j = (j + S_i) mod 2^n$.
  4. If S_j is unassigned, continue with the remainder of the algorithm for each possible assignment of S_j.
  5. Swap S_i and S_j.
  6. Set $t = (S_i + S_j) mod 2^n$.
  7. If S_t is unassigned and K_z does not yet appear in the s-box, set $S_t = K_z$.
  8. If $S_t \neq K_z$, the state information is incorrect. Terminate this round.
  9. Increment z.

10. If z is equal to the length of the keystream, output the current state as a solution and terminate the run.

Two cases are considered for the observed keystream; an arbitrarily chosen nonzero keystream, and a zero keystream. The zero keystream can be analysed more quickly than a more general keystream.

Several variations of this algorithm are possible. Backtracking, in which the keystream is processed in reverse, appears to be easier to implement efficiently because s-box entries are fixed sooner. Probabilistic variants, which use truncated tracking analysis or other information to determine the "best" node to follow in the tracking analysis, may be able to analyse more keystream, avoiding keystreams which result in a more difficult search. These variants as well as the performance of these algorithms are discussed in [21].

These results imply an upper bound on the complexity of RC4 cryptanalysis discussed in the following table.

Table 1: Estimated upper bound on the complexity of cryptanalysis of RC4-n, [21].

n	dimension of internal state (in bits)	effective dimension of internal state (in bits)	attack [21] complexity arbitrary keystream	attack [21] complexity zero keystream
2	2^{8}	$2^{4.58}$	2^{4}	2^{3}
3	2^{24}	$2^{15.30}$	2^{8}	2^{7}
4	2^{64}	$2^{44.25}$	2^{20}	2^{17}
5	2^{160}	$2^{117.66}$	2^{69}	2^{42}

Part IV: Proposal of a Novel Technique for Cryptanalysis of RC4

7 A Novel Cryptanalysis of RC4

This section yields a novel algorithm for cryptanalysis of RC4. The developed algorithm follows certain reported results on the cryptanalysis based on the time-memory-data trade-off approach and it employs a particular RC4 key/output correlation. On the other hand, the developed algorithm is a complete implementation of the time-memory-data trade-off concept which is based on a number of specific elements relevant for RC4 cryptanalysis.

7.1 Origins

The origins for developing an algorithm for cryptanalysis of RC4 are given by the following statements.

Theorem 1. With probability $p = (1 - \frac{3}{2^n})^{2^n-3}$, a secret key K with the first $3n$ bit-values which correspond to the integers 0, 0 and $2^n - 1$, yields as the first two keystream sequence elements 0 and 0, assuming $n > 2$.

Proof. Let $S(i)$, $i = 0, 1, ..., 2^n - 1$, denotes current value of the ith location of S.

When the first $3n$ bits of a secret key K correspond to the integers 0, 0 and $2^n - 1$, the evolution of KSA yields that the first three locations of S are given by the following:

time instant i	$S(0)$	$S(1)$	$S(2)$
0	0	1	2
1	0	1	2
2	2	1	0

On the other hand, the probability P that the first three locations of S will not be updated during the reaming $2^n - 3$ time instants of the initialization is equal to the following:

$$p = (1 - \frac{3}{2^n})^{2^n-3} \, , \tag{10}$$

Accordingly, when the first three locations are not updated during the remain $2^n - 3$ initialization steps of S, the keystream generation algorithm directly implies the following evolution of the S and the output.

time instant $i+1$	$S(0)$	$S(1)$	$S(2)$	PRNG output
1	2	1	0	0
2	2	0	1	0

Accordingly, the first two output symbols will be 0 and 0.

Theorem 1 directly implies the following corollary.

Corollary 1. When $n = 8$, with probability $p = (1 - \frac{3}{256})^{253} \approx 0.05$, a secret key K with the first 24 bit-values which correspond to the integers 0, 0 and 255, yields as the first two keystream sequence elements 0 and 0.

Theorem 1 implies a strong correlation between certain secret keys and the beginning of corresponding output sequences. This correlation can be employed for mounting, based on the re-synchronization, an efficient time-memory-data trade-off attack for recovering a large class weak seesion keys which have the prefix consisting of $2n$ zeros and n ones.

Recall that most of the Time/Memory/Data tradeoff attacks on stream ciphers are based on the following paradigm. The attacker keeps a database of $[state, output]$-pairs (sorted by output) and lookup every subsequence of the output stream in this database. When a (sufficiently long) database sequence is located in the output, the attacker can conclude that the actual state is the one stored along with this sequence and predict the rest of the stream.

In a general case, a drawback of this approach is that the large database must be stored in a hard disk(s) whose random access time is about a million times slower than a computational step.

To improve that attack we can keep on disk only states which guaranteed to produce outputs with some (relatively) rare but easy recognizable property (e.g., starting with some prefix). In this case only output sequences that have this exspecial property have to be searched in the database, and thus the expected time and the expected number of disk probes is significantly reduced.

On the other hand, in general, producing a pair $[state, output]$ with such a rare property costs much more than producing a random pair. $O(1/p)$ random states are required to obtain a single pair, where p is the probability of a random output to have the desired property.

However, if we can efficiently enumerate states that produce such outputs, the number of sampled states decreases dramatically, and this method can be applied without significant additional cost during the preprocessing stage. Such an attack can be applied to RC4 in two ways, based on the KSA and PRGA parts. An attack on the generation part constructs a database of pairs $[RC4state, outputsubstring]$ and analyzes all the substrings along a single output stream. The database construction is very simple since it is easy to enumerate states which produce outputs that have some constant prefix. However, this enumeration seems to be useless due to the huge effective key of this part (1684 bits) which makes such a tradeoff attack completely impractical. A more promising approach is based on the KSA part which uses a key of 40-256 bits and might be vulnerable to tradeoff attacks. In this case, the pairs in the database are [secret-key, prefix of the output stream], and the attack requires prefixes from a large number of streams (instead of a single long stream).

The correlation described by Theorem 1 provides an efficient sampling of keys that are more likely to produce output prefixes of the patterned type specified above

7.2 Preliminaries: Time-Memory-Data Trade-Off Concept

This section yields an overview of the time - memory - data trade-off concept according to [13] and [1].

Let $f(\cdot)$ denotes a one-way function, and K is the secret key. So, computing $f(K)$ is simple, but computing K from $f(K)$ is equivalent to cryptanalysis.

The time-memory trade-off concept is based on the following two phases: precompilation phase which should be performed only once, and the processing phase which should be performed for reconstruction of a particular secret key.

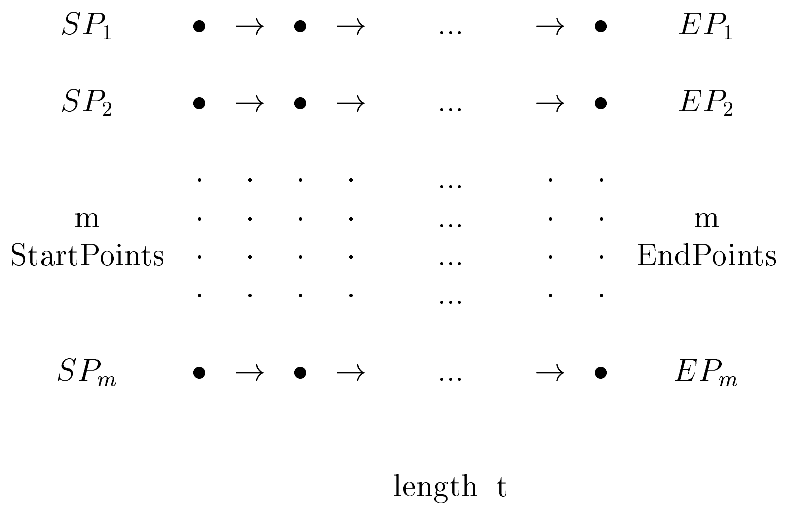

Figure 1: The underlying matrix for time-memory trade-off.

As part of the precomputation, the cryptanalyst chooses m starting points, $SP_1, SP_2, ..., SP_m$, each an independent random variable drawn uniformly from the key space $\{1, 2, ..., N\}$. For $1 \leq i \leq m$ he lets

$$X_{i0} = SP_i \tag{11}$$

and computes

$$X_{ij} = f(X_{i,j-1}) \ , \ \ 1 \leq j \leq t \ , \tag{12}$$

following the scheme given in the above figure. The parameters m and t are chosen by the cryptanalyst to trade-off time against memory.

The last element or endpoint in the ith chain (or row) is denoted by EP_i. Clearly,

$$EP_i = f^t(SP_i) \ . \tag{13}$$

To reduce memory requirements, the cryptanalyst discards all intermediate points as they are produced and sorts $\{SP_i, EP_i\}_{i=1}^{m}$ on the endpoints. The sorted table is stored as the result of this precomputation.

Now suppose someone chooses a key K and the cryptanalyst is obtained with

$$C_0 = S_K(P_0) \ , \tag{14}$$

where $S_K(\cdot)$ denotes the enciphering operation under key K, P_0 is a plaintext, and C_0 is corresponding ciphertext.

Suppose that the following is valid

$$Y_1 = f(K) \ . \tag{15}$$

Note that the cryptanalyst can check if Y_1 is an endpoint in one "operation" because the $\{(SP_i, EP_i)\}$ are sorted on the endpoints. Accordingly, note the following:

- If Y_1 is not an endpoint, the key is not in the next to the last column in Fig. 1. (If it is there, Y_1, which is its image under f, would be an endpoint.)

- If $Y_1 = EP_i$, either $K = X_{i,t-1}$ (i.e., K is in the next to last column of Fig. 1), or EP_i has more than one inverse image. We refer to this latter event as a false alarm. If $Y_1 = EP_i$, the cryptanalyst therefore computes $X_{i,t-1}$ and checks if it is the key, for example by seeing if it deciphers C_0 into P_0. Because all intermediate columns in Fig. 1 were discarded to save memory, the cryptanalyst must start at SP_i and recompute $X_{i,1}, X_{i,2}, ...$, etc. until he reaches $X_{i,t-1}$.

- If Y_1 is not an endpoint or a false alarm occurred, the cryptanalyst computes

  $$Y_2 = f(Y_1) \tag{16}$$

  and checks if it is an endpoint. If it is not, the key is not in the $t-2$nd column of Fig. 1, while if $Y_2 = EP_i$ the cryptanalyst checks if $X_{i,t-2}$ is the key.

- In a similar manner, the cryptanalyst computes

  $$Y_3 = f(Y_2), \ ..., \ Y_t = f(Y_{t-1})$$

  to check if the key is in the $t-3$rd, ..., or 0th column of Fig. 1.

If all mt elements in the 0th through $t-1$st columns of Fig. 1 are different and if K is chosen uniformly from all possible values, the probability of success $Pr(S)$ would be mt/N. Only m words of memory and t operations are required, so the time-memory product has come into play. An exhaustive search with t operations has only $Pr(S) = t/N$, while a table lookup with m words of memory has only $Pr(S) = m/N$.

If the matrix in Fig. 1 has some overlap, but a fixed fraction of distinct elements, the probability of success is only lowered by the same fixed fraction. A mild amount of overlap therefore can be tolerated in the matrix without affecting the basic gain inherent in the time-memory trade-off. The analysis also neglects other constant and logarithmic factors (e.g., it counts an encipherment and check for Y_1 equal to an endpoint as one operation).

Finally, suppose that we have D different ciphertexts obtained by a stream cipher. The attack is successful if any one of the D given outputs can be found in the matrix corresponding to Fig. 1. Accordingly we can reduce the total number of points covered by the matrix from about N to N/D points, and still get (with high probability) a collision between the stored and actual states obtaining a desired time-memory-data trade-off.

Assuming that P denotes the pre-processing complexity, the time - memory - data trade-off attack on stream ciphers proposed in [1] satisfy the following relation $P = N/D$, $tm^2D^2 = N^2$ for any $D^2 \leq t \leq N$. A typical point on this trade-off relation is $P = N^{2/3}$ pre-processing time, $t = N^{2/3}$ attack time, $m = N^{1/3}$ memory space, and $D = N^{1/3}$ available data. For $N = 2^{100}$ the parameters $P = t = 2^{66}$ and $m = D = 2^{33}$ are all (barely) feasible, [1].

In the following sections a method for the cryptanalysis of RC4 is proposed. As the first, underlying ideas for developing the algorithm are pointed out. The algorithm is proposed by specification of its pre-processing and the processing phases. Finally the algorithm complexity is discussed.

7.3 Underlying Ideas for Developing the Cryptanalytic Algorithm and Notations

Recall that the time - memory - data trade-off is based on the following: The **birthday paradox** implies that two random subsets of a space with N points are likely to intersect when the product of their sizes exceeds N.

Any time-memory-data trade-off approach for cryptanalysis consists of the following two phases: pre-processing phase and realtime (processing) phase. During the pre-processing phase which can take a very long time, the attacker explores the general structure of the cryptosystem, and summarize his findings in large tables which are not tied to particular keys. During the realtime phase, the attacker is provided with actual data produced based on a particular unknown key, and his goal is to use the precomputed tables in order to find the key as quickly as possible.

Accordingly, most of the Time/Memory/Data tradeoff attacks on stream ciphers are based on the following paradigm.

The attacker keeps a database of [state,output] pairs (sorted by output) and lookup every subsequence of the output stream in this database. When a (sufficiently long) database sequence is located in the output, the attacker can conclude that the actual state is the one stored along with this sequence and predict the rest of the stream. A drawback of this approach i that the large database must be stored in a hard disk(s) whose random access time is about a million times slower than a computational step. To improve that attack we can keep on disk only states that are guaranteed to produce outputs with some rare but easy recognizable property (e.g.,starting with some prefix). In this case only output sequences that have this property have to be searched in the database,and thus the expected time and the expected number of disk probes is significantly reduced. In general, producing a pair [state,output] with such a rare property costs much more than producing a random pair. $O(1/p)$ random states are required to find a single pair, where p is the probability of a random output to have this property. However, if we can efficiently enumerate states that produce such outputs, the number of sampled states decreases dramatically, and this method can be applied without significant additional cost during the preprocessing stage. The sampling resistance of a stream cipher provides a lower bound on the efficiency of such enumeration. Such an attack can be applied to RC4 in two ways, based on the KSA and PRGA parts. An attack on the generation part constructs a database of pairs [RC4 state,output substring] and analyzes all the substrings along a single output stream. The database construction is very simple since it is easy to enumerate states which produce outputs that have some constant prefix. However,this enumeration seems to be useless due to the huge effective key of this part (1684 bits) which makes such a tradeoff attack completely impractical. A more promising approach i based on the KSA part which uses a key of 40-256 bits and might be vulnerable to tradeoff attacks. In this case, the pairs in the database are [secret-key, prefix of the output stream], and the attack requires prefixes from a large number of streams (instead of a single long stream). The key-output correlation previously described provides an efficient sampling of keys that are more likely to produce output prefixes of the specified patterned type. For example, consider the problem of sampling M keys which are transformed by the KSA into streams whose first five words are fixed (mod 16). This property of random streams has probability of 2^{-20}, and the expected number of disk probes during the actual attack is reduced by this factor.

For stream ciphers with high sampling resistance, such a filter would increase the preprocessing time by a factor of one million, as one would have to sample a million random keys in order to find a single "good" key.

For RC4 (due to the invariance weakness), the preprocessing time increases by a factor of less than four, as more than one quarter of the exact special keys produce such streams, which have this fixed pattern. Consequently, the preprocessing stage is accelerated by a factor of 2^{18}. So, it is shown that RC4 has relatively low Sampling Resistance, which greatly improves the efficiency of tradeoff attacks based on its KSA.

Accordingly, main underlying ideas for developing the algorithm for cryptanalysis of RC4, assuming a known plaintext attack, can be summarized as the following:

- fact that certain prefixes of the output sequences are highly correlated with certain subsets of all possible session secret keys;
- developing a particular time-memory-data trade-off approach for reconstruction of the secret-keys which belongs to certain subsets of all possible secret keys;
- developing corresponding pre-processing and processing algorithms.

The allgorithm to be proposed is based on the following assumptions.

- A large collection of the PRGA output segments obtained for different session keys (i.e. the same secret key and different public randomization parameters) is available.
- Recovering of the secret key from the recovered session key is a separate problem which depends on the employed re-keying mechanism.

We assume the following notations:
- N: size of a reduced secret key space;
- D: size of available data;
- M: size of available memory;
- P: the pre-processing time complexity;
- T: the processing time complexity;
- R: the rate of certain pattern in the generator output sequence.

Finally, assume the following:

- $f(\cdot)$ is a function which performs input/output mapping in the same way as the keystream generator RC4 maps a secret key into the keystream output segment.

7.4 The Algorithm for Cryptanalysis

Next two section specify the pre-processing and processing phases of the algorithm developed for cryptanalysis of RC4.

7.4.1 Pre-Processing

In the pre-processing phase, the M-dimensional memory required for the time-memory-data trade-off, is initialized. This initialization should be done only once and the memory can be used for any secret key which belongs to certain subset of all possible secret keys which produces an output with certain prefix.

In a general case, the memory initialization requires computation of a number of the matrices like the matrix displayed in Fig. 1. Note that the memory can be initialized employing a stochastic approach with or without overlapping check which provide that all the elements of the underlying matrix are different. Usually, the overlapping check increases the initialization complexity, and the algorithm proposed bellow does not employ the overlapping check.

We assume that, in a general case, the complete memory of size M consists of q smaller memories each of which has size m. Also, we assume that $D \geq R^{-1}$ is length of the sample available for the processing. The complete memory initialization is based on appropriate employment of the following algorithm.

- *INPUT*: Algorithm parameters:
  - RC4 parameters: n and the session secret key length ℓ (in bits);
  - the binary pattern z, with $3n$ bit-values which correspond to the integers 0, 0 and $2^n - 1$;
  - the integers $m \leq M$ and $t \leq T$.

- *MEMORY INITIALIZATION*:
  For each of m memory locations, do the following:

  1. Set ℓ-bit vector $\mathbf{x}_1$ as the following:
     – concatenate the binary pattern z with a randomly selected $(\ell - 3n)$-bit pattern $\mathbf{r}$ different from all the previously selected ones.

2. generate $\mathbf{x}_2 = f(\mathbf{x}_1)$.
3. Generate the vector sequence $\mathbf{x}_i$, $i = 3, 4, , ..., t$, such that

$$\mathbf{x}_i = f(\hat{\mathbf{x}}_{i-1})$$

where $\hat{\mathbf{x}}_{i-1}$ is a modification of $\mathbf{x}_{i-1}$ obtained by presetting the first $3n$ bits to the binary pattern z.
4. Set as a memory location content the pair (*startpoint, endpoint*) $= (\mathbf{x}_1, \mathbf{x}_t)$.
5. Go to the step 1 if there is any non-initialized location of the memory; otherwise go to to the Output.

- *OUTPUT*: Initialized memory containing m pairs (*startpoint, endpoint*) = (*the initial state, corresponding output after t recalculations*), sorted in an increasing order of the endpoints.

Remark 1. When a stochastic initialization without the overlapping check is employed, two related important issues are the following ones: size of the underlying matrix, and characteristics of the matrix generating function $(\mathbf{x}_i = f(\hat{\mathbf{x}}_{i-1}))$.

Remark 2. According to [13] and [1], in order to make unlikely overlapping of the matrix elements, the following should hold: $mt^2 \leq N$. Also, recall that [13] has pointed out that the previous condition related to the random function model of the matrix generating function is a conservative one which in a number of cases underestimate possible values of the mt product: For example, when the generating function has only one cycle, than the starting and endpoints could be spaced $N^{1/2}$ apart and completely cover the key space with $m = t = N^{1/2}$. On the other hand, the results on the cycles structure of RC4 reported in [mister-tavares] imply that the relaxed condition on mt are more appropriate than the conservative one.

Remark 3. The function $f(\cdot)$ which maps an ℓ-bit keystream secret key into the ℓ-bit output segment can be modelled as a random function due to known characteristics of the keystream generator. On the other hand, the input/output pairs $(\mathbf{x}_{i-1}, \mathbf{x}_i)$ of the generating function, employed in the proposed algorithm for the memory initialization, are the segments of the keystream generator output sequences. Accordingly, $\hat{\mathbf{x}}_{i-1}$ into $\mathbf{x}_i$ can also be modelled as a random mapping relevant for consideration of the proposed memory initialization. The previous implies that it is very likely that q matrices each of size m $(qm = M)$ cover a fraction of the effective key space equal to $2^{\ell-3n}/(qDR)$ which is required for the time-memory trade-off.

7.4.2 Processing

The processing phase assumes reconstruction of the secret key based on a given output segment from RC4 keystream generator. The developed algorithm for the processing is the following one.

- *INPUT*:
  - RC4 parameters: parameter n and the session secret key length ℓ (in bits);

- the binary pattern z, with $3n$ bit-values which correspond to the integers 0, 0 and $2^n - 1$;
- the set of q memories initialized in the pre-processing;
- a set of D RC4 output sequences of length ℓ such that the prefix $\mathbf{c}$ appears with the rate $R = 2^{-2n}$ assuming that $\mathbf{c}$ is the all zero $2n$-bit pattern.
- the algorithm parameter t.

- *Recovering the Secret Key*:
  For each of the q memories, do the following:

  1. - In the given set of the generator output sequences find a previously unconsidered ℓ-bit segment $\mathbf{x}$ with the prefix $\mathbf{c}$.
     - If no one segment can be found go to the Output (b).
  2. - If the currently considered segment $\mathbf{x}$ is equal to one of the endpoints, go to the step 4.
     - Otherwise go to step 3.
  3. (a) Set the vector $\mathbf{x}_1$ to the considered segment $\mathbf{x}$, and set $i \rightarrow 2$.
     (b) Generate $\mathbf{x}_i$, as the following
     $$\mathbf{x}_i = f(\hat{\mathbf{x}}_{i-1}) \ ,$$
     where $\hat{\mathbf{x}}_{i-1}$ is a modification of $\mathbf{x}_{i-1}$ obtained by presetting the first $3n$ bits to the binary pattern z.
     (c) - If $\mathbf{x}_i$ is equal to the certain endpoint accept $\hat{\mathbf{x}}_{i-1}$ as the secret key and go to the Output (a).
     - Otherwise, set $i \rightarrow i + 1$ and,
       i. if $i < t$ go to the step 3b,
       ii. if $i = t$ go to the step 1.
  4. Set $\mathbf{x}_1$ to the corresponding startpoint, and generate the vector sequence $\mathbf{x}_j$, $j = 2, 3, ..., t$, such that $\mathbf{x}_J = \mathbf{x}$, and each $\mathbf{x}_j$ is generated as the following
     $$\mathbf{x}_j = f(\hat{\mathbf{x}}_{j-1}) \ ,$$
     where $\hat{\mathbf{x}}_{i-1}$ is a modification of $\mathbf{x}_{i-1}$ obtained by presetting the first $3n$ bits to the binary pattern z.
     Accept $\mathbf{x}_t$ as the secret key which generates the considered output segment $\mathbf{x}$, and go to the Output (a).

- *OUTPUT*:
  (a) Recovered session secret key.
  (b) Statement that the algorithm fails to recover session secret key.

7.5 Complexity of the Proposed Algorithm for Cryptanalysis

According to the algorithm for memory initialization (see Section 7.4.1), the following statement can be directly verified.

Proposition 1. The time complexity of pre-processing P is proportional to $qmt = Mt$.

According to the algorithm for the session secret key recovering (see Section 7.4.2), the following statement holds.

Proposition 2. The time complexity of processing T is proportional to $\frac{P}{M}DR$, and it yields the session secret key recovering with a high probability if
$PDR2^{-n}(1 - \frac{3}{2^n})^{-(2^n-3)} > 2^{(\ell-3)n}$.

Table 1 numerically illustrates main characteristics of the developed algorithm.

Table 2: Numerical illustrations of the proposed algorithm main characteristics: trade-offs between required data, required memory, pre-processing and processing complexity, when $R = 2^{-2n}$ and the number of submemories $q = 2^8$.

parameter n	dimension of the session key ℓ (in bits)	Data D	Memory M	Pre-process P	process-Time T
6	100	2^{44}	2^{32}	2^{66}	2^{61}
7	100	2^{44}	2^{32}	2^{63}	2^{59}
8	100	2^{44}	2^{32}	2^{60}	2^{56}
6	128	2^{52}	2^{40}	2^{86}	2^{81}
7	128	2^{52}	2^{40}	2^{83}	2^{79}
8	128	2^{52}	2^{40}	2^{80}	2^{76}

Note that the lengths of the session secret key are intentionally selected to have given in some cases "artificial" values in order to the comparison purposes. Usually, the length ℓ is a multiple of the word length n.

Part V: Concluding Discussions and References

8 A Discussion on Security of SSL Based on RC4

Curently, RC4 is used in a number of applications. One of its most important applications is in SSL, which is used to secure most of the worlds electronic commerce over the world wide web. It is also used in WEP, the IEEE 802.11 wireless networking security standard. It can also be found in a number of other applications including email encryption products.

All these protocols apply both symmetric authentication (MAC) and encryption of transmitted data.

Security consideration of a stream cipher in a complex cryptographic system which performs a number of functions in order to support privacy, integrity, and authenticity, for example, should take into account different aspects including the following:
(i) overall employment of the secret keys and cryptographic components;
(ii) method of combining the secret key and the public randomization parameter;
(iii) strength of the employed stream cipher.

Employment of RC4 in SSL should be considered from the above point of view, as well.

On Overall Employment of the Secret Keys and Cryptographic Components

In [16] the question of how to generically compose symmetric encryption and authentication when building "secure channels" for protection of communications over insecure networks is addressed. It has been shown that any channel protocol designed to work with any combination of secure encryption (against chosen plaintext attacks) and secure MAC must use the encrypt-then-authenticate method. It is demonstrated this by showing that the other common methods for composing encryption and authentication, including the authenticate-then-encrypt method used in SSL, are not generally secure. Particularly, [16], presents an example of an encryption function that provides perfect security but when combined with any MAC function under the authenticate-then-encrypt method yields a totally insecure protocol (for example, finding passwords or credit card numbers transmitted under protection of such protocol becomes an easy task for an active attacker). The same applies to the encrypt-and-authenticate method used in SSH.

It is shown in [16] that the athenticate-then-encrypt method as in SSL is not generically secure under the sole assumption that the encryption function is secure against chosen plaintext attacks and the MAC secure against chosen message attacks. A major issue to be highlight here is that the attack is not against the authenticity of information but against its secrecy. This result is particularly unfortunate in the case of SSL where protection of this form of sensitive information is one of the most common use of the protocol.

On the positive side, [16] shows that the authenticate-then-encrypt method is secure if the encryption method is a stream cipher that xor the data with a pseudorandom pad, implying that RC4 does not produce a weak protocol in the generic sense. Thus, while [16] shows the generic security of SSL to be broken, the current practical implementations of the protocol that use RC4 are safe from the considered generic point of view which assumes that RC4 is a secure stream cipher.

On Method for Combining Secret Key and Public Randomization Parameter

We consider employment of $RC4_8$ which is used for encryption based on the secret key and a per-packet 3 byte initial vector IV, and using the IV followed by the secret key as the RC4 key.

It is shown in [3] how dangerous could be employment of short public randomization parameters (24 bits) in context of RC4 encryption. It has been reported in [3] a number of flaws in the WEP algorithm. Recall that in WEP the secret key encryption is used to encrypt packets before they are transmitted.

Particularly, the following attack is reported in [3]. It is a dictionary-building attack that, after analysis of about a day's of traffic, allows real-time automated decryption of all traffic. This attack is based on the small space of possible initialization vectors so that an attacker is able to build a decryption table according to the following:
- once an attacker learns the plaintext for some packet, he can compute the RC4 key stream generated by the IV used;
- this keystream can be used to decrypt all other packets that use the same IV;
- over time, perhaps using the technique above, the attacker can build up a table of IVs and corresponding key streams;
- this table requires a fairly small amount of storage ($\approx 15GB$); once it is built, the attacker can decrypt **every** packet that is sent over the wireless link.

Note that the above dictionary building attack can be mounted against arbitrary long keystream sequences, assuming that appropriate memory size is available, i.e. it is not restricted on the attack against WEP only.

Another aspect of the considered problem of combining secret key and public randomization parameter is discussed in [8]. It is assumed that the attacker is able to retrieve the first byte of RC4 output from each packet because of the payload format used with 802.11, the first byte of each plaintext is a known constant, and hence the attacker is able to derive the first byte RC4 output. The attacking scenario is based on the following: To recover key byte B, the attacker needs to know the previous key bytes, and then search for IVs that sets up certain permutation.

Impact of the Encryption Scheme Weaknesses

This report has pointed out a number of undesirable properties of RC4. A natural question is: What is the impact of these characteristics on the practical security of SSL with RC4. But such a question, although natural one is not enough precise one in order to imply a precise answer. A more detail specification of a realistic (practical) attack is required.

On the other hand it should be pointed out that the attacks on RC4 usually require a large samples set for the processing, and collecting the required samples over the Internet should be considered as only a moderate problem.

Summary Estimation of SSL/RC4 Practical Security

The previous considerations imply a lot of challenges when RC4 is employed in SSL particularly due to the following reasons:
- the most powerful methods for attacking RC4 are based on collecting large samples and it appears not to be a big problem if we are working over the Internet whre SSL is widely deployed.

On the other hand, for example, RC4_n with $n = 8$ and the secret key length $\ell = 256$ bits is practicaly resistant against all currently known methods for the secret key recovering including the method proposed in this report.

9 Conclusions

This report originates from a collection of documents which includes all relevant currently known results related to RC4 security and it yields a security evaluation of RC4 based on the following:
- consideration of various known attacks against stream ciphers relevant for RC4;
- consideration of novel approach for RC4 attacking;
- consideration of practical security of RC4 within SSL.

Particularly, this report yields a proposal of an algorithm relevant for cryptaalysis of RC4. The developed algorithm is based on the time/memory/data tradeoff. It is the first in details specified algorithm for cryptanalysis of RC4 based on the time/memory/data tradeoff paradighm which takes into account a particular RC4 feature. Also, the algorithm main characteristics are discussed.

Security Evaluation Based on Various Known Attacks

Security evaluation of RC4 based on various known attacks is considered in Sections 4, 5 and 6 following the security evaluation guidelines given in Section 3.

RC4 is fully resistant on almost all reported general techniques for cryptanalysis of stream ciphers: For example, RC4 is resistant on general algebraic or fast correlation attacks which imply breakability of certain encryption algorithms (as an illustration of the performances of these techniques against some other ciphers see [25] and [24], as well as [23] which deals with cryptanalysis of a scheme based similarlly as RC4 on a time-variant table).

The main exception is the time/memory/data tradeoff method [1]. More precisely, it seems that this method is a candidate for developing the most powerful attacking techniques and a proposal towards this direction is given in this report.

This report has re-considered various security evaluation approaches relevant for RC4. The following issues were discussed:

- weak and related keys;
- weaknesses of the key scheduling algorithm and recovering secret keys;
- distinguishing RC4 outputs from the random ones;
- prediction of RC4 internal states or outputs;
- recovering RC4 internal state.

Particularly, note the following.
(i) A weakness of RC4 exists which results from the simplicity of its key scheduling algorithm appears to imply a serious vulnerability. It is recommend to neutralize this weakness by discarding the first $N = 2^n$ words (n is number of bits in a word) of each generated stream. This recommendation is motivated by the fact that after N rounds, every element of S is swapped at least once and the permutation S and the index j are expected to be independent of the initialization process.
(b) A weakness of RC4 in a common mode of operation in which attacker visible IV's are concatenated with a fixed secret key is also pointed out. Recall that it is easy to extend the attack to other simple types of combination operators (e.g., when we XOR the IV and the fixed key) with essentially the same complexity. It is recommend to neutralize this weakness by avoiding this mode of operation, or by using a secure hash to form the key presented to the KSA from the IV and secret key.

Novel Approach for RC4 Cryptanalysis

In section 7, this report yields a framework for mounting the attacks for the secret key recovering based on a correlation between certain session keys and the output sequence. This correlation which is a particular consequence of RC4 low sampling resistance yields a possibility for recovering a session key which belongs to a reduced class of all possible session keys employing a time/memory/data trade-off approach developed to employ the detected correlation.

Practical Security of SSL/RC4

Practical security of SSL/RC4 is discussed in Section 8. The weaknesses of RC4 reported in this report imply a lot of challenges when RC4 is employed in SSL particularly due to the following reasons:
- the most powerful methods for attacking RC4 are based on collecting large samples and it appears not to be a big problem if we are working over the Internet whre SSL is widely deployed.

It is evident that the encryption schemes, members of RC4 family are far from perfect ones, and that certain of these schemes are completely breakable (assuming relatively small n or the secret key length ℓ).

The results due to [19], [8] and [26] as well as the algorithm for cryptanalysis proposed in this report are of the most practical importance.

[19] describes the bias in the second byte of RC4, which is zero with probability twice what it is expected.

[8] presents an analysis of a broad class of weak keys based on some parity-preserving properties in key scheduling.

[26] yields a conservative estimate for the number of bytes that should be discarded in order to be safe, and it is is based on the distinguisher that detects a bias in the first byte.

Note that RC4 has been **not** selected as a final candidate for the NESSIE Project recommended stream cipher, although it is de facto existing standard over the Internet. The following main reasons have been given in [29] for this decision. The second output byte of RC4 can be easily distinguished from a random one, [19]. Although this can be easily overcome, the keystream still can be distinguished from a random one [7], and the key schedule has severe weaknesses [8]. Also, as a negative characteristic it is claimed that no form of rekeying is defined for RC4.

This report has summarized even more weaknesses or potential weaknesses of RC4 than it has been taken into account in the NESSIE Report, including a novel method for cryptanalysis based on a particular session key - the first two output bytes correlation and employment of the time/memory/data tradeoff concept.

On the other hand, for example, RC4$_n$ with $n = 8$ and the secret key length $\ell = 256$ bits is practically resistant against all currently known methods for the feasible secret key recovering including the method proposed in this report.

All-after-all, most of the members of RC4 family can still be considered as practically secure if it is used a hash function to form session keys from secret keys and IV's, and discard the first 2^{n+1} words of output before use.

On the other hand, note that this report summarizes a number of, at least very undesirable and potentialy dangerous characteristics of RC4, and this is also a reason more why further research towards RC4 keystream generator security is very recommendable.

References

[1] A. Biryukov and A. Shamir, "Cryptanalytic time/memory/data tradeoffs for stream ciphers", ASIACRYPT2000, *Lecture Notes in Computer Science*, vol. 1976, pp. 1-13, 2000.

[2] A. Biryukov, A. Shamir and D. Wagner, "Real time cryptanalysis of A5/1 on a PC", FSE 2000, *Lecture Notes in Computer Science*, vol. 1978, pp. 1-18, 2001.

[3] N. Borisov, I. Goldberg and D. Wagner, "(In)Security of the WEP algorithm", 2001, htpp://www.isaac.cs.berkeley.edu/isaac/wep-faq.html.

[4] "CRYPTREC Report 2000", Information Technology Promotion Agency (IPA), Japan, August 2001,
http://www.ipa.go.jp/security/index-e.html

[5] G. Durfee, "Distinguishers for the RC4 stream cipher", unpublished manuscript, 2001.

[6] H. Finney, "An RC4 cycle that can't happen", Post in sci.crypt, message-id 35hq1u$c72@news1.shell, 18 Sept. 1994.

[7] S. Fluhrer and D. McGrew, "Statistical analysis of the alleged RC4 key stream generator", FSE 2000, *Lecture Notes in Computer Science*, vol. 1978, pp. 19-30, 2001.

[8] S. Fluhrer, I. Mantin and A. Shamir, "Weaknesses in the key scheduling algorithm of RC4", SAC 2001, *Lecture Notes in Computer Science*, vol. 2259, pp. 1-24, 2001.

[9] J.Dj. Golić and M.J. Mihaljević, "Minimal linear equivalent analysis of a variable-memory binary sequence generator", *IEEE Transactions on Information Theory*, vol. 36, pp. 190-192, 1990.

[10] J.Dj. Golić, "Linear statistical weakness of alleged RC4 keystream generator", EUROCRYPT'97, *Lecture Notes in Computer Science*, vol. 1233, pp. 226-238, 1997.

[11] J.Dj. Golić, "Linear models for a time-variant permutation generator", *IEEE Transactions on Information Theory*, vol. 45, pp. 2374-2382, Nov. 1999.

[12] A.L. Grosul and D.S. Wallach, "A related-key cryptanalysis of RC4", Technical Report, Rice University, TR00-358, June 2000,
http://www.wisdom.weizmann.ac.il/ itsik/RC4/Papers/GrosulWallach.ps.

[13] M.E. Hellman, "A cryptanalytic time-memory trade-off", *IEEE Transactions on Information Theory*, vol. IT-26, pp. 401-406, July 1980.

[14] R.J. Jenkins, "ISAAC and RC4", 1996,
http://burtleburtle.net/bob/rand/isac.html.

[15] L. Knudsen, W. Meier, B. Preneel, V. Rijmen and S. Verdoolaege, "Analysis method for (alleged) RC4", ASIACRYPT'98, *Lecture Notes in Computer Science*, vol. 1514, pp. 327-341, 1998.

[16] H. Krawczyk, "The order of encryption and authentication for protecting communications (or: How secure is SSL)", CRYPTO2001, *Lecture Notes in Computer Science*, vol. 2139, pp. 310-331, 2001.

[17] P.D. Kundarewich, S.J.E. Wilton and A.J. Hu, "A CPLD-based RC-4 cracking system", *1999 IEEE Canadian Conf. on Electrical and Computer Eng.*, Edmonton, Alberta, May 1999, Proceedings, pp. 397-401.

[18] I. Mantin, *Analysis of the stream cipher RC4*. Masters's Thesis, Weizmann Institute, Israel, 2001.

[19] I. Mantin and A. Shamir, "A practical attack on broadcast RC4", FSE 2001, *Lecture Notes in Computer Science*, vol. 2355, pp. xxx-yyy, 2002.

[20] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton: CRC Press, 1997.

[21] S. Mister and S. Tavares, "Cryptanalysis of RC4-like ciphers", SAC '98, *Lecture Notes in Computer Science*, vol. 1556, pp. 131-143, 1999.

[22] M.J. Mihaljević, "A correlation attack on the binary sequence generators with time-varying output function", AUSCRYPT'94, *Lecture Notes in Computer Science*, vol. 917, pp. 67-79, 1995.

[23] M.J. Mihaljević, "A family of fast keystream generators based on programmable linear cellular automata over GF(q) and time-variant table", *IEICE Transactions on Fundamentals*, vol. E83-A, pp. 32-39, 1999.

[24] M.J. Mihaljević, M.P.C. Fossorier and H. Imai, "Fast correlation attack with the list decoding and an application", FSE2001, *Lecture Notes in Computer Science*, vol. 2355, pp. 196-210, 2002.

[25] M.J. Mihaljević and R. Kohno, "Cryptanalysis of fast encryption algorithm for multimedia", *IEEE Communications Letters*, vol. 6, Sept. 2002.

[26] I. Mironov, "(Not so) random shuffles of RC4", CRYPTO2002, to appear in *Lecture Notes in Computer Science*, 2002.

[27] "New European Schemes for Signatures, Integrity and Encryption (NESSIE) Project",
http://www.cryptonessie.org.

[28] "Description of Methodology for Security Evaluation", NESSIE Public Report, NES/DOC/RHU/WP3/D10/3, 2000,
http://www.cryptonessie.org.

[29] "Update on the selection of algorithms for further investigation during the second round", NESSIE Public Report, NES/DOC/ENS/WP5/D18/1, 2002,
http://www.cryptonessie.org.

[30] R. Rivest, "The RC4 encryption algorithm", RSA Data Security, Inc, Mar. 1992.

[31] R. Rivest, "RSA security response to weaknesses in key scheduling algorithm of RC4",
http://www.rsasecurity.com/rsalabs/technotes/wep.html

[32] RSA Laboratories FAQ, Question 3.6.3.
http://www.rsasecurity.com/rsalabs/faq/3-6-3.html.

[33] A.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.

[34] A. Stubblefield, J. Ioannidis, and A.D. Rubin, "Using the Fluhrer, Mantin and Shamir attack to break WEP", AT&T Labs, Technical Report, TD-4ZCPZZ, August 6, 2001 (8 pages).

[35] B. Schneier, *Applied Cryptography*, Second edn. New York: John Wiley and Sons, 1996.

[36] R. Wash, "Lecture notes on stream ciphers and RC4", unpublished manuscript.

[37] D. Wagner, "My RC4 weak keys", Post in sci-crypt, message-id 4470il$cbj@cnn.princeton.edu, 26 Sept. 1995.