On Reconstruction of RC4 Keys from Internal States

Shahram Khazaei[1] and Willi Meier[2]

[1] EPFL, Lausanne, Switzerland
[2] FHNW, Windisch, Switzerland

Abstract. In this work key recovery algorithms from the known internal states of RC4 are investigated. In particular, we propose a bit-by-bit approach to recover the key by starting from LSB's of the key bytes and ending with their MSB's.

Keywords: Binary Hypothesis Testing, Stream Ciphers, Key Recovery, RC4.

1 Introduction

Synchronous stream ciphers are symmetric cryptosystems which are suitable in software applications with high throughput requirements, or in hardware applications with restricted resources (such as limited storage, gate count, or power consumption). RC4 is probably the most popular stream cipher in use. In this work we are going to investigate the key recovery algorithms from the known internal states of RC4. Roos in 1995 [6] noticed that some of the elements of the initial permutations have a bias towards a linear combination of the secret key bytes. A theoretical proof of these biases was given by Paul and Maitra [5] which was later generalized by Biham and Carmeli [2]. In [5,2] the authors also provide algorithms for key reconstruction from the internal state using the derived biases. However the algorithms from [5] have high complexities, low success probabilities and the one from [2] has low complexity, and still low success probability. In addition, the authors of [2] did not analyze the complexity of their algorithm for having a higher success probability. In fact, the newly found generalized biases have not been exploited to the degree they deserve in the key recovery algorithm of [2]. The main idea of our work is to *fully* exploit the whole distribution of noises expressing these biases. In a hypotheses testing model, we then study how far one can go by using only the distribution of noises. Having carefully analyzed the noise distributions, we then propose a *bit-by-bit* approach to recover the key bits by starting from LSB's of the key bytes and ending with their MSB's. The nice feature of our algorithm is that we are able to estimate its complexity versus success probability, showing possibility of recovering the key with high success probability but reasonable time complexity.

2 Description of RC4 and Notations

RC4 is composed of a Key Scheduling Algorithm (KSA) and a Pseudo Random Generation Algorithm. It works with the set of integers $\mathbb{Z}_N = \{0, ..., N-1\}$ and its internal state is a permutation $S = (S[0], \dots S[N-1])$ over $\mathbb{Z}_N$ ($N = 256$ in practice). RC4 uses keys $k = (k[0], \dots, k[l-1])$ of length l (typically $5 \le l \le 16$) over $\mathbb{Z}_N$. The KSA computes the internal state from the key k to be used by the PRGA in order to produce a keystream sequence of integers over $\mathbb{Z}_N$, see Alg. 1 and 2.

J. Calmet, W. Geiselmann, J. Müller-Quade (Eds.): Beth Festschrift, LNCS 5393, pp. 179–189, 2008.

Notations: $K = (K[0], \ldots, K[N-1])$ denotes an array of size N over $\mathbb{Z}_N$ such that $K[i] = k[i \mod l]$ for $0 \leq i \leq N-1$. The value of the array S right after the KSA is denoted by S_N and the array $C = (C[-1], \ldots, C[N-1])$ over $\mathbb{Z}_N$ is defined as $C[-1] = 0$ and $C[i] = S_N[i] - \frac{i(i+1)}{2}$ for $0 \leq i \leq N-1$. For an array A we use the notation $A[i,j] = \sum_{t=i}^{j} A[t]$.

Algorithm 1. RC4 KSA

```
for $i = 0, 1, \ldots, N-1$ do
    $S[i] = i$.
end for
$j = 0$.
for $i = 0, 1, \ldots, N-1$ do
    $j = j + S[i] + K[i]$.
end for
```

Algorithm 2. RC4 PRGA

```
$i = 0$.
$j = 0$.
repeat
    $i = i + 1$.
    $j = j + S[i]$.
    Swap $S[i]$ and $S[j]$.
    Output $z = S[S[i] + S[j]]$.
until enought outputs have been produced.
```

The goal of the attacker is to determine the secret key out of a known segment of keystream. This can be done in two steps: first to determine a state out from the keystream segment, and in a second step to determine the key out of the internal state. Note that the initial state S_N can be easily computed given any intermediate internal state at any time during the PRGA. In this paper we only deal with the second step, *i.e.* recovering the secret key k from a given initial state S_N (or equivalently from array C). The interested reader is referred to [4] for the best known attack on the first step, *i.e.* recovering the internal state from the known keystream segment. Another type of attack on RC4 is key recovery attack when the secret key contains a known initialization vector part and the attacker has access to the keystreams of many (chosen) initialization vectors for the same unknown key part, see [8] for a recent attack.

3 Previous Results

Roos in 1995 [6] noticed that some of the elements of the initial permutations have a bias towards a linear combination of the secret key bytes. A theoretical proof of these biases was given by Paul and Maitra [5], later generalized by Biham and Carmeli [2]. Thanks to our choice $C[-1] = 0$, these results can be given in a unified theorem as follows.

Theorem 1. *Assuming that during the KSA the pseudo-random index j takes its values uniformly at random from $\mathbb{Z}_N$, for every $-1 \leq i_1 < i_2 \leq N-1$ then the probability that the following equation holds*

$$C[i_2] - C[i_1] = K[i_1+1, i_2] \tag{1}$$

is at least p_{i_1,i_2} where

$$p_{-1,i_2} = (1-\frac{i_2}{N}) \cdot (1-\frac{1}{N})^{\frac{i_2(i_2+1)}{2}+N} + \frac{1}{N} \tag{2}$$

for $0 \leq i_2 \leq N-1$ [5] and

$$p_{i_1,i_2} = (1-\frac{i_2}{N})^2 \cdot (1-\frac{i_2-i_1+2}{N})^{i_1} \cdot (1-\frac{2}{N})^{N-i_2-1} \prod_{r=0}^{i_2-i_1-1} (1-\frac{r+2}{N}) + \frac{1}{N} \tag{3}$$

for $0 \leq i_1 < i_2 \leq N-1$ [2].

In [5] the probabilities in Eq. (2) are used to retrieve the secret key of RC4. The algorithm uses equations of (1) for $i_1 = -1$ (which in this case is simplified as $C[i_2] = K[0, i_2]$) in the following way. For each combination of m independent equations out of the first n equations (*i.e.* $1 \leq i_2 \leq n$), the algorithm exhaustively guesses the value of $l-m$ key bytes, and solves the m equations to recover the remaining key bytes. The success of the the algorithm depends on the existence of m correct and linearly independent equations among the first n equations. The success probabilities and the running time of the the algorithm for different key sizes and some choices for the parameters m and n are presented in Table 1 (taken from [2], see also footnote 1 therein).

In [2] the probabilities in Eq. (3) along with Eq. (1) are used in a more sophisticated way which lead to a better key recovery algorithm. The results can be seen in Table 2. Very recently, Akgün, Kavak and Demirci [1] have developed new biases for RC4, combined them with previous results and provided a new key recovery algorithm from the internal state. It performs better than existing ones including ours. In particular, in a theorem similar to Theorem 1, they provide a lower bound for the probability that $C'[i_2] - C'[i_1] = K[i_1+1, i_2]$ for $0 \leq i_1 < i_2 \leq N-1$ where $C'[i] = S_N^{-1}[i] - \frac{i(i+1)}{2}$ for $0 \leq i \leq N-1$. In this paper we have only used the biases suggested by Theorem 1 since our work had been finalized before having been aware of [1]. Yet [1] *does not*

Table 1. Success probability and running time of the algorithm from [5] according to [2]

l	n	m	P_{succ}	Time
5	16	5	0.250	2^{18}
5	24	5	0.385	2^{21}
8	16	6	0.273	2^{34}
8	20	7	0.158	2^{29}
8	40	8	0.092	2^{33}
10	16	7	0.166	2^{43}
10	24	8	0.162	2^{40}
10	48	9	0.107	2^{43}
12	24	8	0.241	2^{58}
12	24	9	0.116	2^{50}
16	24	9	0.185	2^{60}
16	32	10	0.160	2^{63}
16	32	11	0.086	2^{64}
16	40	12	0.050	2^{64}

Table 2. Success probability and running time (in seconds) of the algorithm from [2] compared to [5]

l	P_{succ}	Time [2]	Time [5]
5	0.8640	0.02	366
8	0.4058	0.60	2900
10	0.0786	1.46	183
10	0.1290	3.93	2932
12	0.0124	3.04	100
12	0.0212	7.43	1000
16	0.0005	278	500

fully exploit the distribution of the noises, leaving a room to unify the biases from [1] with those from Theorem 1 in a future work.

4 A Hypotheses-Testing Approach to the Problem

Each of the equations (1) gives us a noisy value of $K[i_1+1, i_2]$ which is a linear combination of the key values $k[i]$. Let assume that $e_{i_1,i_2} \in \mathbb{Z}_N$, $-1 \le i_1 < i_2 \le N-1$, denotes the noise of each equation, *i.e.*

$$K[i_1+1, i_2] = C[i_2] - C[i_1] + e_{i_1,i_2} \ . \tag{4}$$

The random variables corresponding to these noises, denoted by E_{i_1,i_2}, are not independent (see proofs of Theorem 1 in [5,2]). Moreover Theorem 1 does not completely characterize their probability distribution, since it only suggests $p_{i_1,i_2} = \Pr\{E_{i_1,i_2} = 0\}$ (we ignore the inequality). However, it is reasonable to assume that E_{i_1,i_2} takes other values with equal probabilities, *i.e.* $\Pr\{E_{i_1,i_2} = e\} = \frac{1-p_{i_1,i_2}}{N-1}$ for $e \in \mathbb{Z}_N^\star$.

With this new view on the problem we try to recover the key in a correlation based attack by taking a hypotheses-testing approach. This can be seen as a generalization of the original correlation attack on binary LFSR's by Siegenthaler [7]. First, we assume an attacker with an unlimited amount of computational power, capable of making an exhaustive search over all N^l possible keys. Like [7], we make the assumption that for a wrong guess $\bar{k}$ (or equivalently the corresponding $\bar{K}$) of the key, the values of $C[i_2] - C[i_1]$ and $\bar{K}[i_1+1, i_2]$ are uncorrelated. Under this assumption, we are facing the following binary hypothesis testing problem. Given $N(N+1)/2$ samples of $e_{i_1,i_2} = \bar{K}[i_1+1, i_2] - (C[i_2] - C[i_1])$, $-1 \le i_1 < i_2 \le N-1$, as a realization of the random variables E_{i_1,i_2}, decide if the guess $\bar{k}$ is correct. Our ability in distinguishing between a correct key ($\bar{k} = k$) from a wrong key ($\bar{k} \ne k$) depends on the following two distributions:

$$H_0 : \bar{k} = k, \quad \Pr\{E_{i_1,i_2} = e | H_0\} = \begin{cases} p_{i_1,i_2} & e = 0 \\ \frac{1-p_{i_1,i_2}}{N-1} & e \in \mathbb{Z}_N^\star \end{cases} \tag{5}$$

$$H_1 : \bar{k} \ne k, \quad \Pr\{E_{i_1,i_2} = e | H_1\} = \frac{1}{N}, \forall e \in \mathbb{Z}_N \ . \tag{6}$$

The quality of a decision rule (distinguisher) is related to two kinds of error probabilities: *false alarm probability* $p_{\text{fa}} = \Pr\{H_0|H_1\}$ and *non-detection probability*

$p_{\text{nd}} = \Pr\{H_1|H_0\}$. Ideally, we would like to minimize both error probabilities but normally there is a trade-off. The optimum decision rule is given by *Neyman-Pearson lemma* [3]. It can be shown that for the hypothesis testing problem given by Eq. (5) and (6), the optimum decision rule chooses H_0 if $M(S_N, \bar{k}) > T$ and selects H_1 otherwise, where

$$M(S_N, \bar{k}) = \sum_{-1 \leq i_1 < i_2 \leq N-1} \log \frac{(N-1)p_{i_1,i_2}}{1 - p_{i_1,i_2}} \cdot \delta(e_{i_1,i_2}) \tag{7}$$

and $\delta : \mathbb{Z}_N \to \{0,1\}$ being the Dirac delta function (*i.e.* $\delta(e) = 1$ iff $e = 0$). Remember that $e_{i_1,i_2} = \bar{K}[i_1+1, i_2] - (C[i_2] - C[i_1])$ only depends on S_N and $\bar{k}$. The parameter T determines the false alarm and non-detection probabilities. More precisely we have,

$$p_{\text{fa}} = \Pr\{M(S_N, \bar{k}) > T | \bar{k} \neq k\} \tag{8}$$

and

$$p_{\text{nd}} = \Pr\{M(S_N, \bar{k}) \leq T | \bar{k} = k\}\ . \tag{9}$$

4.1 Complexity Analysis

Since there are $N(N+1)/2$ terms in the sum (7), the complexity of exhaustive search algorithm is $\frac{1}{2}N(N+1)N^l$. As it was noticed in [2] the sum of all the key elements, *i.e.* $s = k[0, l-1]$ is quite useful for reducing the complexity. In this section we will show how we can reduce complexity to $\frac{1}{2}l(l+1)N^l$, using $Nl(l+1)/2$ memory. The idea is based on the following relations

$$K[i_1+1, i_2] = (q_2 - q_1) \cdot s + \begin{cases} \sum_{t=r_1}^{r_2} k[t] & if\ r_1 \leq r_2\ \&\ (r_1, r_2) \neq (0, l-1) \\ 0 & if\ r_1 = r_2 + 1 \\ -\sum_{t=r_2+1}^{r_1-1} k[t] & if\ r_2 + 1 \leq r_1 - 1 \\ s & if\ (r_1, r_2) = (0, l-1) \end{cases} \tag{10}$$

for $-1 \leq i_1 < i_2 \leq N-1$ where $r_1 = (i_1 + 1 \mod l)$, $q_1 = \lfloor \frac{i_1+1}{l} \rfloor$, $r_2 = (i_2 \mod l)$ and $q_2 = \lfloor \frac{i_2}{l} \rfloor$. Eq. (10) suggests that $K[i_1+1, i_2]$ can be written as $u(s, i_1, i_2) + \alpha k[r_1, r_2]$ where $\alpha \in \{-1, 0, 1\}$ and u being a function of $s = k[0, l-1]$, i_1 and i_2. It then follows that Eq. (7) can be written as follows:

$$M(S_N, \bar{k}) = v_{0,l-1}(\bar{s}, S) + \sum_{\substack{0 \leq r_1 \leq r_2 \leq l-1 \\ (r_1, r_2) \neq (0, l-1)}} v_{r_1,r_2}(\bar{k}[r_1, r_2], \bar{s}, S), \tag{11}$$

with v_{i_1,i_2}, $0 \leq r_1 \leq r_2 \leq l-1$ being some known functions and $\bar{s} = \bar{k}[0, l-1]$. Once $\bar{s}$ is known, one can precompute and store $v_{0,l-1}$ and all other v_{i_1,i_2}'s for all N possible values of $\bar{k}[i_1+1, i_2]$. This simply suggests an implementation needing a feasible amount of $N\,(l(l+1)/2 - 1) + 1$ memory and $N^l l(l+1)/2$ table look-ups (we ignore the additive time $N^2(N+1)/2$ for evaluating v_{i_1,i_2}'s). Note that $l(l+1)/2$ table look-ups are much faster than direct evaluation of Eq. (7) since $l \ll N$.

4.2 Simulation Results

Our simulation results show that the distributions of $M(S_N, \bar{k})$, given by Eq. (7) or (11), under H_0 and H_1 are separated enough to provide a key recovery attack with high success probability. Note that this is a key recovery attack which ONLY takes advantage of the probabilities under our assumptions which are not totally correct (independence of noises and uniform distribution for noise under H_1). Moreover, our simulations show that if we consider the hypotheses $H_1' : \bar{k} \neq k$ & $\bar{s} = s$, the distributions of $M(S_N, \bar{k})$ under H_0 and H_1' get a bit closer but still separated enough, see Fig. 1.

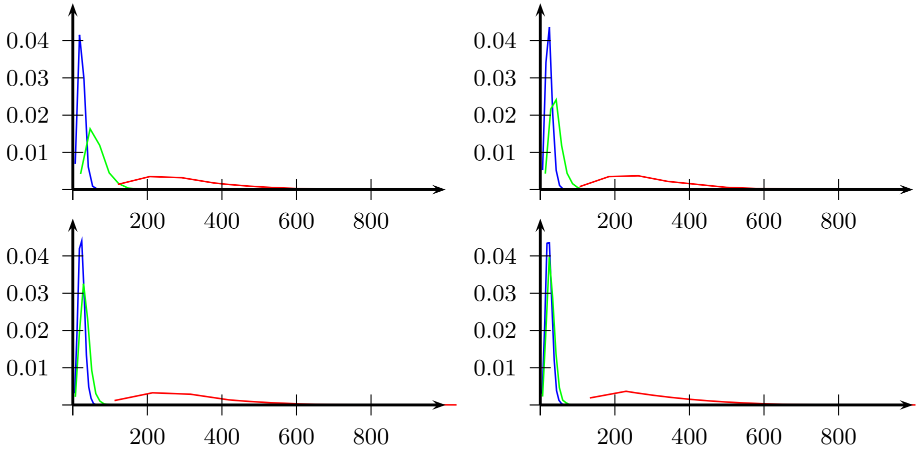

Fig. 1. Empirical distribution of $M(S_N, \bar{k})$ under H_0, H_1 and H_1' (red, blue and green resp.) for $l = 5, 8, 12$ and 16 (up right, up left, down left and down right resp.)

We introduced H_1' to slightly compensate the ideal assumption that the noise under H_1 is uniformly distributed. This also helps to estimate a more exact bound for the success probability of a distinguisher which suggests a small set of candidates for the key (*i.e.* the distinguisher having $p_{\mathrm{fa}} \approx N^{-l}$). In practice it is not possible to do the simulations for this value of p_{fa} due to limited number of samples and therefore a practical value should be chosen. Table 3 shows the simulated values for p_{nd} corresponding to $p_{\mathrm{fa}} \approx 2^{-10}$, and hence an upper bound estimation for the success probability p_{suc}.

We expect that the actual success probabilities be very close to our estimations, especially for larger values of l. The key recovery algorithm of [2] has a much lower success

Table 3. An upper bound estimation for the success probability

l	5	6	7	8	9	10	12	16
p_{nd} (for $p_{\mathrm{fa}} \approx 2^{-10}$)	0.215	0.125	0.055	0.020	0.018	0.010	0.005	0.000
p_{suc}	0.785	0.875	0.945	0.980	0.982	0.990	0.995	1.00

probability though it slightly takes into consideration the dependencies between noises. Also notice that for larger values of l, the success probability increases which again shows that the algorithm of [2] is far from being optimal.

Remark 1. We emphasize that the values of $p_{\mathrm{suc}} = 1 - p_{\mathrm{nd}}$ (for $p_{\mathrm{fa}} \approx N^{-l}$), give the success probability for an algorithm which makes exhaustive search over the key, only uses the measure $M(S_N, \bar{k})$ to identify a small subset of candidates for the key, and more importantly does not use the KSA. To achieve a higher success probability one can allow a higher p_{fa} resulting in a bigger set of candidates for the key which can later be filtered to find the correct one by applying KSA.

We are interested in algorithms for reconstructing the key which avoid computation of the measure $M(S_N, \bar{k})$ for all N^l keys. In the next section we propose an algorithm which starts with an estimate on LSB's of the key bytes and then continues to bits of higher significance.

5 A Bit-by-Bit Approach for Key Recovery

The idea is to take into account the probability distribution of $\Pr\{E_{i_1,i_2} \mod 2^r\}$ instead of $\Pr\{E_{i_1,i_2}\}$ and considering these two hypotheses: $H_0^r : \bar{k} = k \mod 2^r$ and $H_1^r : \bar{k} \neq k \mod 2^r$. Then one can show that Eq. (5) and (6) become as follows

$$H_0^r : \quad \Pr\{E_{i_1,i_2} = e \mod 2^r | H_0\} = \begin{cases} p_{i_1,i_2}^r & e = 0 \\ \frac{1 - p_{i_1,i_2}^r}{2^r - 1} & e \in \mathbb{Z}_{2^r}^{\star} \end{cases} \tag{12}$$

$$H_1^r : \quad \Pr\{E_{i_1,i_2} = e \mod 2^r | H_1\} = \frac{1}{2^r}, \forall e \in \mathbb{Z}_{2^r} \ . \tag{13}$$

where

$$p_{i_1,i_2}^r = p_{i_1,i_2} + \frac{N - 2^r}{2^r(N-1)}(1 - p_{i_1,i_2}) \tag{14}$$

assuming N is a power of 2. Similarly, the measure which should be computed is as below

$$M^r(S_N, \bar{k}) = \sum_{-1 \leq i_1 < i_2 \leq N-1} \log \frac{(2^r - 1)p_{i_1,i_2}^r}{1 - p_{i_1,i_2}^r} \cdot \delta(e_{i_1,i_2} \mod 2^r) \tag{15}$$

having related p_{fa}^r and p_{nd}^r similar to those in Eq. (8) and (9). Note that $M^r(S, \bar{k})$ only depends on the first r LSB's of $\bar{k}$.

Fig. 2 shows the empirical distribution of $M^r(S_N, \bar{k})$ (for $r = 1, 2, \ldots, 8$) under three hypotheses H_0^r, H_1^r and $H_1'^r$ for $l = 16$ ($H_1'^r$ is defined similar to Sect. 4.2). It is clear that the bigger r is, the more separable the distributions become. Hence a tree-based search can reduce the complexity with a huge factor. The idea is to search over all 2^l possible values of the r-th LSB of the key elements, assuming that the first $r-1$ LSB's of the key are known, and choose only N_r out of them with the highest correlation measure $M^r(S, \bar{k})$, given by Eq. (15). The complexity of this tree-based search algorithm is $\mathcal{C} = 2^l(1 + \sum_{i=0}^{R-1} \prod_{j=0}^{R-1} N_j)$ where $R = \lceil \lg_2 N \rceil$. The KSA must

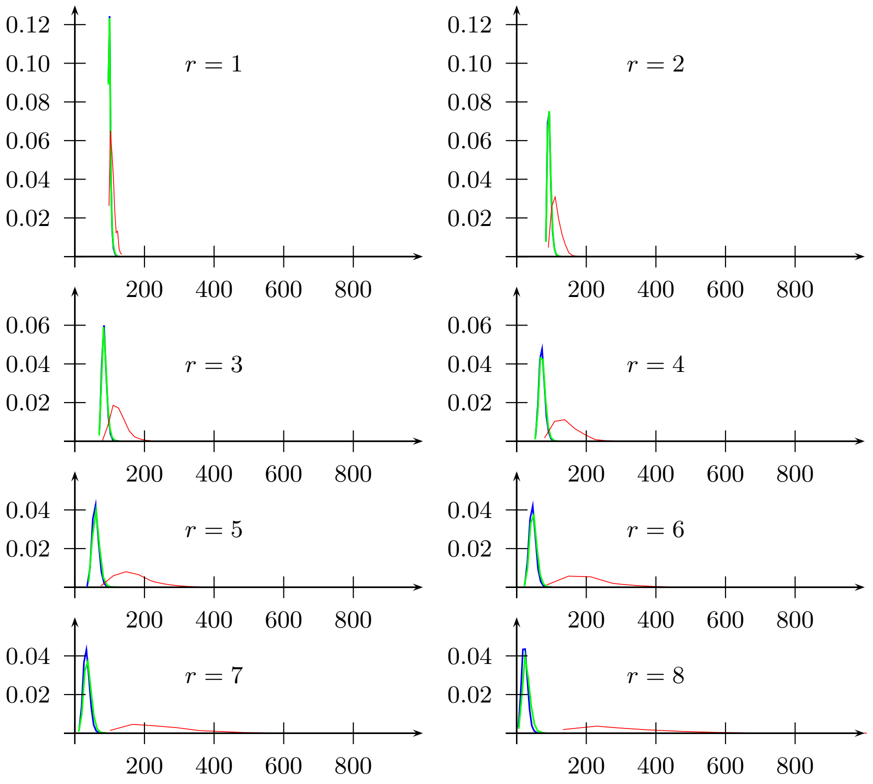

Fig. 2. Empirical distribution of $M^r(S_N, \bar{k})$ (for $r = 1, 2, \ldots, 8$) under H_0^r, H_1^r and $H_1'^r$ (red, blue and green resp.) for $l = 16$

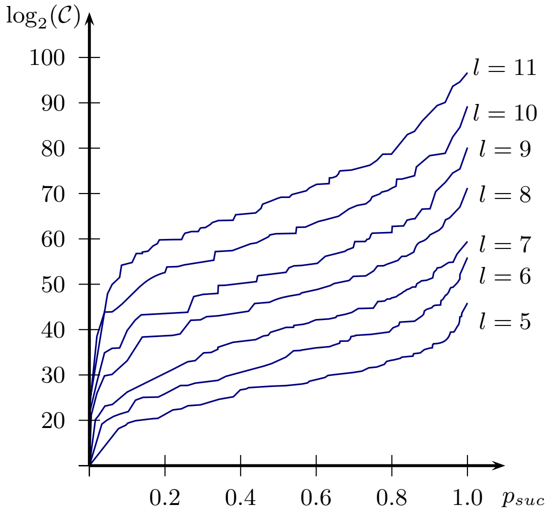

Fig. 3. Empirical complexity for $l = 5, \ldots, 11$ versus success probability

be applied to the $\prod_{i=0}^{R-1} N_r$ key candidates which have reached the final leaves of the tree in order to identify the (possibly) correct one. The complexity of the second step is negligible compared to the first step. The success probability of the attack relies on the parameters N_r's. Fig. 3 shows the complexity of the attack versus success probability for optimized parameters of the attack, for different values of l. Refer to Appendix A to see how these curves have been achieved.

6 Conclusion

A recent statistical weakness in the key initialization of RC4 was used to efficiently recover the key from the internal state. We started by *fully* exploiting the whole distribution of noises expressing these newly found biases in RC4 in a hypotheses testing model. Having carefully analyzed the noise distributions, we proposed a tree-based bit-by-bit approach to recover the key bits. It turned out that the complexity of our algorithm can be empirically computed versus its success probability. Further work is still open thank to the more recently developed biases from [1] which we did not exploit.

References

1. Akgün, M., Kavak, P., Demirci, H.: New Results on the Key Scheduling Algorithm of RC4. Indocrypt (to appear, 2008)
2. Biham, E., Carmeli, Y.: Efficient reconstruction of RC4 keys from internal states. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 270–288. Springer, Heidelberg (2008)
3. Cover, T., Thomas, J.A.: Elements of Information Theory. Wiley series in Telecommunication. Wiley, Chichester (1991)
4. Maximov, A., Khovratovich, D.: New state recovery attack on RC4. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 297–316. Springer, Heidelberg (2008), http://eprint.iacr.org/2008/017
5. Paul, G., Maitra, S.: Permutation after RC4 key scheduling reveals the secret key. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 360–377. Springer, Heidelberg (2007), http://eprint.iacr.org/2007/208.pdf
6. Roos, A.: A Class of Weak Keys in the RC4 Stream Cipher. In: Two posts in sci.crypt. (1995), http://marcel.wanda.ch/Archive/WeakKeys
7. Siegenthaler, T.: Decrypting a class of stream ciphers using ciphertext only. IEEE Transactions on Computers C-34, 81–85 (1985)
8. Vaudenay, S., Vuagnoux, M.: Passive–only key recovery attacks on RC4. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 344–359. Springer, Heidelberg (2007)

A Deriving Optimized Parameters for Tree-Based Bit-by-Bit Search Algorithm

In order to derive the complexity diagram versus success probability, Fig. 3, for the optimized parameters N_r's for the tree-based bit-by-bit search algorithm in Sect. 5 we proceed as follows. For every given key size l, we produce $\mathcal{N}$ random keys $k^i = (k^i[0], \ldots, k^i[l-1])$, $1 \leq i \leq \mathcal{N}$. Then for each key k^i, we compute a vector $N^i = [N^i_0, \ldots, N^i_7]$ where N^i_r, $0 \leq r \leq 7$, satisfies $1 \leq N^i_r \leq 2^l$ and denotes the number of choices of the r-th LSB of $\bar{k}$ (out of 2^l possible choices) having a correlation measure $M^r(S_N, \bar{k})$ greater than $M^r(S_N, k^i)$ provided that $\bar{k}[j] = k^i[j] \mod 2^r$, $0 \leq j \leq l-1$. As it was mentioned in Sect. 5, for given parameters N_r's for the bit-by-bit recovery algorithms, the complexity of the attack is $\mathcal{C} = 2^l(1 + \sum_{i=0}^{R-1} \prod_{j=0}^{R-1} N_j)$ where $R = \lceil \lg_2 N \rceil$. The success probability of the attack can then be estimated as the percentage of the samples for which $N^i_r \leq N_r$, $\forall\ 0 \leq r \leq 7$. However, the parameters N_r's may not be optimal and a better choice for them may exist having less time complexity while providing the same success probability. In our simulation we chose $\mathcal{N} = 1000$ and we tried to find the optimal parameters using a simulated-annealing-like procedure.

B Improved Recovery of Sum of the Key Elements

In [2] a method has been proposed to recover s, the sum of the key elements. Our simulations show that using the optimal measure $v_{0,l-1}(\bar{s}, S)$ slightly improves the results. Table 4 gives the probabilities that the measure $v_{0,l-1}(\bar{s}, S)$ suggest that s has the i-th highest measure ($i = 1, 2, 3, 4$) along with results from [2].

Table 4. Probabilities that s has any of the first four highest measure

l	Measure	Highest	Second highest	Third highest	Fourth highest
5	$v_{0,l-1}$	0.888	0.041	0.012	0.016
	[2]	0.8022	0.0618	0.0324	0.0195
8	$v_{0,l-1}$	0.641	0.064	0.042	0.023
	[2]	0.5428	0.1373	0.0572	0.0325
10	$v_{0,l-1}$	0.539	0.091	0.044	0.025
	[2]	0.4179	0.1604	0.0550	0.0332
12	$v_{0,l-1}$	0.441	0.070	0.051	0.041
	[2]	0.3335	0.1618	0.0486	0.0287
16	$v_{0,l-1}$	0.279	0.070	0.039	0.026
	[2]	0.2309	0.1224	0.0371	0.0240

C Potential Improvements

The tree-based bit-by-bit search algorithm still has some potential for improvements. For example one can imagine a path-ranking on the tree according to their correlation measures and start proceeding on tree from the ones with highest correlation measure at each step. Another idea could be just to ignore some branches in middle if their correlation measure is less than some threshold value. Although these techniques can definitely improve the average time complexity for a given success probability, they are harder to analyze. Another way to improve the bit-by-bit search algorithm is to first recover the sum of the key elements, and then to use the same method. This way the attack complexity reduces by a factor of about 2^8.

One can also consider the problem as an optimization problem and apply the known methods like genetic algorithm, etc. These methods can easily converge to a key $\hat{k}$ that maximizes $M(S_N, k)$. Our simulations show that the achieved value $M(S_N, \hat{k})$ is almost always much greater than the correct one $M(S_N, k)$. The reason is that there is very small fraction of the keys which have a measure greater than correct key. The fraction is so small that they do not show up in the simulation which provides Fig. 1 and as a result we have separated curves. However, once we use optimization algorithms, it always end up with one of these false keys with highest amount of measure M. Although it is very unlikely that the resultant key $\hat{k}$ be the same as correct key k, but they are quite correlated. For example, usually at least one of the elements of $\hat{k}$ and k are the same. It is an open question if we can somehow try to end up with the correct key.