A Practical Attack on the Fixed RC4 in the WEP Mode

I. Mantin

NDS Technologies, Israel
imantin@nds.com

Abstract. In this paper we revisit a known but ignored weakness of the RC4 keystream generator, where secret state info leaks to the generated keystream, and show that this leakage, also known as Jenkins' correlation or the RC4 glimpse, can be used to attack RC4 in several modes. Our main result is a practical key recovery attack on RC4 when an IV modifier is concatenated to the beginning of a secret root key to generate a session key. As opposed to the WEP attack from [FMS01] the new attack is applicable even in the case where the first 256 bytes of the keystream are thrown and its complexity grows only linearly with the length of the key. In an exemplifying parameter setting the attack recovers a 16-byte key in 2^{48} steps using 2^{17} short keystreams generated from different chosen IVs. A second attacked mode is when the IV succeeds the secret root key. We mount a key recovery attack that recovers the secret root key by analyzing a single word from 2^{22} keystreams generated from different IVs, improving the attack from [FMS01] on this mode. A third result is an attack on RC4 that is applicable when the attacker can inject faults to the execution of RC4. The attacker derives the internal state and the secret key by analyzing 2^{14} faulted keystreams generated from this key.

Keywords: RC4, Stream ciphers, Cryptanalysis, Fault analysis, Side-channel attacks, Related IV attacks, Related key attacks.

1 Introduction

RC4 is the most widely used stream cipher in software applications. Among numerous applications it is used to protect Internet traffic as part of the SSL and is integrated into Microsoft Windows. It was designed by Ron Rivest in 1987 and kept as a trade secret until it leaked out in 1994. RC4 has a secret internal state which is a permutation of all the $N = 2^n$ possible n bits words, associated with two indices in it, when in practical applications $n = 8$, and thus RC4 has a huge state of $log_2(2^8! \times (2^8)^2) \approx 1700$ bits.

In this paper we revisit a known but previously ignored property of RC4, which we denote as the *Glimpse property* also known as Jenkins' correlations. The glimpse is a leakage of information from RC4 secret state to the generated keystream, where every keystream word hints on a state word through the correlation $S[j] = i - z$ which occurs with doubled probability (1/128 instead of

B. Roy (Ed.): ASIACRYPT 2005, LNCS 3788, pp. 395–411, 2005.

$1/256$), when i is a known index of RC4 state, z is the hinting keystream word and $S[j]$ is the hinted entry of the secret internal state.

The glimpse property was first mentioned in the web page of Jenkins ([Jen96]) and was first brought to formal literature in [MS01] in 2001. In Chapter 7 of [Man01] Mantin analyzed the glimpse property, defined a generalized version of the correlation and discovered small biases in the keystream that stem from it. However, due to the fact that the glimpse discovers a negligible part of the internal state (one byte out of 1700) and the fact that it does so with biased but still small probability, that was the last trial for exploiting this property to attack RC4.

In this paper we revisit the glimpse in RC4 and RC4-like stream ciphers, analyze its origin and discuss the ways a cryptanalyst can use it. We define a generalized version of the glimpse and discuss the availability of the generalized correlations in RC4 and RC4-like ciphers.

Our main result is a practical key recovery attack on RC4 that works even when the common recommendation of throwing a 256-byte prefix of the keystream is adopted. The attack works in a mode of operation where an initial value (IV) is concatenated to the beginning of the root key and works in both the chosen IV and known IV models. The attack allows some data-time tradeoff that depends on the length of the root key. For example, some parameter setting for a 16-byte key allows the attacker to recover the key in 2^{48} steps using 2^{17} data or with 2^{32} steps using 2^{20} data. In the known (random) IV model the data complexity of the attack requires an additional multiplicative factor of $N = 256$ in order to have a sufficient number of "good" IVs.

In the second part of the work we present the *fork* model where many instances of RC4 are available to the attacker with almost equal state and show that in this model an attacker can use the glimpse property to recover RC4 internal state. We show two realizations of this model; the first is where the IV modifier is concatenated to a end of a secret root key in order to generate many independent RC4 keystreams from a single secret root key. In this mode we mount a chosen and known (random) IV attacks that recover the secret key by analyzing 2^{22} keystreams that were generated from this key and different IVs. Another realization of this model is where the attacker injects faults into the execution of RC4 and distorts the generated keystream. In that case we mount a fault attack that uses 2^{14} faulty keystreams to recover the internal state and the secret key.

The rest of the paper is organized in the following way: In Sect. 2 we describe RC4 and previous cryptanalysis. In Sect. 3 we re-present the glimpse property and analyze its origin and availability. In Sect. 4 we describe key recovery attacks on RC4 in the preceding IV mode when the first 256 bytes are thrown. In Sect. 5 we present the *Fork* model and use the glimpse property to mount an attack on RC4 in this model. In Sect. 6 we adjust the fork model attack to mount a key recovery attack on the succeeding IV mode. In Sect. 7 we adjust the fork model attack to mount an efficient fault attack on RC4. We summarize our work in Sect. 8.

KSA($K[0 \dots \ell - 1]$)
Initialization:
  For $i = 0 \dots N - 1$
    $S[i] = i$
  $j = 0$
Scrambling:
  For $i = 0 \dots N - 1$
    $j = j + S[i] + K[i \bmod \ell]$
    $Swap(S[i], S[j])$

PRGA(K)
Initialization:
  $i = 0$
  $j = 0$
  $S = KSA(K)$
Generation loop:
  $i = i + 1$
  $j = j + S[i]$
  Swap($S[i], S[j]$)
  Output $z = S[S[i] + S[j]]$

Fig. 1. The Key Scheduling Algorithm and the Pseudo-Random Generation Algorithm

2 RC4 and Its Security

2.1 Description of RC4

RC4 consists of 2 parts (described in Fig. 1): A key scheduling algorithm KSA which turns a variable-size key (with typical size of 5-32 bytes) into an initial permutation S of $\{0, \dots, N-1\}$, and an output generation part PRGA which uses this permutation to generate a pseudo-random keystream.

The PRGA initializes two indices i and j to 0, and then loops over four simple operations which increment i as a counter, increment j pseudo randomly, exchange the two values of S pointed to by i and j, and output the value of S pointed to by $S[i] + S[j]$[1].

2.2 Previous Analysis of RC4

Cryptanalysis of RC4 is divided into two main parts, analysis of the initialization of RC4 and analysis of the keystream generation. The first part focuses on the KSA, the PRGA initialization and the integration of both, whereas the last focuses on the internal state and the round operation of the PRGA.

Due to the simplicity of the initialization part and the major difference between the typical key sizes and the effective size of RC4 state, this part was subject to extensive analysis and indeed numerous significant weaknesses were discovered of many types, including classes of weak keys ([Roo95]), patterns that appear twice and three times the expected probability (the second byte bias [MS01]), propagation of key patterns through the KSA to the initial permutation and through the PRGA initialization to the prefix of the stream (the invariance weakness [FMS01]), related key attacks ([GW00]), statistical biases in different prefixes of the generated stream ([FMS01] and [PP04]) and analysis of the biased distribution of RC4 initial permutation ([Mir02] and [Man01]). However, the most devastating attack on RC4 was described in [FMS01] where RC4 was proved to have serious related-key vulnerabilities, exposing several implementations of RC4 to practical key recovery attacks, where the effected implementations are those that employ trivial key-IV combination methods such as

[1] Here and in the rest of the paper all the additions are carried out modulo N.

concatenation or exclusive-or. A subsequent work by Stubblefield et-al ([SIR01]) implemented the attack on the security protocols of the international standard for wireless LAN communication 802.11*b* (WEP) that used RC4 in the IV concatenation mode, and these protocols were declared as broken.

This attack had a great impact on the trust of cryptographers and security designers in RC4 and the common practice for using RC4 today includes hardening of the initialization process by omitting some prefix of the keystream, usually 256 bytes as recommended by RSA laboratories in [RSA01]. This hardening neutralizes most of the attacks and weaknesses that were discovered in RC4 initialization. However, this mode still has some weaknesses, including a biased distribution of the PRGA initial permutation ([Mir02]) and statistical biases in the first bytes that are emitted after the 256^{th} round ([PP04]).

Statistical analysis of the keystream generation part gave rise to several weaknesses and biased patterns in RC4 keystreams. Golić ([Gol97]) and Fluhrer and McGrew ([FM00]) designed distinguishers of RC4 streams from random streams that require $2^{44.7}$ and $2^{30.6}$ keystream words respectively. Subsequently Mantin improved these results in [Man05] and designed a 2^{28} distinguisher. In his paper he also described several families of patterns denoted in [Man05] as recyclable patterns, which occur in RC4 keystreams with extremely high probability that is several times the probability in random streams, and described an algorithm that uses these patterns to predict in some rare cases bits and full bytes of RC4 with success probabilities that are close to 1.

Several other classes of RC4 partial states were defined and analyzed in [FM00], [MS01] and [PP03] as such that create unique patterns in the output stream and allow a viewer of the output stream to recover parts of the internal state with more than trivial probability (chapter 2 of [Man01] contains an overview of these classes). The cycles structure of RC4 state progression was also analyzed in [MT98] and [Fin94], where the last describes short cycles that are unreachable by RC4. [KM+98], [MT98] and [Gol00] describe state recovery attacks through backtracking with complexity that is less than the square root of an exhaustive search over all possible states. However, due to the hugeness of the state (1700 bits for $n = 8$), these attacks are completely impractical as they require more than 2^{700} steps. Mantin in [Man05] describes an approach that under some circumstances can improve this attack significantly through using the recyclable patterns.

Two variants of RC4 were recently proposed, both slightly more complex than the original RC4 and are claimed to be more secure than it. RC4A ([PP04]) was designed by Paul and Preneel and works with two RC4 tables. The generation stage of RC4A is slightly more efficient than RC4's, but the initialization stage requires at least twice the effort of RC4 initialization. VMPC ([Zol04]) was designed by Zoltak and includes several changes to the KSA, the IV integration method, the PRGA initialization, the round operation and the output selection method. Maximov described in [Max05] distinguishers for both variants, requiring 2^{54} data for VMPC and 2^{58} data for RC4A and Tsunoo et-al subsequently described in [TS+05] a prefix distinguishers for VMPC and RC4A keystream

generators, requiring 2^{23} keystream prefixes for RC4A and 2^{24} keystream prefixes for VMPC. A regular distinguisher (as opposed to a prefix distinguisher) of RC4A was shown in [Man05] that needs a keystream of 2^{29} keywords and is an adjusted variant of the RC4 distinguisher mentioned in this work.

The trend of side-channel attacks had not skipped RC4. Hoch and Shamir made in [HS04] an exhaustive fault analysis of many stream ciphers including RC4 and found them all vulnerable to key recovery attack in this model. In particular their attack on RC4 requires 2^{16} faults. Biham et-al proposed in [BGN05] two other fault attacks on RC4; in the impossible fault attack is based on using faults to force the cipher to enter the impossible states known as Finney's states ([Fin94]). In the differential fault attack, the attacker compares many faulty keystreams to a non-faulty keystream and identifies the three permutation entries that are used in the first round, the second round, etc. Several variants and optimizations for this attack are described and the best configuration of the attack requires 2^{10} faults and key resets.

2.3 Notations

In vast majority of RC4 implementations $N = 256$ and $n = 8$. In many cases we simplify expressions by using numbers instead of parameters. Whenever appropriate, we mention this conversion.

For a positive integer X we use the notation $[X]$ to specify the domain of indices modulo X, i.e., $[X] = \{0, 1, \ldots, X-1\}$. We denote the domain of permutations of $[X]$ as $\mathcal{P}[X]$.

We use the notations i_t, j_t and S_t for the indices i and j and the permutation S after round t, where the rounds are indexed in accordance with i, i.e., $i_t = t$. Thus the KSA has rounds $0, \ldots, 255$ and the PRGA has rounds $1, 2, \ldots$. We use the same indexing for both the KSA permutations and the PRGA permutations and whenever there might be a confusion, we use the notations $S^{(KSA)}$ and $S^{(PRGA)}$ respectively.

The output function $Z : \mathcal{P}[X] \times [X] \times [X] \to [X]$ is defined on RC4 states as $Z(S, i, j) \stackrel{def}{=} S[S[i] + S[j]]$. We denote output words with z and index them in the same manner as i and j, i.e., $z_t = Z(S_t, i_t, j_t)$.
We denote the KSA key as K and its length as ℓ_K.

3 The Glimpse

The glimpse property as was first introduced in [Jen96] is defined in Theorem 1.

Theorem 1 (The Glimpse Main Theorem). *Let $i \in [N]$. Then*

$$\mathbb{P}_{j \in_R [N], S \in_R \mathcal{P}[N]}[S[j] = i - Z(S, i, j)] \approx 2/N \tag{1}$$

$$\mathbb{P}_{j \in_R [N], S \in_R \mathcal{P}[N]}[S[i] = j - Z(S, i, j)] \approx 2/N \tag{2}$$

In other words, when z is the output then

$$\mathbb{P}[S[j] = i - z] \approx \mathbb{P}[S[i] = j - z] \approx 2/N$$

The proof of Theorem 1 appears in the discussion of useful states in Sect. 2.3 of [Man01], and we only bring the intuition behind one of them (the second stems from symmetry). In the case where $i = S[i] + S[j]$, the correlation occurs with probability 1 since

$$Z(S, i, j) = S[S[i] + S[j]] = S[i] = i - S[j] \tag{3}$$

In the other case ($i \neq S[i] + S[j]$), the correlation occurs with a probability of $1/N$ and thus the overall probability is

$$1/N \cdot 1 + (1 - 1/N) \cdot 1/N \approx 2/N \tag{4}$$

A generalized version of the glimpse was proved in Sect. 7 of [Man01], where different relations between i and z hint on corresponding relations between $S[i]$ and $S[j]$. This generalization is given in Theorem 2.

Theorem 2. *Let f be a $[N] \rightarrow [N]$ function and let $h_f(x) \stackrel{def}{=} f(x)+x$. Suppose that h_f is on-to-one in the domain $[N]$ and onto $[N]$. Then for every $i \in [N]$,*

$$\mathbb{P}_{j \in_R [N], S \in_R \mathcal{P}[N]}[S[j] = f(S[i]) | i = h_f(Z(S, i, j))] \approx 2/N \tag{5}$$

The original glimpse is a special case with the degenerated function $f(x) \stackrel{def}{=} i - z$ and $h_f(z) = i$. The base condition $i = h_f(Z(S, i, j))$ occurs always and thus the probability of the derived condition $[S[j] = f(S[i])$ is always $2/N$. Thus many relations between the index i and the output word z imply corresponding relations between the permutation entries that are used.

In Sect. A of the appendix we discuss the availability of the glimpse and show that it exists in many other output selection functions. Notice that since the index j is secret, the output hints on a value in an unknown location. However, the value in this location was in a known location i immediately before this round and furthermore, this is the same value that was used to update j in this round. These facts underline the analysis in the rest of the paper.

4 Attacking the Truncated RC4

One of the most popular IV combination methods for RC4 and other stream ciphers is a concatenation of the IV to the root key in order to obtain a one-time session key. This mode of operation was attacked by Fluhrer et-al in [FMS01] both in the case where the IV is concatenated to the end of the root key (we denote this mode as the succeeding IV mode) and in the case where the IV is concatenated to the beginning of the root key (we denote this mode as the preceding IV mode). Their attack on the preceding IV mode was found applicable to the RC4 implementation in WEP and it is sometimes referred to as the WEP attack or the FMS attack. The attack recovers the bytes of the root key one at a time, where in the iterative step IVs are selected that cause leakage of information from the target keyword into the first word of the generated

keystream[2]. Since the publication of this attack in 2001 the common practice in implementations of RC4 is to throw a prefix of 256 bytes from the keystream and thus prevent access of the attacker to the first output word and foil the attack. We denote this usage mode of RC4 as *the truncated RC4*.

In this section we present a new attack on the truncated RC4. The attack resembles the FMS attack in many aspects, where instead of using the leakage of the keyword to the first output word, we use the glimpse property to redirect the leakage to the 257^{th} keystream word and thus overcome the omission of the first keystream words.

The rest of this section is organized as follows. We first describe the WEP attack and the way in which particular keywords leak to the first keystream word. Afterwards we present a new leakage scenario where keyword info leaks to the 257^{th} keystream word. We describe how the attack uses this leakage scenario to recover the root key and end with complexity analysis and a comparison to the WEP attack.

4.1 The WEP Attack

We denote the root key as RK and the session key that is combined from RK and an IV as SK. We denote the length of these keys by $|RK|$ and $|SK|$ respectively and the length of the IV by ℓ_{IV} and thus $|SK| = |RK| + \ell_{IV}$.

The attack recovers the keywords one at a time. The iterative step of the attack assumes knowledge of some prefix of the RC4 keywords, and uses the first word of each of several keystreams to derive the next keyword (which we denote below as the target keyword). The attack starts with the known IV as a basis, and repeatedly applies the iterative step in order to recover all the keywords in the root key. The keystreams from which first words are taken to recover the target keyword, are carefully selected according to the IV that was used to generate them.

The Iterative Step. The iterative step for the $(x+1)^{th}$ keyword $SK[x]$ (which is $RK[x - \ell_{IV}]$) as the target keyword simulates the first x steps of the KSA using the x known keywords in order to recover the permutation after x rounds S_{x-1}. The KSA uses the next keyword, which is the target keyword, to calculate the value of j in the next step j_x and thus this keyword can be easily derived from j_x. Since the swap in round x occurs in locations $i_x = x$ and j_x, $S_x[x] = S_{x-1}[j_x]$ and knowing S_{x-1}, $S_x[x]$ leads to j_x and further on to the target keyword.

The first output value is $S[S[1] + S[S[1]]]$ and thus only three permutation entries are used for this calculation; the ones in locations 1, $S[1]$ and $S[1] + S[S[1]]$. When these locations are lower than or equal to x after round x they are guaranteed not to be visited by i during the subsequent $N - x$ rounds and

[2] The attack is therefore a Known Plaintext Attack (KPA). Stubblefield et-al subsequently showed that the first plaintext byte in typical WEP applications is a constant header and thus the KPA model is realistic.

with high probability of more than $e^{-3} \approx 5\%$ (using Lemma 1, which is proved in Appendix B) no to be visited by the pseudo-random index j during these rounds. In that case, the first keystream word can be deduced with high probability from the values that are at these locations in the permutation S_x. Furthermore, when in addition $S[1] + S[S[1]] = x$ (in [FMS01] it was denoted as the resolved condition) the first keystream word z_1 is exactly $S_x[x]$, from which the target keyword can be derived.

Formally, when $1, S[1] < x$ and $S[1] + S[S[1]] = x$, then with probability of at least 5%

$$\begin{aligned} SK[x] = & \quad j_x - j_{x-1} - S_{x-1}[x] \quad = \\ & = S_{x-1}^{-1}[S_x[x]] - j_{x-1} - S_{x-1}[x] \overset{w.p.\ 1/e^3}{=} \\ & = \quad S_{x-1}^{-1}[z_1] - j_{x-1} - S_{x-1}[x] \end{aligned} \tag{6}$$

Thus when IVs are selected that cause S_{x-1} to satisfy the resolved condition, the above calculation points to the correct target keyword with probability of about 5%. Using this observation Fluhrer et-al recovered the target keyword through employing a simple voting mechanism where every first keystream word gives a vote to a keyword candidate.

Analysis of the WEP Attack. In [FMS01] it was estimated that in order to mount the attack for a particular keyword the attacker needs about 60 votes from which an average number of three votes go to the correct target keyword. In order to guarantee the 5% probability, these votes must come from situations where the resolved condition was satisfied and thus in the chosen IV model the number of IVs that are needed is 60 per keyword.

In the known IV model the situation is more complicated where the attacker must wait for IVs that lead to the resolved condition, which under reasonable randomness assumptions have a fraction of $\frac{x}{N^2}$ and thus the $(x+1)^{th}$ keyword requires $60N^2/x$ IVs. Since the data can be reused for different iterative steps the main complexity parameter for the attack is the maximal number of IVs for a keyword, which is $60 \cdot N^2/\ell_{IV}$. A somewhat surprising result is that the attack works better when longer IVs are used.

4.2 The New Attack

We present a similar leakage from the target keyword in two stages, to $S_1^{(PRGA)}[1]$ and through it to the 257^{th} keystream word z_{257}.

We first describe the way $S_x[x]$ reaches location 1 of S. Suppose that after round $x-1$ of the KSA we have $S_{x-1}[1] = x$. In the next round some arbitrary value Y, pointed to by j, is swapped into location x. This Y leads to the target keyword in the same manner as in the WEP attack (known S_{x-1}, Y leads to j_x, j_x leads to $SK[x]$). Suppose that during the remaining $N - x$ KSA rounds the values x and Y remain at locations 1 and x. The probability of this event is at least $1/e^2$ (using Lemma 1). In the first round of the PRGA round we get

$$i_1 = 1 \tag{7}$$

$$j_1 = j_0 + S_0^{(PRGA)}[1] = S_x^{(KSA)}[1] = x \tag{8}$$

$$S_1^{(PRGA)}[1] = S_0^{(PRGA)}[j_1] = S_x^{(KSA)}[x] = Y \tag{9}$$

Thus with probability $1/e^2$ target keyword info leaks into $S_1^{(PRGA)}[1]$. In the next 255 rounds of the PRGA i traverses locations $2, \ldots, 255, 0$ and with probability $1/e$ (again we use Lemma 1) the index j also skips location 1 and then $S_{256}^{(PRGA)}[1] = S_1^{(PRGA)}[1] = Y$. However, the glimpse property causes information on this particular byte to leak to the next keystream word and thus we complete a leakage chain from the target keyword to z_{257}.

Combining these observations with the glimpse probabilities we get (probabilities are presented over the equality sign)

$$i_{257} - z_{257} \overset{2/N}{=} S_{256}^{(PRGA)}[1] \overset{1/e}{=} S_1^{(PRGA)}[1] \overset{1/e^2}{=} Y \tag{10}$$

Thus we reach a probability of $2e^{-3}/N$ for the complete scenario to occur and in this case the correct target keyword is

$$SK[x] = S_{x-1}^{-1}[i_{257} - z_{257}] - j_{x-1} - S_{x-1}[x] \tag{11}$$

Notice that when the chain breaks, there is still a probability of $1/N$ to have a lucky guess and thus the overall probability for a successful guess is

$$\begin{aligned} \mathbb{P}[\, SK[x] = S_{x-1}^{-1}[i_{257} - z_{257}] - j_{x-1} - S_{x-1}[x]] \approx \\ \approx \quad \frac{2}{e^3 N} \cdot 1 + (1 - \frac{2}{e^3 N}) \cdot \frac{1}{N} = \\ = \quad \frac{1}{N} + \frac{2}{e^3 N} \cdot (1 - \frac{1}{N}) \approx \\ \approx \quad \frac{1}{N} \cdot (1 + \frac{2}{e^3}) \approx 1.1 \cdot \frac{1}{N} \end{aligned} \tag{12}$$

Simulations we carried out show that this analysis is somewhat optimistic and that the actual probability for a correct guess (given that the IV conditions are satisfied) is $1.075 \cdot 1/N$.

4.3 Complexity Analysis

Next we compare the attack parameters and probabilities to those of the WEP attack. The probability of having a "good" IV *increases* from $\frac{x}{N^2}$ in the WEP attack to $1/N$ (need only $S_{x-1}[1] = x$). However, the advantage in the voting process significantly decreases from 5% to 1.075/256. Thus the voting in this attack is much harder than in the WEP attack, even though a larger fraction of the IVs are "good" and this voting requires almost one million "good" IVs (see Fig. 2) for recovering the target keyword with a probability that is close to 1.

However, a smarter key recovery algorithm can tolerate some errors in the guessing. The algorithm can guess C possible values for every keyword and check all the

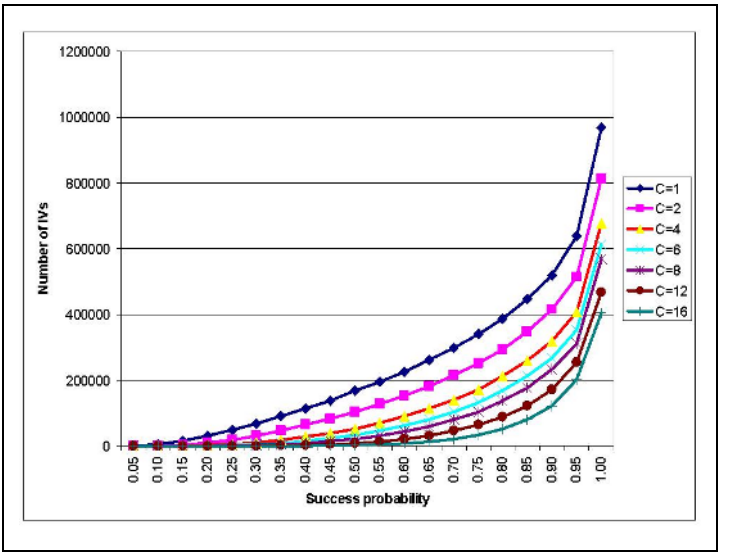

Fig. 2. The number of IVs that are required for different success probabilities (for the attack on the Truncated RC4). The different graphs are for different selections of the branching factor C.

possible C^{ℓ_K} branches, where a typical value for C is 4-5. Typical RC4 keys are 16 bytes or below, which makes the number of possibilities checked by the algorithm no more than $5^{16} \approx 2^{37}$. This attack can be further optimized by using a smart branching strategy that instead of using a fixed branching factor, selects the number of branches according to the result of the voting, e.g., avoiding branching when a single value sticks out clearly as the correct keyword. However, in this extended abstract we limit the discussion to the simple case of a fixed branching factor.

In Fig. 2 we show the number of samples of "good" IVs that are required for different success probabilities and different selections of the branching factor C. For example, for $C = 8$ the attack requires a practical amount of 2^{17} "good" IVs in order to get a success probability of 80% for recovery of every keyword. The selection of C depends heavily on the key length, where large C's can be used only when the key is short. For example, for a 16-byte key, using $C = 8$ implies a time complexity of $8^{16} = 2^{48}$ and using $C = 4$ the time complexity drops to 2^{32}.

In the known (random) IV model the data complexity increases by a factor of N, which is the expected number of IVs until a "good" one is found. With the above parameter setting the data complexity for a known IV attack grows to 2^{25} whereas the time complexity remains the same 2^{48}.

5 The Fork Model

In this section we discuss a situation where many identical instances of RC4 diverge at a certain point, i.e., at a certain point they have the same state

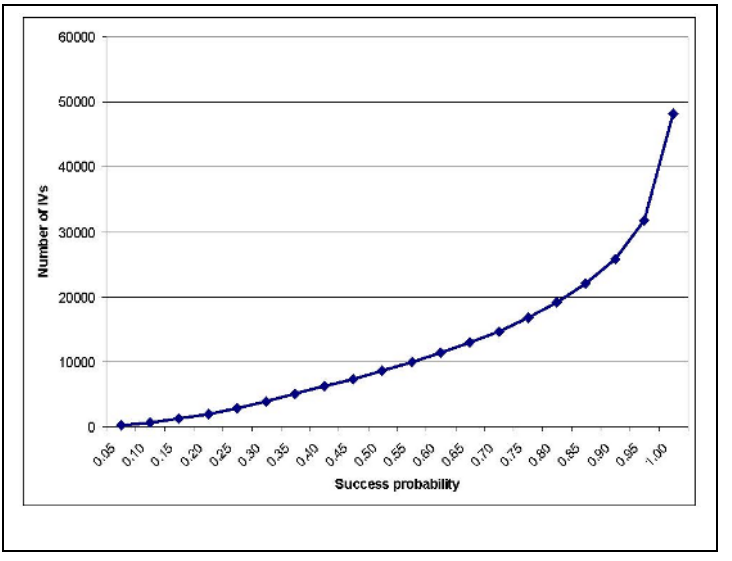

Fig. 3. The number of IVs that are required for different success probabilities (for the attack on RC4 in the fork model)

(permutation and indices) and afterwards some small change to the state occurs, causing each of the instances to evolve differently. A small change in this context may be a change in j and possibly change to a small number of permutation entries. We show that in this model, given a sufficient number of instances the permutation at the divergence point can be recovered.

The attack goes iteratively over the permutation entries and recovers one permutation value at a time. Let S be the permutation in the divergence point and let t be an index for which the attacker wants to reveal $S[t] = x$. The attacker waits until the round where the index i reaches location t and looks at the keystream word that was emitted at that point. If the attacker is lucky, the value x remains in location t until that round (we denote this event as A) and due to the glimpse property the emitted value will be biased towards $i - S[j] = i - x$ (we denote the event where $z = i - x$ by B). Using Lemma 1 (which is proved in Appendix B) we estimate the probability of A with $p_A = 1/e$ and the glimpse property guarantees that the event B occurs with probability $2/N$. Assuming independence of the events and uniform distribution of the output when both event do not occur (when one event does not occur the probability if 0) we get

$$\begin{aligned}
\mathbb{P}[x = i - z] &= \mathbb{P}[A, B] + \mathbb{P}[\neg A, \neg B] \cdot 1/N = \\
&= 2/N \cdot p_A + (1 - 2/N) \cdot (1 - p_A) \cdot 1/(N - 1) \approx \\
&\approx 2/N \cdot p_A + (1 - p_A) \cdot 1/N = \\
&= 1/N \cdot (1 + p_A) \approx \\
&\approx 1/N \cdot (1 + 1/e) \qquad (13)
\end{aligned}$$

This probability was verified through simulations and indeed the correct x value has a significant advantage on other guesses. Through using a voting mechanism where votes are given to values $i - z$, the correct value of x is expected to noticeably stick out. In Fig. 3 we analyze the number of iterations that are required for recovery of x under these circumstances. After 20,000 iterations, every permutation entry is recovered with success probability of 80%. The iterative step is repeated for each of the permutation entries and under reasonable assumptions of independency the same data can be reused for each of the locations.

Notice that in the case where 80% of the guesses are correct, there are still 50 permutation entries that are guessed incorrectly. However, the attacker can avoid guesses that have only small advantage and use only those with high level of confidence. As was shown in [KM+98] having a significant part of the permutation provides the critical mass for completion of the state recovery task.

6 Attacking the Succeeding IV Mode

While presenting a practical key recovery attack for the preceding IV mode, Fluhrer et-al only showed is [FMS01] several sets of weak keys for the succeeding IV mode.

However, this mode of operation "almost" realizes the fork model, where the first rounds of the KSA use an identical part of the key (the root key) whereas the following rounds use different part of the key (IVs). The "almost" is due to the fact that the KSA does not output words and thus the first leakage occurs only in the beginning of the PRGA, i.e., after $N - \ell_K$ rounds that ruin $N - \ell_K$ entries from the divergence permutation.

However, this hurdle can be overcome through appropriate adjustments. In order to reveal a single permutation entry, the attacker can direct the leakage of this value to a fixed location ℓ_K, which leaks through the ℓ_K^{th} keystream word.

In every step of the attack, the attacker fixes $IV[0]$ and uses varying values for the rest of the IV. After ℓ_K rounds of using words of the root key, The index j in round ℓ_K depends on the "keyword" $IV[0]$ in an additive manner and thus every value $IV[0]$ implies a different j_{ℓ_K}, a different value in location ℓ_K after round ℓ_K and eventually a leakage of a different permutation entry to the ℓ_K^{th} keystream word.

Thus for every value of $IV[0]$ a new value leaks to the keystream. Notice that the keywords are used in an additive manner and thus any increase of $IV[0]$ causes a similar increase in j at the corresponding round and eventually the attacker learns the permutation at the divergence point, but with a fixed shift that depends on the unknown j_{ℓ_K-1}, and needs to try all possible 256 shifts in order to recover the correct permutation. Notice that j_{ℓ_K-1} is unknown at this stage and thus every step of the attack (with a fixed $IV[0]$) exposes a permutation entry from an unknown location. Thus the attacker needs to try all possible values for j_{ℓ_K-1} in order to complete the recovery of the permutation.

Since every stage of the attack needs IVs with different $IV[0]$, data cannot be reused for the different stages and a multiplicative factor of N should be

considered when evaluating the amount of data that is needed for the attack, i.e., instead of recovering the permutation with 2^{14} keystreams and IVs the attack needs 2^{22} keystreams and IVs. The number of steps is proportional to the amount of IVs.

Notice that the attack is somewhat wasty as it always works with one location out of the ℓ_K locations that leak information to the keystream. This attack can thus be further optimized for at least partial reuse of the data. The optimized attack uses only N/ℓ_K values for $IV[0]$ that have additive differences of ℓ_K between them, and each of these values is reused for recovering ℓ_K permutation entries. This optimization improves the data complexity of the attack by a factor of ℓ_K and thus for a 16-byte key, the data complexity of the attack drops to 2^{20}. However, this optimization works only in the chosen IV model.
The last step of the attack is a recovery of the root key from the permutation. An efficient implementation of this stage is described in the appendix of [FMS01].

7 Fault Attack on RC4

In this section we describe a fault attack on RC4 that is based on realization of the fork model.

7.1 The Attack Model

We assume that the attacker can apply several types of faults to the cryptographic device; In a data fault the attacker causes some bit flipping changes to RAM or internal registers. In a flow Fault the attacker causes small changes to the flow of the executed program, e.g., skipping an instruction, changing the address of accessed memory, etc.

Following [HS04] we assume that the attacker has only limited control over the fault, that he can select the fault area but not a particular bit and that he has no knowledge on which fault eventually occurred and when exactly had it happened. As usually assumed in fault analysis, we assume that the attacker can reset the system with the same key, i.e., cause the system to get back to the original configuration, cancelling the previously made faults and reuse the same key. This model is somewhat conservative, but more realistic than a model where the attacker is more powerful.

7.2 The Attack

The objective of the attack is to recover the initial permutation of RC4 S_0, which is the output of the KSA and the input of the PRGA. Other permutations can be recovered through similar approach. The attacker injects to the PRGA process faults that change the progression of j, where in order to do that, the attacker needs to inject either a fault to j or a fault to one of the entries of S that are located closely after the index i.

The identical part of the instances is the execution until the fault and the divergence is in the fault. By reusing the analysis from Sect. 5 we conclude that the number of faults that are required for recovery of the state that precede the fault is 2^{14}.

8 Summary

In this paper we presented several new attacks of RC4, all relying on a combination of a leakage of state information to the keystream with a slow evolution of the state, both of which are inherent properties of RC4 fundamental mechanisms. Since the leakage is from a "moving target" part of the state we could not exploit it to attack the keystream generation of RC4 and the applicability of the attack is limited to particular modes of operation.

We proved the common belief that throwing 256 words removes all the vulnerabilities of RC4 initialization to be faulty by showing that the preceding IV mode remains weak even in this case. Despite of the fact that the attack is applicable only for a particular key-IV combination method, we believe that similar attacks on equivalent key-IV combination methods such as exclusive-or and succeeding IV, are not out of reach. RC4 KSA is intolerably sensitive to related key analysis and minimal control is sufficient for an attacker to direct this leakage to desired places.

RSA Security recommends in [RSA01] on employing *at least one* of omitting 256 bytes and employing stronger key-IV combination method. From our findings this recommendation turns to be insufficient as it "allows" modes of operation that are completely insecure. Our recommendation is to avoid using RC4 without employing both strengthening methods or at least to throw a longer prefix of the keystream as proposed by Mironov in [Mir02] .

In addition, we presented attacks on the succeeding IV mode than are stronger than previously known ones and a new fault attack that is comparable to known ones in its complexity.

References

[Fin94] Hal Finney: An RC4 Cycle that Can't Happen. (1994)

[Roo95] Andrew Roos: A Class of Weak Keys in the RC4 Stream Cipher. Posted to sci.crypt (1995)

[Jen96] Robert J. Jenkins: Isaac and RC4. http://burtleburtle.net/bob/rand/isaac.html.

[Gol97] Jovan Dj. Golić: Linear Statistical Weakness of Alleged RC4 Keystream Generator. EUROCRYPT 1997: 226-238

[KM+98] Lars R. Knudsen, Willi Meier, Bart Preneel, Vincent Rijmen and Sven Verdoolaege: Analysis Methods for (Alleged) RC4. ASIACRYPT 1998: 327-341

[MT98] Serge Mister, Stafford E. Tavares: Cryptanalysis of RC4-like Ciphers. Selected Areas in Cryptography 1998: 131-143

[Gol00] Jovan Dj. Golić: Iterative Probabilistic Cryptanalysis of RC4 Keystream Generator. ACISP 2000: 220-233

[GW00] Alexander L. Grosul and Dan S. Wallach: a Related-Key Cryptanalysis of RC4. Technical Report TR-00-358, Department of Computer Science, Rice University (2000)
[FM00] Scott R. Fluhrer and David A. McGrew: Statistical Analysis of the Alleged RC4 Keystream Generator. FSE 2000: 19-30
[MS01] Itsik Mantin and Adi Shamir: A Practical Attack on Broadcast RC4, FSE 2001: 152-164
[FMS01] Scott R. Fluhrer, Itsik Mantin and Adi Shamir: Weaknesses in the Key Scheduling Algorithm of RC4. Selected Areas in Cryptography 2001: 1-24
[Man01] Itsik Mantin: The Security of the Stream Cipher RC4. Master Thesis (2001) The Weizmann Institue of Science
[SIR01] Adam Stubblefield, John Ioannidis, Aviel D. Rubin: Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. NDSS 2002
[RSA01] RSA Security Response to Weaknesses in Key Scheduling Algorithm of RC4. Technical Report, RSA Data Security (2001)
[Mir02] Ilya Mironov: (Not So) Random Shuffles of RC4. CRYPTO 2002: 304-319
[PP03] Souradyuti Paul, Bart Preneel: Analysis of Non-fortuitous Predictive States of the RC4 Keystream Generator. INDOCRYPT 2003: 52-67
[PP04] Souradyuti Paul, Bart Preneel: A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. FSE 2004: 245-259
[Zol04] Bartosz Zoltak: VMPC One-Way Function and Stream Cipher. FSE 2004: 210-225
[HS04] Jonathan J. Hoch, Adi Shamir: Fault Analysis of Stream Ciphers. CHES 2004: 240-253
[BGN05] Eli Biham, Louis Granboulan, Phong Q. Nguyen: Impossible Fault Analysis of RC4 and Differential Fault Analysis of RC4. FSE 2005: 359-367
[Max05] Alexander Maximov: Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of RC4 Family of Stream Ciphers. FSE 2005: 342-358
[Man05] Itsik Mantin: Predicting and Distinguishing Attacks on RC4 Keystream Generator, EUROCRYPT 2005: 491-506
[TS+05] Yukiyasu Tsunoo, Teruo Saito, Hiroyasu Kubo, Maki Shigeri, Tomoyasu Suzaki, and Takeshi Kawabata: The Most Efficient Distinguishing Attack on VMPC and RC4A, SKEW 2005

A Availability of the Glimpse

The existence of the glimpse stems from the usage of permutation access of depth two when selecting the output value. In Conjecture 1 we generalize the glimpse in a different direction than Theorem 2 and claim that the glimpse will exist for almost any output selection function of depth two.

We begin with defining a general output selection function. Let $f, g : [N] \rightarrow [N]$ be invertible functions and denote the corresponding inverse functions by F and G respectively. Let $h : [N] \times [N] \rightarrow [N]$ be a 2-parameter function that is invertible in each of its parameters, and let H_1 and H_2 be the inverse functions of h where

$$\forall X, Y \in [N],\ H_1(X, h(X, Y)) = Y, H_2(h(X, Y), Y) = X$$

Conjecture 1 (The Generalized Glimpse Conjecture).
Let $Z(S,i,j) \stackrel{def}{=} f(S[h(S[g(i)],S[j])])$ be an output selection function of an RC4-like keystream generator. Then,

$$\mathbb{P}_{j\in_R[N],S\in_R\mathcal{P}[N]}[S[j] = H_1(F(Z(S,i,j)),g(i))] \approx 2/N \tag{14}$$

We will give the intuition behind Conjecture 1. In order to simplify the expressions we define $Z' \stackrel{def}{=} F \circ Z$ and $i' \stackrel{def}{=} g(i)$ and thus given that $Z'(S,i,j) \stackrel{def}{=} S[h(S[i'],S[j])]$ we need to show that

$$\mathbb{P}_{j\in_R[N],S\in_R\mathcal{P}[N]}[S[j] = H_1(Z'(S,i,j),i')] \approx 2/N \tag{15}$$

We define two functions over the domain of RC4 states and two corresponding events. The internal dependency function $IDF(S,i',j)$ is defined as $h(S[i'],S[j])$ and the event where $i' = IDF(S,i',j)$ is denoted as A_{IDF}. The external dependency function $EDF(S,i',j)$ is defined as $H_1(Z'(S,i',j),i')$ and the event where $S[j] = EDF(S,i',j)$ is denoted as A_{EDF}. Our arguments follow the original proof of the glimpse.

We observe two cases. In the first case A_{IDF} occurs. In that case A_{EDF} occurs with probability 1 in the same manner as the original glimpse

$$z' \stackrel{def}{=} Z'(S,i,j) = S[h(S[i'],S[j])] = S[i'] \tag{16}$$

$$i' = h(S[i'],S[j]) = h(z',S[j]) \tag{17}$$

$$S[j] = H_1(z',i') = H_1(F(Z(S,i,j)),g(i)) \tag{18}$$

In the other case, $IDF(S,i,j)$ is almost random and with the uncertainty in S and j causes a distribution that is very close to uniform for z'. Thus the probability of A_{EDF} is $1/N \cdot 1 + (1-1/N) \cdot 1/N \approx 2/N$.

B RC4 State Evolution

RC4 permutation evolves fairly rapidly with the generation, where on every round two values change locations. The index i progresses in a predictive manner traversing the permutation sequentially and thus guarantees that no location or value is left untouched during a sequence of N rounds (it is possible that a value is swapped with itself). The index j adds pseudo-randomness to the state progression by jumping between the permutation entries in a seemingly unpredicted manner. However, when concentrating on a sequence that is shorter than N rounds, there are permutation entries which are guaranteed not to be visited by the index i, and these entries have relatively high probability not to be touched also by j during this sequence of rounds.

We formalize this situation in Lemma 1 and quote its proof from [Man05].

Lemma 1 (The Evolution Lemma). *Let $\mathcal{I}$ be a set of r permutation locations. Suppose that RC4 is in a state where the predictable course of the index i in the next k rounds avoids visiting $\mathcal{I}$. Then the probability of the permutation S k rounds later to have the values in $\mathcal{I}$ unchanged is approximately $e^{-kr/N}$.*

Proof. The index i does not reach any of the indices in $\mathcal{I}$. The index j progresses in a pseudo-random manner and reaches each of the r positions in each of the k rounds with probability $1/N$. Failing in these kr trials results with having the set $\mathcal{I}$ untouched and the probability of this event to occur is $(1-1/N)^{kr} \approx e^{-kr/N}$. □

In the special case where $r = 1$ and $k \leq N$ we have a bound of $1/e$ for the probability of a single value, located more than k entries ahead of i to remain in place during the following k rounds.